DataStations: Ubiquitous Transient Storage for Mobile Users

Sai Susarla and John Carter
{*sai, retrac*}*@cs.utah.edu*

UUCS-03-024

School of Computing
University of Utah
Salt Lake City, UT 84112 USA

November 14, 2003

Abstract

In this paper, we describe *DataStations*, an architecture that provides ubiquitous transient storage to arbitrary mobile applications. Mobile users can utilize a nearby DataStation as a proxy cache for their remote home file servers, as a file server to meet transient storage needs, and as a platform to share data and collaborate with other users over the wide area. A user can roam among DataStations, creating, updating and sharing files via a native file interface using a uniform file name space throughout. Our architecture provides transparent migration of file ownership and responsibility among DataStations and a user's home file server. This design not only ensures file permanence, but also allows DataStations to reclaim their resources autonomously, allowing the system to incrementally scale to a large number of DataStations and users.

The unique aspects of our DataStation design are its decentralized but uniform name space, its locality-aware peer replication mechanism, and its highly flexible consistency framework that lets users select the appropriate consistency mechanism on a per-file replica basis. Our evaluation demonstrates that DataStations can support low-latency access to remote files as well as ad-hoc data sharing and collaboration by mobile users, without compromising consistency or data safety.

1 Introduction

The amount of data handled by mobile users is steadily increasing as they use increasingly complex applications and the cost of storage decreases. Our research focuses on providing efficient globally-accessible file storage for mobile users. Many systems exist that provide wireless clients access to "local" file servers when they are roaming within their personal work space. However, when mobile users roam far from their normal work environment, they are typically forced to either store all the files they might want to access on their mobile client or access the files via a WAN connection (if available). Similarly, when such mobile users wish to create a new file, they are typically forced to create it either on their mobile device or via a WAN connection. Neither of these solutions are entirely satisfying. Providing large storage devices on the mobile client greatly increase its weight and power consumption, and files stored on the client are not kept consistent with (or made visible to) the home file system, which greatly complicates file sharing between users. Always creating and accessing files on the "home" file server provides a single shared name space, enables file sharing, and ensures consistency, but introduces significant performance problems due to the long latency of remote file access. Also, access to a remote file system requires a stable network connection, which is often unavailable over WANs

In this paper, we describe *DataStations*, an architecture for providing file storage to mobile clients on servers physically located near the mobile client. A datastation is a self-managing file server that runs on a commodity PC and leases out storage space for a negotiated time to mobile users on demand. Multiple autonomous datastations distributed across the Internet cooperate to provide a wide-area file system with a decentralized uniform name space and several useful consistency semantics. Users can access, update, create files in their leased space from anywhere and link them into their home file system, all using their native file interface. A trusted agent runs at a user's home site to coordinate his file access via datastations without compromising security. Mobile users can utilize a nearby datastation in several ways: (i) as a staging area to cache files from far-away home file servers (or other datastations), (ii) as a file server to meet transient storage needs (e.g., to offload pictures from a digital camera), and (iii) to share data and collaborate with other users while on the move.

Providing transient localized file storage as a basic service to mobile users has several benefits. Previous work has shown that caching remote data near a mobile device can vastly improve access latency to logically remote file system data [3]. Reconciling updates with far away servers takes longer and consumes more power than reconciling with a nearby staging server, especially over lossy wireless links. Ubiquitous access to globally accessible file storage allows mobile users to roam freely without worry of losing access to their data. The shared file name space of data stations allows users to share data easily and collaborate

with other mobile users.

We envision public spaces such as airports, hotels, coffee shops and office buildings being equipped with datastations for the benefit of their customers or visitors. Datastations will be connected to the wired Internet via high-bandwidth links, while mobile clients typically access nearby datastations via a wireless network. However, for widespread deployment, datastations must require virtually no maintenance beyond initial setup. For widespread use, datastations must not be able to compromise the privacy and integrity of user data, the security of file servers with whom they interact, or the security of other machines in their local network.

Our architecture gives datastations complete autonomy in managing their resources subject to the constraint that users' data and/or updates are not discarded until they are propagated safely to another datastation or file server. Datastations transparently migrate files and their responsibility to other datastations or the remote file server and reclaim their storage resources.

