# Dimorphic Computing

**H. Andrés Lagar-Cavilla[†], Niraj Tolia[⋆], Rajesh Balan[⋆], Eyal de Lara[†], M. Satyanarayanan[⋆], David O'Hallaron[⋆]**
**[†]University of Toronto, [⋆]Carnegie Mellon University**

April 2006
CMU-CS-06-123

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

**Abstract**

*Dimorphic computing* is a new model of computing that switches between thick and thin client modes of execution in a completely automated and transparent manner. It accomplishes this *without* imposing any language or structural requirements on applications. This model greatly improves the performance of applications that alternate between phases of compute- or data-intensive processing and intense user interaction. For such applications, the thin client mode allows efficient use of remote resources such as compute servers or large datasets. The thick client mode enables crisp interactive performance by eliminating the harmful effects of Internet latency and jitter, and by exploiting local graphical hardware acceleration. We demonstrate the feasibility and value of dimorphic computing through *AgentISR*, a prototype that exploits virtual machine technology. Experiments with AgentISR confirm that the performance of a number of widely-used scientific and graphic arts applications can be significantly improved without requiring any modification.

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. All unidentified trademarks mentioned in the paper are properties of their respective owners.

**Keywords:** interactive response, network latency, network delay, network bandwidth virtual machines, thin client, thick client, graphics hardware, agents, migration, Maya, QuakeViz, ADF, Kmenc15

# 1 Introduction

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.

In this paper, we describe a new model of computing called *dimorphic computing* that combines the strengths of thick and thin clients. During resource-intensive application phases, a dimorphic client behaves like a thin client. During interaction-intensive phases, it behaves as a thick client. Transitions are completely transparent to the user, who just sees excellent performance at all times. Dimorphic computing does not require that applications be modified in any way or that they be written in any specific programming language. It can be used with closed-source legacy applications, an important distinction from language-based mobile code approaches using Java or C#. Further, dimorphic computing does not restrict applications to a specific operating system.

This new computing model is intended for a growing class of applications that alternate between a resource-intensive *crunch* phase that involves little or no user interaction, and a *cognitive* phase that is intensely interactive. Examples of these applications include digital animation and video editing in amateur and professional movie production, simulation and visualization of phenomena in scientific computing, computer assisted design in engineering, protein modeling for drug discovery in the pharmaceutical industry, and computer-aided diagnosis in medicine. Dimorphic computing optimizes performance for both the crunch and the cognitive phase of such applications. The user thus gets the best of both worlds: short completion time for the crunch phase, and crisp interactive performance in the cognitive phase.

We have implemented a prototype called *AgentISR* that demonstrates the feasibility and value of dimorphic computing. Our design uses virtual machine (VM) technology to encapsulate application execution, a peer-to-peer distributed storage system to transport VM state between application execution sites, a graphical interface capable of providing VMs with graphics hardware acceleration, and a migration manager for automating relocation decisions. Experiments with AgentISR confirm that a number of widely-used applications from the scientific and graphic arts domains, some commercial and closed-source, can be run on AgentISR to yield good performance in both the crunch and cognitive phases.

This paper makes four contributions. First, it introduces the concept of dimorphic computing and shows how it fills a gap in today's computing landscape. Second, it shows the feasibility and value of dimorphic computing by describing a prototype implementation and experimental results from its use on a number of demanding applications. Third, it introduces a new approach to quantifying the interactive performance of tightly-coupled applications under conditions of high latency. Fourth, it describes a new tool for conducting repeatable experiments of interactive user sessions on tightly-coupled applications.

# 2 Origins

The roots of dimorphic computing reach deep into the past. Its 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). Dimorphic computing continues this evolution by seamlessly and transparently using local and remote machines in flexible roles. It is driven by the recent convergence of three forces, which we describe in the following sections:

- Growth of relevant applications.
- Availability of graphics-accelerated thick clients and renewed interest in thin clients.
- Coming of age of virtual machine technology.

## 2.1 Relevant Applications

There are a growing number of applications with both crunch and cognitive phases in the domains of graphic arts, science, and engineering. For example, the creator of a high-definition animated movie will typically alternate between a crunch phase that generates an animation segment from key frames, and a cognitive phase for artistic review and refinement of that segment. As another example, in a molecular modeling application, the output of the crunch phase is an optimized 3-D model whose geometric properties are interactively examined by a scientist. Based on insights from this cognitive phase, the scientist may modify the modeling parameters and iterate.

Depending on the application, the crunch phase may be CPU-intensive, memory-intensive, disk-intensive or some combination of all three. This often leads to the use of a remote machine cluster or a supercomputer, possibly located far away. Sometimes the crunch phase may use datasets that are too large to mirror or cache locally. In domains such as health care, regulatory or organizational policies may forbid copying of the data and thus prevent mirroring or caching. The only option in that case is to execute the application at the site where the data is located. On the other hand, the crisp interactive performance demanded by

| End Points | RTTs (ms) | | | |
|---|---|---|---|---|
| | Min | Mean | Max | *c* |
| Berkeley – Canberra | 189.0 | 189.1 | 199.0 | 79.9 |
| Berkeley – New York | 85.0 | 85.0 | 85.0 | 27.4 |
| Berkeley – Trondheim | 190.0 | 190.0 | 193.0 | 55.6 |
| New York – Atlanta | 28.0 | 28.0 | 29.0 | 8.0 |
| New York – Zurich | 108.0 | 108.0 | 109.0 | 42.2 |
| New York – Taipei | 245.0 | 245.7 | 248.0 | 83.7 |
| Pittsburgh – Seattle | 83.0 | 83.8 | 84.0 | 22.9 |

These RTT measurements were obtained from NLANR [35] on April 14th, 2006. 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 1: Observed Round Trip Times**

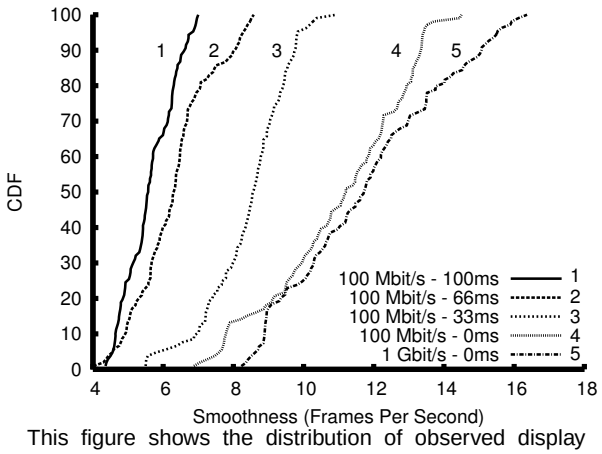
the cognitive phase is only possible with low network latency and jitter. This requires execution to occur very close to the user.