To ensure privacy and integrity of data, we adopt an end-to-end approach based on encryption and secure hashing. For privacy, a mobile client establishes session keys with datastations as well as their home file server for communication over untrusted networks. The user-server session key is used to encrypt all sensitive data, e.g., file contents and directory path names, exchanged between the user's mobile client and its home server. Hence, datastations never see unencrypted user data. The know the replication state of user data, but not the actual data. To ensure data integrity, each file is accompanied by a secure hash of its contents that can be verified with the home server at any time. We assume that mobile clients, datastations, and home file servers have their own public-private key pairs, which they use to establish secure channels between themselves. Finally, a user can provide a datastation the means to authenticate itself with the user's home file server on the user's behalf by providing the datastation with an encrypted time-limited password that identifies the user and the intended (read-only or update) access privileges to the home server. A mobile user can supply similar passwords to let other users access their private data.

Datastations differ from existing data staging solutions in several novel ways. Unlike Fluid replication [7], datastations employ a location-aware peer replica network, so data can be served from a nearby copy (if available) without going through the home file server. Also, since a datastation is a full-fledged file server, it can be used to meet ad-hoc storage needs without having to synchronize with a separate home file server.

Datastations employ a novel approach to consistency management called *composable consistency* [15], that gives each replica significant control over consistency and availability.

Composable consistency gives applications direct control over consistency management decisions along several dimensions, each of which contribute to a portion of the overall consistency and availability semantics. Composable consistency allows datastations to support diverse data sharing patterns, and enables a variety of applications to benefit from the ubiquitous availability of transient storage.

In this paper, we focus on the file system aspects of providing ubiquitous transient storage. We show that datastations provide low-latency access to remote files, facilitate ad-hoc collaboration among mobile users and migrate data automatically to a user's current location avoiding the need to go through his home server whenever possible. In Section 2, we motivate the value of transient storage for mobile users by describing several real-life usage scenarios. In Section 3, we describe the datastations architecture and present our evaluation of its performance benefits in section 4. In Sections 5 and 6 we discuss related work and conclude.

2 Usage Scenarios

In this section we describe a few scenarios where the availability of ubiquitously accessible transient storage space would benefit mobile users.

Example: An executive in New York goes on vacation with her family to a beach resort in the Bahamas. She fills up her digital camera's memory with over 200MB of pictures and needs to free up space for more pictures. Unfortunately, the hotel's Internet link is down at that moment, so she cannot save the pictures to her home machine in New York. Fortunately, her hotel provides datastations to its customers. She saves all the 200MB worth of pictures to a datastation folder, leasing storage space for an extra day, and leaves for more fun. She takes more pictures and heads back home the next day. On the plane, she remembers that she forgot to download her pictures to her home machine. At a stopover in Florida, she accesses an airport datastation via her PDA to quickly drag and drop her hotel folder into the web folder on her home machine in NY. The operation is instantaneous. When the hotel's datastation later decides to evict his files, it discovers that they have a new parent folder homed on the NY file server. It therefore migrates the pictures to NY in 15 minutes and reclaims the space.

Example: A researcher in Europe attends a conference in the U.S. While at the conference venue, he wishes to collaborate with several fellow conference attendees to write a position paper for an upcoming workshop. They could host copies of the working draft on their laptops and synchronize manually, but hosting it on a datastation at the conference venue

would allow them to work independently and commit their changes at their own convenience. So they host the working draft on the datastation itself. Before leaving the hotel, one of them saves the draft to their home file server. When the colleagues disperse, they can still share the draft and work on it together as they visit other places.

Example: A reporter is at a public place such as an airport. He sees a newsworthy event occur and wants to provide a live video feed to his news headquarters. Fortunately, he can stream the video via his video camera's wireless link to a nearby datastation, which has high-bandwidth connectivity to his headquarters. Directly beaming the video from the camera to headquarters is impossible given the lack of sufficient WAN bandwidth directly between the mobile system and the home office.

3 Design and Implementation

3.1 Overview

The datastation service is organized as a peer-to-peer federated file system (called *fedfs*) where each peer datastation can autonomously create files and directories locally, cache remotely created files, and/or migrate files to other datastations. Figure 1 illustrates this organization. Each datastation runs a user-level fedfs daemon process that provides access to fedfs file system via the mount point `/fedfs`. Fedfs daemon serves files to local processes by interacting with Coda's kernel module [8] in place of its client-side daemon, Venus. To enable remote access to personal files, a mobile user also runs a special fedfs daemon called user agent on his home workstation. This agent transparently translates between fedfs' and the local file system's views of his files, while interacting with remote peer datastations. An identical fedfs user agent runs on the user's mobile client devices such as PDAs and laptops. However, on resource-limited devices such as PDAs, the client-side fedfs user agent can be configured to redirect all access requests to a nearby datastation and access files via block-oriented I/O just like an NFS client. This facilitates access to large files from the PDA in pieces. A fedfs user agent provides end-to-end encryption and user authentication in addition to serving as a fedfs daemon. Fedfs daemons internally store file copies in a private directory in the local file system, using the FID as the file name.

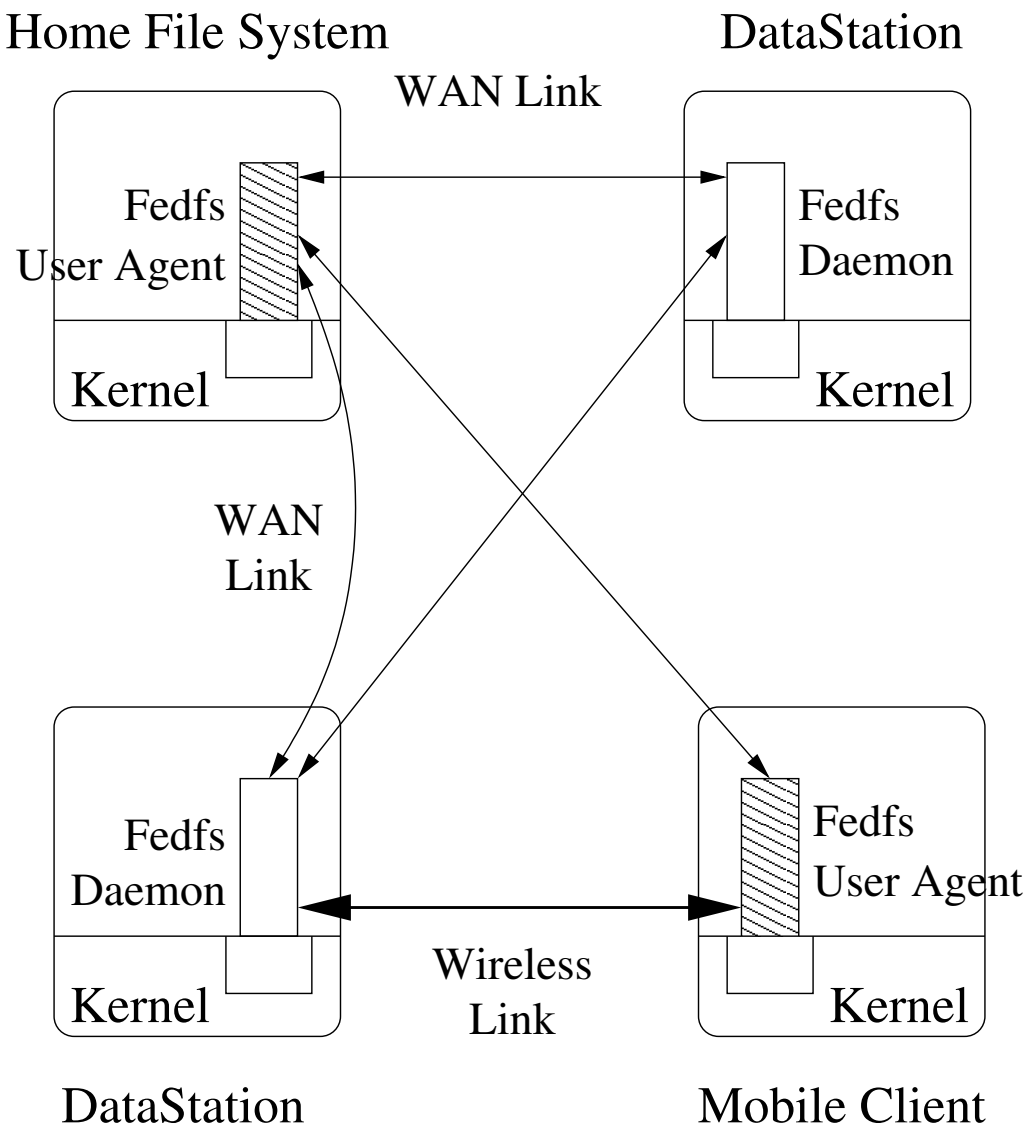

Figure 1: Datastations Architecture

File Naming

A datastation file is globally uniquely identified by a 128-bit number called a FID. A FID is a combination of the creating datastation's network ID (currently, the IP address), a unique node-local fedfs ID of the file and its generation number incremented everytime the ID is reassigned to a new file. A fedfs directory is a special file that keeps name-to-FID mappings and provides the traditional file naming hierarchy on top of FIDs. A typical fedfs path name looks like `/fedfs/fid:ds2.1234.5/home/me`, where `"fid:ds2.1234.5"` indicates the FID of a directory created at datastation with IP address `ds2`, file ID `1234` and generation number `5`. In practice, users can hide the numeric prefixes by creating symbolic links with more intuitive names.