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.

## 2.2 Thick and Thin Clients

The term "thick client" is industry jargon for a full-fledged personal computer that can operate stand-alone, in isolation from network resources. Thick clients provide an ideal platform for the cognitive phase of applications such as scientific visualization and digital animation. The absence of display latency or jitter, combined the widespread availability of 3D graphics acceleration hardware, results in crisp user interaction during this phase. On the other hand, a typical thick client such as a desktop may not be able to meet the CPU or I/O performance demands of the crunch phases of these applications.

A thin client [4, 30, 41] is the modern-day realization of a "dumb terminal" from the timesharing era. It can be a physical piece of hardware, or it can be software such as VNC [42] that runs on a thick client and emulates a thin client. Thin clients provide an elegant way to harness remote computational resources, and are thus an ideal platform for the crunch phase of demanding applications. Unfortunately, even trivial user-machine interactions on a thin client incur network queuing delay. This queuing delay is acutely sensitive to the vagaries of the external computing environment. Whether a thin client can offer a satisfactory user experience during the cognitive phase of an application depends on both the application and on network quality. If near-ideal network conditions (low latency and high bandwidth) can be guaranteed, thin clients offer a good user experience; such conditions cannot be guaranteed on the Internet, especially for distant sites.




This figure shows the distribution of observed display frame rates for a highly interactive segment of roughly one minute, when executed on a thin client at various bandwidths and latencies.

**Figure 1: Network Latency Impact on Smooth Visualization**

It is latency, not bandwidth, that is the greater challenge. Table 1 shows measured round-trip time (RTT) values for a representative sample of Internet2 sites. In every case, the minimum observed RTT value far exceeds the lower bound established by the speed of light. Technologies such as firewalls and overlay networks further exacerbate the problem. Although bandwidths will continue to improve over time, latency is unlikely to improve dramatically.

As latency and jitter increase, the interactive response of thin clients suffers. Tolia et al. [52] show that tightly coupled tasks such as graphics editing suffer more than loosely coupled tasks such as web browsing. Even simple actions, such as pressing a mouse button to pop up a menu or tracing an arc in a figure, appear jerky and sluggish. This distracts the user and reduces the depth of cognitive engagement with the application. Although users can adapt to higher response times, they experience frustration with the system and a drop in productivity.

Figure 1 illustrates the impact of latency on smoothness of visualization of a highly interactive application on a thin client. At 1 Gbit/s with zero latency, observed frame rates span 8–16 FPS. At 100 Mbit/s with zero latency, they drop slightly to 7–15 FPS. But the impact of increasing latency is much greater: observed frame rates at 100 Mbit/s span 6–10 FPS with 33 ms latency, 4–8 FPS with 66 ms latency, and 4–6 FPS with 100 ms latency. The sustainable frame rate is limited by the end-to-end delay of the system, and our experience confirms that this metric correlates well with subjective user experience. At the lower frame rates in Figure 1, the system's interactive response is sluggish and annoying.

## 2.3 Virtual Machines

VM technology is a critical building block for implementing dimorphic computing. The use of VM migration makes it possible to implement transitions between thick and thin

client execution in a way that is transparent to an application and its entire runtime infrastructure, including the operating system. Although the VM concept dates back to the late 1960s [16], its realization on modern hardware is of more recent origin. Today, the availability of VMWare [19], Xen [5], and VT support for hardware virtualization [54] collectively represent a maturing of VM technology into a robust, field-deployable building block.

*Process migration* is an alternative technology on which to base dimorphic computing. This operating system capability allows a running process to be paused, relocated to another machine, and continued there. It has been implemented in many experimental operating systems such as Demos [39], V [51], Mach [56], Sprite [11], Charlotte [3], and Condor [55]. Yet, no operating system in widespread use today (proprietary or open source) supports it as a standard facility because it is excruciatingly difficult to get the details right. A typical implementation involves so many external interfaces that it is easily rendered incompatible by a modest external change. Process migration is thus a brittle abstraction, and a much less attractive building block for dimorphic computing.

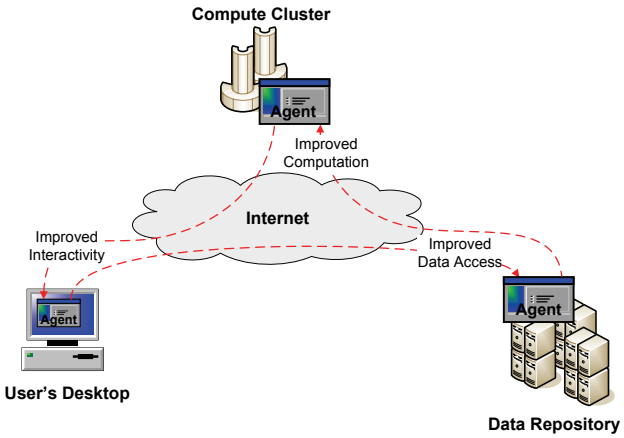
# 3 AgentISR

To explore the applicability and value of dimorphic computing, we have built a prototype called *AgentISR*. Our prototype leverages current VM technology, but extends it with several new mechanisms specifically motivated by dimorphic computing.

AgentISR has four key attributes, the first two of which arise directly from the use of VMs, while the other two result from our research efforts. First, applications do not have to be modified, recompiled, or relinked to use AgentISR. A corollary is that the source code for the application does not have to be available. This greatly simplifies real-world deployments where use of proprietary rather than open-source applications may be unavoidable. Second, the application does not have to be written in a specific language, nor does it need to be built using specific libraries. By requiring almost nothing of applications except the existence of distinct crunch and cognitive phases, AgentISR invites the broadest possible usage. Third, 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. Finally, migration is transparent and seamless to the user, beyond the unavoidably noticeable (and desirable) effects of improved interactive or computational performance.