Mobile users identify their files on home servers as well as on datastations by their fedfs path names. Their local fedfs user agents enable them to access fedfs files via the operating system's native file interface. Table 1 illustrates how a mobile client could access files via fedfs path names.

Interacting with a Datastation

When a mobile user requires transient storage, he directs his mobile client locates a nearby datastation using a resource discovery scheme such as JINI or by simple broadcast on the local network. The client registers with the selected datastation and sends it a space lease request (e.g., 500MB for 2 days) encrypted with its public key. In response, the datastation creates a new unique user ID, a local root directory and a session key for the client, and supplies them encrypted with the client's public key. The mobile client saves this information and can subsequently create files in the supplied root directory. It can also cache remote files by presenting their fedfs path names to the datastation after supplying a token (e.g., a time-limited password) to authenticate with the remote fedfs user agent. The datastation establishes a secure channel with the remote agent and supplies this token for all subsequent operations on behalf of this user.

A mobile user can also give other users access to his datastation/home files by supplying appropriate authentication tokens and fedfs pathnames. A user generates these tokens by directly contacting the servers. A mobile user cannot access other users' files on a datastation unless he knows their fedfs path and their authentication tokens.

Replication

By default, when a user requests a datastation to create a file even within a remote home directory, it creates the file locally and gives it a local FID. Datastations refer to files by their FIDs for all internal communication. A file's creating node is called its custodian. It is responsible for tracking the location of the file's primary copy (also referred to as the root copy or home copy hereafter). When a datastation needs a file not present locally, it requests another known file copy (the custodian by default, as its IP address is hardcoded in the FID). The responder can supply the requestor a copy by ifself, in which case they form a parent-child relationship. It can also forward the request to a few randomly selected children or send a list of its children to the requestor to be queried further. In the last case, the requestor selects the "nearest" replica (in terms of network quality) and sends it the request. This recursive process builds a dynamic replica hierarchy rooted at the current primary copy. Also, our peer replica creation mechanism from other nearby replicas provides low latency access to files by avoiding slow links when possible, unlike centralized schemes (e.g., Fluid replication, AFS, Coda) that always need to pull files from a central home server.

Replicas continually monitor the network link quality (currently RTT) to other known replicas and rebind to a new parent if they find a replica closer than the current one. The fanout of any node in the hierarchy is limited by its load-handling capacity. When a link or node in the hierarchy goes down, the roots of the orphaned subtrees try to re-attach themselves to the hierarchy, starting at a known copy site (home node by default), effectively repairing the replica hierarchy.

Lastly, all consistency-related communication happens only along a file's replica hierarchy, avoiding duplicate messages. Further details on the replication mechanism are given elsewhere [16].

Migration

One important property of datastations is that they only provide transient storage and must be able to reclaim storage resources autonomously to allow self-management and scaling. To ensure this, all files created by a user at a datastation must eventually be migrated elsewhere or discarded before the user's space lease expires.

If a file is merely cached locally, the datastation can simply evict it after propagating outstanding updates to its parent and informing its neighboring replicas that it is going away. However, if the datastation is holding the root copy of a file, it first needs to identify a new

node to transfer the file's root replica responsibilities. Datastations employ a simple heuristic to select a new root. Each file's parent directory entry is maintained in its metadata. A datastation uses this information to decide where to migrate a file. If any directory along the path of a file to its top-level directory is is known to be rooted at a different node, then the file is also migrated to that node. With this scheme, a user can ensure permanence of all files created in a datastation's temporary folder by simply moving that folder to his home file system. Failing that, if a file has other replicas, one of the replicas is chosen to be the new root replica. With this scheme, transient files created by a user at a datastation can be kept alive and made to follow the user as he roams between datastations. If no other replica exists, the file is simply deleted as it cannot be reached from any permanent fedfs path.

Even after a datastation migrates the root replica responsibilities elsewhere, it still keeps a small forwarding entry to redirect future queries for the file by its FID to its new root, as the FID hardcodes the original creating node's network ID. The FID itself can be released for reuse after allocating a new FID for the file and fixing its entry in the parent directory to refer to the new FID. Datastations currently forbid a mobile user moving the root directory supplied at registration time elsewhere, as it is difficult to fix its FID permanently after a migration.