## 3.1 Agent Abstraction

AgentISR implements the notion of an *agent*, a migratable embodiment of an application that transparently and seamlessly relocates itself to achieve optimal performance.




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

**Figure 2: Dimorphic Computing Example**

We use this term for the entity that a user interacts with in dimorphic computing because it resembles an *autonomous agent*, as that term is used in artificial intelligence (AI) [25]. Of course, our work differs in fundamental ways from the huge body of research on agent technology in AI. Most obviously, we take a low-level systems approach and strive for compatibility with unmodified legacy applications. This contrasts with the high-level programming environments typical of AI agents. We also pay considerable attention to interactive performance, which is typically not of interest in AI.

Figure 2 shows the example of an agent that starts at the user's desktop, where it executes in thick client mode to favor interactive execution. It then migrates to several remote sites where it executes in thin client mode to favor CPU performance or I/O performance, and then returns to the desktop for the next cognitive phase.

An agent may execute on any AgentISR-enabled host for which its owner has SSH access privileges; these SSH credentials are used to encrypt all communications. However, 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.

Table 2 shows the command line interface for AgentISR. 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 3.5. However, they are available for explicit user control, if desired.

Migratable agents are implemented in AgentISR through the combination of four components, which we describe in the rest of this section: a Virtual Machine Monitor (VMM); WANDisk, a peer-to-peer system for transporting an agent's persistent state; an agent interface capable

| 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 2: AgentISR Commands**

of providing hardware accelerated graphics; and a migration manager that automatically triggers modality changes from thick to thin and vice versa. While the first component leverages existing work, the other three result from our research.

## 3.2 Use of Xen

AgentISR uses a 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 AgentISR is based on the Xen 3.0.1 VMM. We chose Xen because its open-source nature makes it attractive for experimentation. Our design is sufficiently modular that using a different VMM such as VMWare will only require modest changes.

AgentISR uses VM migration [29, 45] 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 transfered to the target, where VM execution is resumed. AgentISR uses *live-migration* [9] 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 critical mass, a brief pause is sufficient to transfer control.

## 3.3 The WANDisk Storage System

VMM 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). Each VM disk operation after migration therefore involves network access to the source host. While this may be tolerable on a LAN, it is unacceptable for the high-latency WAN environments in which we envision AgentISR 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 [43], AFS [22], Coda [47], and Lustre [7]. Storage Area Networks (SANs) are also in extensive use today. Unfortunately, most of these systems are designed for a LAN environment and perform poorly in WANs. Although AFS and Coda do perform acceptably on WANs, they embody a client-server architecture in which server creation is a heavyweight action. This is a misfit for the dynamic usage model envisioned for AgentISR.

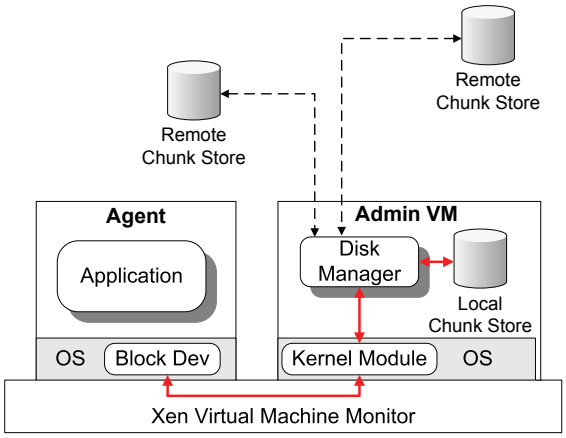



**Figure 3: The WANDisk Storage System Architecture**

We have therefore implemented a distributed storage system called WANDisk that provides efficient WAN access to an agent's virtual disk. Its design reflects the unique set of challenges and opportunities that the AgentISR's deployment model presents. To provide flexibility in the choice of migration site, WANDisk follows a peer-to-peer (P2P) approach where any Internet host can maintain a persistent replica of the agent's state. Replicas are created on demand as new migration sites are identified. WANDisk relies on the persistence of the replicas to reduce data transfers. Its replica control mechanism uses many techniques for optimizing the efficiency of agent migration: partial replication to reduce the startup cost of initializing a new migration site; differential transfers between replicas to reduce synchronization overhead; and lazy synchronization to avoid unnecessary data transfers to inactive migration sites or for unused parts of a virtual disk.

Figure 3 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 fake block device that is mapped to an agent's virtual block device. All agent-originated block requests are handled by the fake 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; our implementation uses a chunk size of 128 KB, 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 freshest copy of each chunk. The chunk table thus becomes a critical extension of a Xen VM migratable state. To account for this, we have modified live migration in Xen to include the chunk table. We use the rsync algorithm [53] to perform efficient chunk transfer.

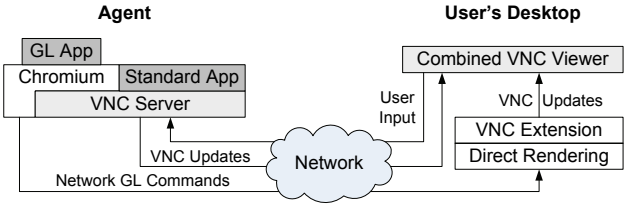
The heavyweight `sync` command shown in Table 2 is available for bringing any replica up to date under explicit user control. This command may be used for performance or reliability reasons. The command blocks until the replica at the specified migration site is both complete and up to date. After that, agent execution can continue at that site even if it is disconnected from other replicas.

## 3.4 Unified Graphical User 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 dimorphic computing, e.g., scientific visualization or digital animation, require the use of 3D graphics acceleration hardware, a feature absent from most virtualized execution environments.

To meet these requirements AgentISR provides an enhanced thin client interface based on VNC. 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 IP address to the new physical host. Relocations across subnets are supported with VNETs [48], a Layer-2 proxy.

As efficient virtualization of graphics hardware requires manufacturer support, we instead virtualize the OpenGL API. We leverage Chromium [23], a framework that was originally designed for distributed rendering on clusters. Chromium uses 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 rendering module running on the user's desktop, where they are rendered directly by the locally available 3D graphics acceleration hardware. This solution differs from other systems that also virtualize the OpenGL API [20] as it does not require application modification and can support 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: Thin client extensions for 3D support**

The architecture of our integrated rendering system is shown in Figure 4. GL-primitives from applications launched with Chromium bypass the thin client server, and are rendered using GPU hardware on the client-side. Standard applications using non-3D APIs (e.g. Xlib) are rendered by the thin client server on its virtual framebuffer and shipped to the viewer. A modified thin client viewer composes both streams and offers a combined image to the user. Input events are handled entirely by the thin client protocol.

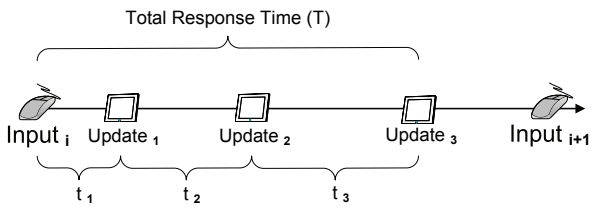
## 3.5 Sensor-Driven Migration Manager

System-controlled agent relocation is a key requirement of dimorphic computing. In other words, the decision to migrate, the choice of migration site, and the collection of information upon which to base these decisions should all happen under the covers in a manner that is transparent to the user and to the agent. We implement this requirement in AgentISR through a *migration manager* module. This module bases its migration decisions on *performance sensors* that extract relevant data from the VMM and the AgentISR user interface. *Migration profiles* express the rules for migration as transitions of a finite state machine triggered by sensor readings. AgentISR's clean separation between policy and mechanism welcomes the use of different profiles, different sensors, or even migration managers based on completely different principles.

### 3.5.1 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 our enhanced thin client viewer. As shown in Figure 5, it collects a stream of time-stamped events corresponding to keyboard/mouse inputs and screen updates. The intensity of the user's inter-




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

**Figure 5: Measuring Interaction Intensity and Smoothness**

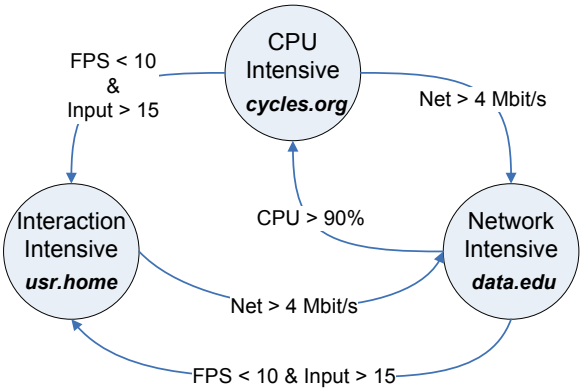
active demand and the smoothness of the agent's response can both be inferred from this stream.

Our measure of *interaction intensity* is the number of input events per unit 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 oldest outstanding 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. Because most thin-client algorithms are non-work-conserving, they skip frames under adverse network conditions to "catch up" with the input. 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 thin-to-thick transition.

#### 3.5.2 Migration Profiles

A migration profile defines a finite state machine that is used to model the agent's behavior. As shown in Figure 6, 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.

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 agent-specific profile, the migration manager uses a default profile that identifies typical crunch and cognitive phases. In the future, we plan to study the use of machine learning techniques to automate the generation of migration profiles. While we did not experience any "state-thrashing" phenomena (an agent transitioning back and forth between two




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

**Figure 6: Finite State Machine of Agent States**

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

**Table 3: Application characteristics**

states), traditional hysteresis mechanisms [39] can shield the migration manager from adopting this erratic behavior.

## 4 Experimental Methodology

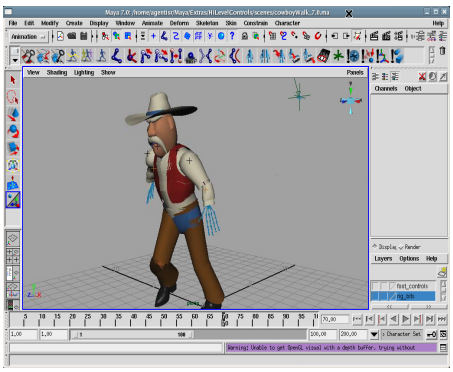
In this section, we present the methodology for the evaluation of AgentISR's performance. To demonstrate the applicability of dimorphic computing to a broad set of application domains, we carried out experiments with four applications, summarized in Table 3. The applications include both open source as well as commercial closed source products, and are representative of the domains of professional 3D animation, amateur video production, and scientific computing. For each application, we designed a representative benchmark that includes both a crunch and a cognitive phase.

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

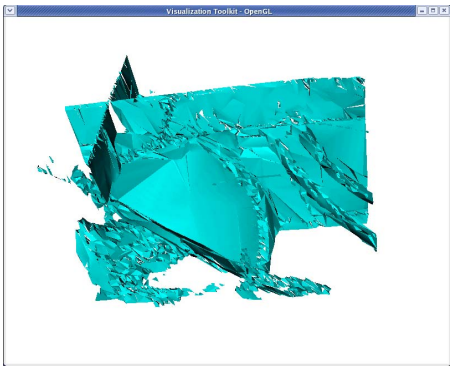
### 4.1 Application Benchmarks

#### 4.1.1 Maya: Digital Animation

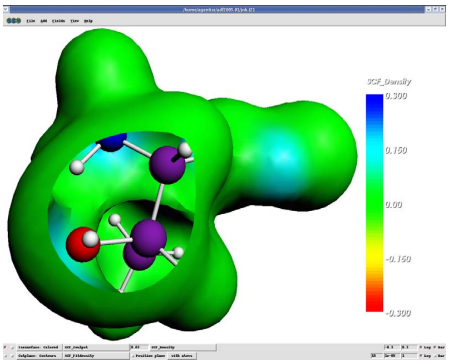
Maya [34] is a commercial close source high-end 3D graphics animation package used for character modeling, animation, digital effects, and production-quality rendering. It is an industry standard and has been employed in several major motion pictures, such as "*Lord of the Rings*," "*War of the Worlds*," and "*The Chronicles of Narnia.*"



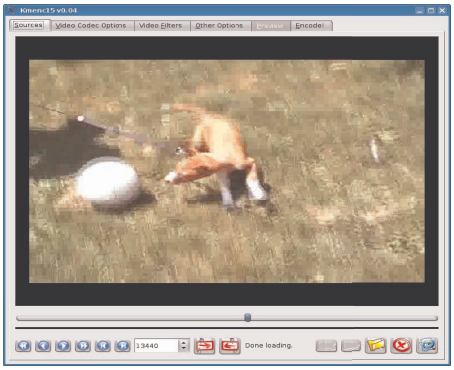
(a) Maya: Character modeling



(b) QuakeViz: Ground motion isosurface



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



(c) Kmenc15: Video Editing

**Figure 7: Application Screenshots**

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 7(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.