Consistency

Datastations employ a novel approach to consistency management called *composable consistency* [15], that gives each replica significant control over consistency and availability. Our approach gives applications direct control over consistency management along several orthogonal dimensions namely, concurrency control, timeliness of updates, update ordering, update visibility, reader isolation from remote updates as well as data availability during disconnections. It provides several useful choices to manage each of these aspects of consistency management. Each of these contribute to a portion of the overall consistency and availability semantics. For instance, when an application opens a file, it can request the local copy to be immediately brought up-to-date at open() time, concurrent writers allowed elsewhere, local writes to be made visible to other replicas only when the session ends, their writes not to be incorporated locally until the session ends. These choices provide close-to-open consistency. On the other hand, by specifying locking mode for writes, an application can prevent conflicting writes, whereas non-locking reads can still proceed to read dirty data. In particular, composable consistency enables datastations to provide on a per-file replica basis: the strong consistency semantics of Sprite [14], the close-to-open consistency of AFS [5] and Coda [8] and the (weak) eventual consistency of Coda, Pangaea [12], and NFS [13]. Datastations can thus support diverse data sharing patterns, and enabling a variety of applications to benefit from the ubiquitous availability of transient

storage.

Datastations implement composable consistency by exchanging (read and write) access privileges among themselves in response to file replica accesses at their sites, via the replica hierarchy. They avoid unnecessary synchronization traffic by caching privilege locally until another replica requests it. This lazy approach enables them to exploit inherent locality in the application, as we demonstrate in section 4.3.3. For more details, the reader is referred elsewhere [16].

4 Evaluation

We conducted a series of experiments to evaluate the performance of datastations in different usage scenarios ranging from personal file access to widespread collaboration among multiple users. Our first experiment simulates the usage scenario 1 mentioned in section 2. We show that datastations can effectively provide ubiquitously accessible storage while exploiting available locality well. Next, we show the effectiveness of peer replication in supporting proxy caching of remote files as users roam across continents. Lastly, we show how multiple developers spread across continents can collaborate on a software project by safely sharing the RCS repository of the source code without compromising consistency.

4.1 Experimental Setup

For all our experiments, we used the University of Utah's Emulab Network Testbed [2]. Emulab allows us to model a collection of PCs connected by arbitrary network topologies with configurable per-link latency, bandwidth, and packet loss rates. The PCs had 850Mhz Pentium-III CPUs with 512MB of RAM. Depending on the experimental requirements, we configured them to run FreeBSD 4.7 (BSD), or Redhat Linux 7.2 (RH7.2). In addition to the emulated network, each PC is connected to a 100Mbps control LAN isolated from the emulated network. Clients and servers log experimental output to an NFS server over the control network. All logging occurs at the start or finish of an experiment to minimize interference.

Client at	Peer mode cmds	Home mode cmds
H	ln -s /fedfs/H.xyz.3 ~/myhome;	mkdir pictures
D1	ln -s /fedfs/D1.abc.2 ~/d1_dir; cd ~/d1_dir	mv camera/*.jpg ~/myhome/pictures
D1	mkdir pics; mv camera/*.jpg pics/	
D2	cd ~/d1_dir/pics	cd ~/myhome/pictures
D2	view 01.jpg ... 25.jpg	view 01.jpg .. 25.jpg
D2	mv 01.jpg 03.jpg .. 25.jpg ~/myhome/pictures	
D2	rm 02.jpg 04.jpg .. 24.jpg	rm 02.jpg 04.jpg .. 24.jpg
D3	cd ~/d1_dir/newpics	cd ~/myhome/pictures
D3	view 26.jpg .. 50.jpg	view 26.jpg .. 50.jpg
D3	mv 27.jpg 29.jpg .. 49.jpg ~/myhome/pictures	
D3	rm 26.jpg 28.jpg .. 50.jpg	rm 26.jpg 28.jpg .. 50.jpg

Table 1: A mobile client's Unix commands for Experiment 1

4.2 Ubiquitous File Storage and Migration

Our first experiment implements the following scenario. A user visits a city far away from home, takes 50 pictures each averaging 1MB in size using his digital camera. He copies them into the temporary storage folder provided a nearby datastation D1 after negotiating a space lease for two days (called the create-D1 phase). The next day, he moves to a different city and browses 25 of his uploaded pictures via a different datastation D2. He removes half of them and saves the other half by moving them to the "pictures" folder on his home machine's file server H ("browse-D2" phase). The user's space lease on D1 expires that night. D1 migrates the moved pictures to his home location H, and the remaining 25 not yet accessed to D2 (evict-D1 phase). Finally, the next day he moves to a different place with a datastation D3, browses the rest, saving and discarding pictures as on D2 (browse-D3 phase). Table 1 lists the commands executed from a mobile client at various sites.