When the animation design is complete, the user initiates a production-quality rendering, i.e., the crunch phase. This is a CPU intensive task as each photo-realistic frame is rendered with a number of lighting effects. 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. 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 [2]. During a computationally intense 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 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. QuakeViz is the only application we use that accesses a remote dataset.

In the ensuing cognitive phase, the scene is synthesized and the meshes are rendered on the screen (see Figure 7(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 last for approximately 23 minutes, and involves exploration of the seismic reaction isosurfaces at 30 different time-steps. The results shown in Figure 1 were obtained from a segment of QuakeViz benchmark's cognitive phase.

#### 4.1.3 ADF: Quantum Chemistry

Amsterdam Density Functional (ADF) [49] 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 a set of results that are visualized in a subsequent cognitive phase: isosurfaces for the Coulomb potential, occupied electron orbitals, and cut-planes of kinetic energy density and other properties (see Figure 7(c)). Analysis of these properties through rotation, zooming, or panning, are examples of the actions performed during the 26 minutes-long cognitive phase.

#### 4.1.4 Kmenc15: Video Editing

Kmenc15 (KDE Media Encoder) [28] 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.

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 7(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 simultaneously, significant gains can be obtained from executing at a multiprocessor.

### 4.2 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. While VNC, like most other thin client protocols, is non work-conserving and can skip intermediate frame updates on slow connections, the client always reaches a stable and similar (albeit not always identical) state for a given input. This enables screen synchronization to succeed and ultimately, given an identical initial application state, allows for reliable replay of the recorded interactive session.

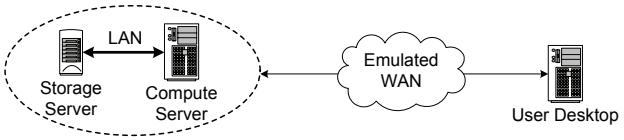



**Figure 8: Experimental Testbed**

Unfortunately, the simple screen synchronization algorithms used by other replay tools [44, 57] do not work well in high-latency environments. These tools 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 single pixel. In turn, this offset causes the algorithm to detect a large number of mismatches.

To address this problem, we use 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 threshold, it is classified as a mismatch. If the number of mismatches are greater than a second threshold, the screenshots being compared are declared to be different. This improved matching algorithm proved to work well in practice over high latency networks.

### 4.3 Experimental Testbed

Figure 8 shows our experimental testbed, which consists of a user desktop, a compute server, and a storage server. The user desktop was a commodity 3.6 GHz Intel Pentium IV equipped with an ATI Radeon X600 Graphics Processor Unit (GPU). The compute server was a four-way SMP (two dual-threaded cores) 3.6 GHz Intel Xeon. The storage server was a commodity PC that serves QuakeViz's dataset through a NFS share[1]. The storage and compute servers were connected via a gigabit LAN.

We used a paravirtualized 2.6.12 Linux kernel for the AgentISR experiments and Fedora's 2.6.12 Linux kernel for the non-AgentISR experiments. Both kernels were configured with 512 MB of RAM. AgentISR uses HPN-SSH [40], a WAN-optimized transport, for all of its data transfers.

The user desktop communicated with the storage and compute servers through a WAN link emulated using NetEm [21]. Based on recent Internet bandwidth measurements on Planetlab [31] and Abilene [24], and representative latencies from NLANR [35], we configured the WAN link with a bandwidth of 100 Mbit/s and chose network round-trip times of 33, 66, and 100 ms.

[1]Note that QuakeViz uses NFS only for the dataset. The virtual disk is still supported by WANDisk.

### 4.4 Benchmark Methodology

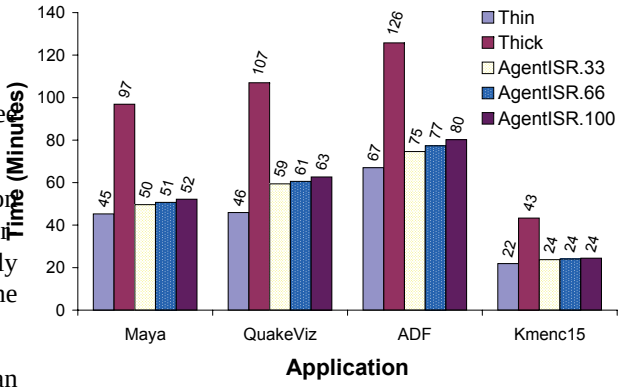
We replayed our benchmarks using the following three configurations:

**Thick Client:** The application executes exclusively on the user's desktop without virtualization. During interactive phases, 3D graphics are rendered using the locally available GPU. This represents the best scenario for the cognitive phase, but the worst case for the crunch phase.

**Thin Client:** The applications 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.

**Dimorphic Client:** AgentISR is employed to dynamically switch client thickness as appropriate. Both the user's desktop and remote compute server run the AgentISR infrastructure: Xen VMM, WANDisk, the enhanced agent GUI, and the migration manager. All benchmarks are initiated in an agent running at the user's desktop. We were able to use a single generic application profile for all of our experiments.

The objectives of the thin and thick client experiments were to provide bounds for the crunch and interactive performance of our target applications on this testbed. We quantify the *performance* and *overhead* of AgentISR against these bounds. AgentISR's ability to provide a *transparent* and *seamless* user experience is measured by the agility of its automated migration mechanisms.




This figure shows the crunch completion time for Thin, Thick, and AgentISR clients. AgentISR 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 9: Total Completion Time - Crunch Phase**

## 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.

### 5.1 Crunch Phase

Figure 9 shows the total completion time of the crunch phase for the four benchmarks under three configurations: thin, thick and AgentISR. We only show results for different network round trip latencies for AgentISR, as the performance of the crunch phase for the thin and thick client configurations was not affected by differences in the network round trip times. This was expected for Maya, ADF, and Kmenc15, which do not access remote data in thick client configuration. However, we expected that differences in network latency would affect the performance of QuakeViz, which reads a single 1.9 GB file over NFS. It appears that the NFS client's readahead functionality effectively masked the impact from increased latency.

By migrating to the remote compute server, AgentISR is able to significantly outperform the thick client configuration. Specifically, at 33 ms, AgentISR reduced the length of the crunch phase for Maya, QuakeViz, ADF, and Kmenc15, by 49%, 44%, 41%, and 45%, respectively. Moreover, it came within 29% of the performance of the thin client configurations for all benchmarks. 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 connection to the data server.

Table 4 show 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 2 minutes on the user's desktop after it entered a crunch phase. The Table also shows that, for agent migration, the maximum time for which an agent would appear to be unresponsive to user input was very small and always less than 7 seconds.

### 5.2 Cognitive Phase

Figure 10 shows the Cumulative Distribution Functions (CDFs) of the number of frames per second (FPS) for each of our four benchmarks under three configurations: thin, thick, and AgentISR. We show results for different network round trip latencies for the thin client and AgentISR configurations. The cognitive phases for QuakeViz and

| | Time (seconds) | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | Best | Latency = 33 ms | | | Latency = 66 ms | | | Latency = 100 ms | | |
| Application | AgentISR | 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 *AgentISR* column measures the best observed crunch time for AgentISR.. The *Detect*, *Migrate*, and *Suspend* columns measures 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.

**Table 4: Crunch Phase - Agent Migration Times**

| | 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 measures 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. Maya and Kmenc15 results are not relevant as they begin interaction in thick client mode and do not need to migrate. 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 - Agent Migration Times**

ADF start on the remote compute server soon after the crunch phase terminates. The migration manager detects this transition and migrates promptly 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 AgentISR performs much better than thin clients for both the median ($50^{th}$ percentile) and the $95^{th}$ percentile cases. For Maya, Figure 10 (a) shows that, in the median case, AgentISR delivers 3.9 times more FPS than a thin client at 33 ms latency and 4.7 times more FPS at 100 ms latency. The results are similar for the $95^{th}$ percentile where AgentISR delivers between 2.8 and 3.8 times more FPS. The results for Kmenc15 are similar.

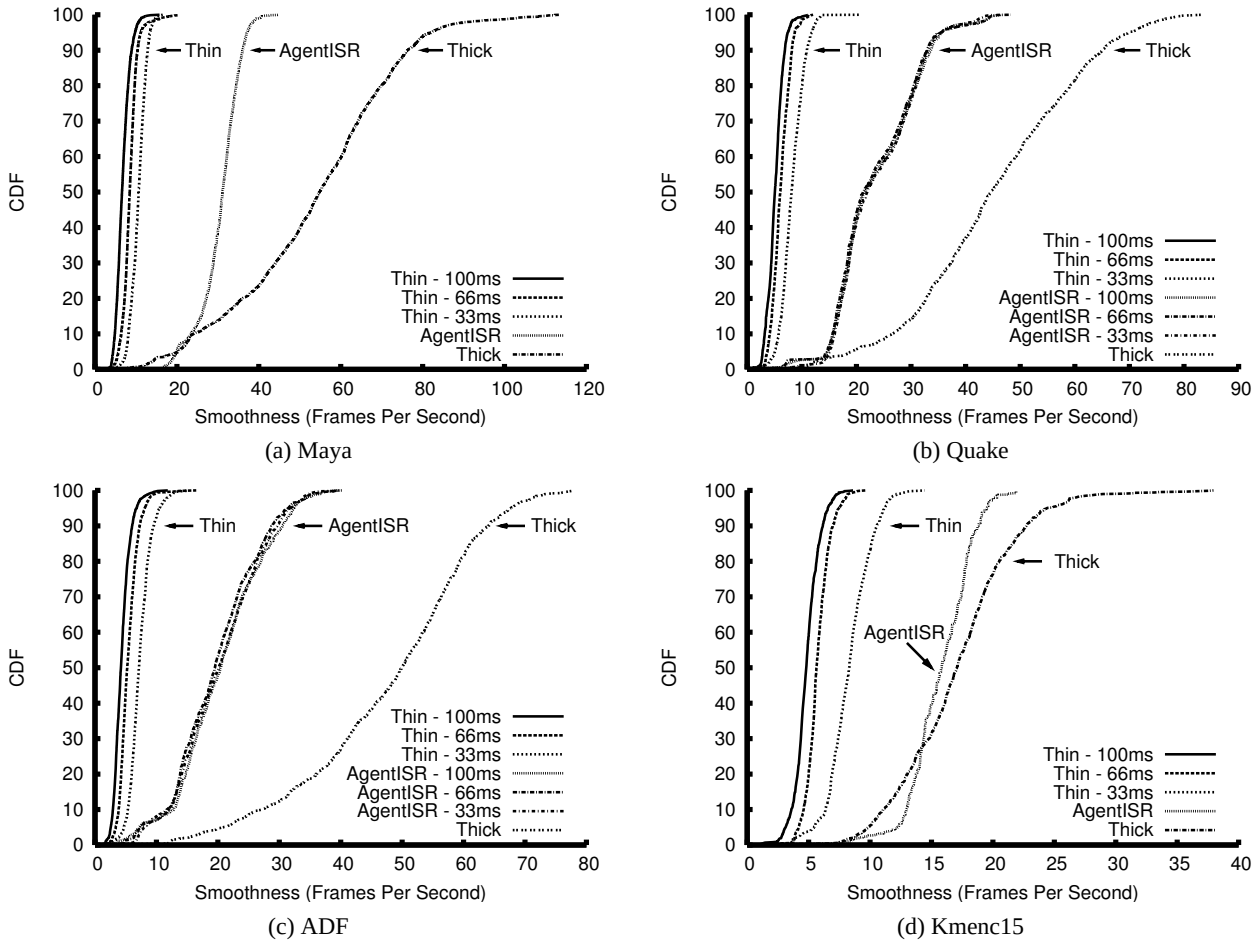
Results from the QuakeViz benchmark, seen in Figure 10 (b), show that even though the agent has to migrate from the compute server to the user's desktop, AgentISR's cognitive performance tends to be independent of the WAN latency. As Table 5 shows, this occurs because both the time taken before the decision to migrate is made and the time required to migrate the agent are independent of the network latency. In the median case, AgentISR 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.

The results from the ADF benchmark are almost identical to the QuakeViz results. However, it is interesting to note that it takes the migration manager slightly longer to invoke migration for ADF in the 33 ms case. This is because the thin client, at 33 ms, delivers adequate performance during the initial ramp-up part of the cognitive phase.

While AgentISR delivers a much better user experience that thin clients, the results show the FPS delivered are not as high as those of thick clients. Thick clients 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 is not a fundamental limitation of dimorphic computing but instead an artifact of the OpenGL virtualization described in Section 3.4. 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 AgentISR's performance much closer to that of an unvirtualized thick client. Further, unlike thin clients, the median number of FPS delivered by AgentISR is above the long established 20 FPS threshold needed for crisp interactivity [1].

# 6 Discussion and Future Work

The results presented in Section 5 confirm the feasibility and value of dimorphic computing for some real-world applications. It is important to note that none of these applications were written by us, or modified for use with AgentISR. Two of the applications (Maya and ADF) are




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

**Figure 10: Interactive Response**

commercial products whose viability in the marketplace confirms their importance. The other two applications (Quake 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. An important aspect of our future work will be the identification of other application domains that can benefit from dimorphic computing.

The key figure of merit for an implementation of dimorphic computing is *agility.* This includes both the swiftness with which migration can be triggered, and the efficiency with which it can be completed. A complementary attribute is *stability,* which characterizes the ability of the implementation to resist 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. Detection and change of modality occurs in roughly 10 seconds, while the migration that follows typically takes about 60 seconds. Since we have not observed any instances of thrashing, we infer that the stability of our prototype is reasonable for the tested applications.

Improving agility would broaden the class of applications for which AgentISR is attractive. A highly agile implementation would enable applications with short crunch times to benefit from AgentISR. It would also allow them to take advantage of remote resources that only offer modest crunch speedup. The combination of short crunch time and modest crunch speedup defines a particularly challenging workload for AgentISR. To improve its ability to handle such workloads we are exploring the use of prefetching WANDisk chunks as well as improvements to the CPU, network, and interaction sensors.

AgentISR's agility in cognitive phases is already acceptable for many applications. Even under extreme networking conditions in which thin client execution is frustrating, a user can be confident that the client will morph into a

thick client within roughly a minute. However, interactive performance under these conditions is noticeably poorer than that of a thick client even though it is much better than that of a thin client. We plan to address this by significantly streamlining the combined Chromium-VNC stack, and thus removing several sources of processing overhead and latency.

In the recent past, PlanetLab [37] has emerged as a widely-used computing infrastructure that offers an application multiple vantage points on the Internet. It is thus ready to catalyze the growth of dimorphic computing. By migrating to a PlanetLab node that is close to a large dataset on the Internet, AgentISR can potentially improve crunch phase performance. We plan to explore this symbiotic relationship between dimorphic computing and PlanetLab in our future work.

# 7 Related Work

Closest in spirit to AgentISR is the large body of research on process migration. However, as described in Section 2.3, process migration is a brittle abstraction: it breaks easily, and long-term maintenance of machines with support for process migration requires excessive effort. In contrast, AgentISR requires no host operating system changes, and builds on the very stable and rarely-changing interface to hardware provided by its virtual machine monitor (VMM). As long as the VMM can be easily deployed, so too can AgentISR. The myriad interfaces that must be preserved during migration are part of the guest OS, and so the code implementing these interfaces is transported with the application. This stability of interfaces comes at a price. The VM approach of AgentISR involves a much larger amount of state than process migration. Fortunately, much of this state rarely changes and can therefore be persistently cached at potential destinations.

*Language-based code mobility* is another well-explored approach to moving computation. The best early example of work in this genre is Emerald [26]. A more recent example is *one.world* [18]. The growth in popularity of Java and its support for *remote method invocation* [38] has made this approach feasible and relevant to a wide range of computing environments. Unfortunately, this approach does not work for legacy applications that were not written in the specified language. In contrast, AgentISR 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 further fine structure, such as the use of multiple large datasets each of which is located at a different Internet site.

*Grid computing toolkits* such as Globus [14], Condor [50], and OGSA [15] 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 and logging in 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. AgentISR complements the functionality provided by these toolkits. It transforms 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 [13, 27, 48]. The closest work in this area, from the Virtuoso project [32], focuses on resource scheduling for interactive VMs running on a single host.

Researchers have developed a number of systems that support distributed visualization of large remote datasets. Examples include Dv [33], GVU [10], Visapult [6], SciRun [36], and Cactus [17]. Unlike AgentISR, 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, AgentISR was inspired by the substantial body of recent work on applying VM technology to a wide range of systems problems [8, 12, 45]. The Internet Suspend/Resume (ISR) project [29, 46] has been the most direct influence, and this ancestry is reflected in the similarity of names.

# 8 Conclusion

We showed that thin and thick clients fail on their own to optimize the performance of a growing class of applications that alternate between a resource-intensive *crunch* phase that involves significant processing, and a *cognitive* phase that is intensely interactive. Examples of these applications include computer animation and video editing, scientific visualization of complex physical phenomena, and computer-aided drug design.

We introduced *dimorphic computing* a new computational model that combines the strengths of thin and thick clients, without having their limitations. During the application's crunch phase, the dimorphic client behaves like a thin client, taking advantage of remote resources such as compute servers and large datasets. During the cognitive phase, the dimorphic client behaves like a thick client and takes advantage of local graphical hardware acceleration to provide crisp interactive performance. Transitions are transparent and seamless to the user, who experiences good performance at all times.

We described the implementation of AgentISR, a dimorphic computing prototype that uses virtual machine migra-

tion to move application execution state between hosts. In experiments with open as well as close-source commercial applications, AgentISR achieves interactive performance that far exceeds that achievable by a thin client, while its crunch phase performance is significantly better than that obtained through thick client execution. AgentISR successfully detected application transitions between crunch and cognitive phases, and automatically morphed to the most appropriate thin or thick client mode.

## Acknowledgments

We would 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] J. M. Airey, J. H. Rohlf, and J. 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.

[2] V. Akcelik, J. Bielak, G. Biros, I. Epanomeritakis, A. Fernandez, O. Ghattas, E. J. Kim, J. Lopez, D. O'Hallaron, T. Tu, and J. Urbanic. High resolution forward and inverse earthquake modeling on terasacale computers. In *Proc. ACM/IEEE conference on Supercomputing*, Phoenix, AZ, Nov. 2003.

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

[4] Baratto, R., Potter, S., Su, G., Nieh, J. MobiDesk:Virtual Desktop Computing. In *Proc. 10th Annual ACM International Conference on Mobile Computing and Networking*, Philadelphia, PA, Sept. 2004.

[5] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield. Xen and the art of virtualization. In *Proc. 19th ACM Symposium on Operating Systems Principles (SOSP)*, pages 164–177, Bolton Landing, NY, Oct. 2003.

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

[7] P. J. Braam. The lustre storage architecture, Nov. 2002. `http://www.lustre.org/docs/lustre.pdf`.

[8] R. Chandra, N. Zeldovich, C. Sapuntzakis, and M. S. Lam. The Collective: A Cache-Based System Management Architecture. In *Proc. 2nd Symposium on Networked Systems Design & Implementation (NSDI)*, Boston, MA, 2005.

[9] C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield. Live migration of virtual machines. In *Proc. 2nd USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, Boston, MA, May 2005.

[10] 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, Aug. 2001.

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

[12] G. W. Dunlap, S. T. King, S. Cinar, M. A. Basrai, and P. 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, Dec. 2002.

[13] R. J. Figueiredo, P. A. Dinda, and J. 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.

[14] 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.

[15] I. Foster, C. Kesselman, J. M. Nick, and S. Tuecke. The physiology of the grid: An open grid services architecture for distributed systems integration, June 2002. `http://www.globus.org/alliance/publications/papers/ogsa.pdf`.

[16] Goldberg, R.P. Survey of Virtual Machine Research. *IEEE Computer*, 7(6), June 1974.

[17] 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.

[18] 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.

[19] L. Grinzo. Getting Virtual with VMware 2.0. *Linux Magazine*, June 2000.

[20] J. G. Hansen. Blink: 3d multiplexing for virtualized applications. Technical Report 06-06, Dept. of Computer Science, University of Copenhagen, Apr. 2006.

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

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

[23] G. Humphreys, M. Houston, R. Ng, R. Frank, S. Ahern, P. D. Kirchner, and J. 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, USA, 2002.

[24] *Abilene Weathermap*. Indiana University Global Network Operations Center, Apr. 2006. `http://weathermap.grnoc.iu.edu/abilene_jpg.html`, Correct as of 7th April 2006.

[25] N. R. Jennings and M. J. W. (Editors). *Agent Technology: Foundations, Applications and Markets*. Springer-Verlag, 2002.

[26] E. Jul, H. Levy, N. Hutchinson, and A. Black. Fine-grained mobility in the emerald system. *ACM Transactions on Computer Systems*, 6 (1):109–133, 1988.

[27] K. Keahey, I. Foster, T. Freeman, X. Zhang, and D. Galron. Virtual workspaces in the grid. *Lecture Notes in Computer Science*, 3648: 421–431, Aug. 2005.

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

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

[30] Lai, A., Nieh, J. Limits of Wide-Area Thin-Client Computing. In *Proc. ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, Marina Del Rey, CA, June 2002.

[31] S.-J. Lee, P. Sharma, S. Banerjee, S. Basu, and R. Fonseca. Measuring bandwidth between planetlab host. In *Proc. 6th Passive and Active Measurement Workshop (PAM)*, Boston, MA, Mar. 2005.

[32] B. Lin and P. 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, Nov. 2005.

[33] J. López and D. 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, Aug. 2001.

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

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

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

[37] L. Peterson, T. Anderson, D. Culler, and T. Roscoe. A blueprint for introducing disruptive technology into the internet. *SIGCOMM Computing and Communication Review*, 33(1), 2003.

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

[39] M. L. Powell and B. P. Miller. Process migration in demos/mp. In *Proc. 9th ACM Symposium on Operating Systems Principles (SOSP)*, Oct. 1983.

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

[41] Ricardo A. Baratto and Jason Nieh and Leo Kim. THINC: A Remote Display Architecture for Thin-Client Computing. In *Proc. 20th ACM Symposium on Operating Systems Principles (SOSP)*, Brighton, UK, Oct. 2005.

[42] Richardson, T., Stafford-Fraser, Q., Wood, K. R., and Hopper, A. Virtual Network Computing. *IEEE Internet Computing*, 2(1):33–38, Jan/Feb 1998.

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

[44] H. Sandklef. Testing Applications with Xnee. *Linux Journal*, 2004 (117):5, 2004. ISSN 1075-3583.

[45] C. P. Sapuntzakis, R. Chandra, B. Pfaff, J. Chow, M. S. Lam, and M. Rosenblum. Optimizing the migration of virtual computers. In *Proc. 5th Symposium on Operating Systems Design and Implementation (OSDI)*, Dec. 2002.

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

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

[48] A. I. Sundararaj and P. 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.

[49] G. te Velde, F. M. Bickelhaupt, E. J. Baerends, C. F. Guerra, S. J. A. van Gisbergen, J. G. Snijders, and T. Ziegler. Chemistry with ADF. *Journal of Computational Chemistry*, 22(9):931–967, 2001.

[50] D. Thain, T. Tannenbaum, and M. Livny. Distributed computing in practice: The condor experience. *Concurrency and Computation: Practice and Experience*, 17:323–356, February-April 2005.

[51] 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, Dec. 1985.

[52] Tolia, N., Andersen, D., Satyanarayanan, M. Quantifying Interactive Experience on Thin Clients. *IEEE Computer*, 39(3), Mar. 2006.

[53] 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.

[54] R. Uhlig, G. Neiger, D. Rodgers, A. L. Santoni, F. C. M. Martins, A. V. Anderson, S. M. Bennett, A. Kägi, F. H. Leung, and L. Smith. Intel virtualization technology. *IEEE Computer*, 38(5):48–56, 2005.

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

[56] E. Zayas. Attacking the Process Migration Bottleneck. In *Proc. 11th ACM Symposium on Operating System Principles (SOSP)*, Austin, TX, Nov. 1987.

[57] N. Zeldovich and R. Chandra. Interactive performance measurement with vncplay. In *Proc. USENIX Annual Technical Conference, FREENIX Track*, Anaheim, CA, Apr. 2005.