We employed the network topology shown in figure 2. The datastations D1, D2 and D3 are connected to a common backbone router R1 via 5Mbps, 20ms RTT links, resulting in 40ms RTT between each pair of them. The home file server H is located in a different city with a 1Mbps link and is 140ms RTT away from each of them. We ran FreeBSD 4.7 on all the nodes.

We repeated the above experiment with a small change. In the upload phase at D1, the user uploads files into a folder on the home file server itself to avoid losing the files in case of inoppurtune lease expiration. Figure 3 shows the latency of each of the four phases in both of the experimental scenarios. The first scenario (saving to local folder) is labelled as "peer" and the second scenario as "home" in the figure.

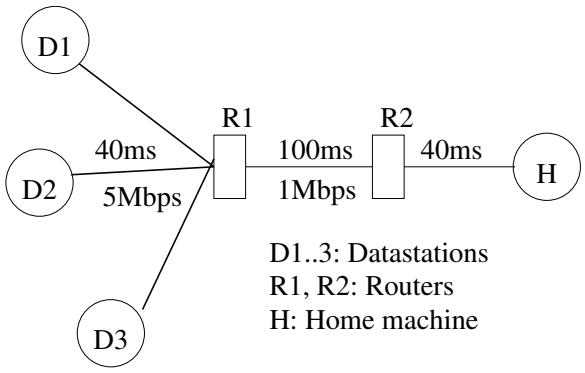

Figure 2: Topology for Migration Experiments

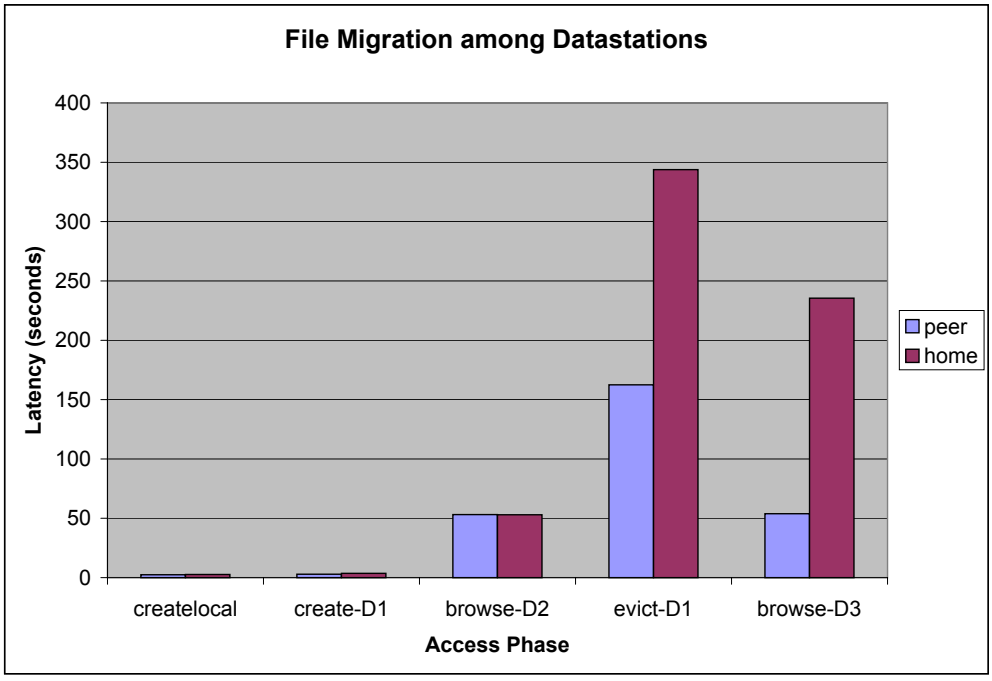

Figure 3: File Migration among Datastations: ”home” means pictures are initially copied into a home folder, ”peer” means they are copied to a temporary folder on D1. In ”peer” case, files ’follow’ the roaming user and access incurs low latency. In ”home” case, they need to be accessed from home file server causing slowdown.

Uploading 50 pictures into a folder on the home server takes almost the same time as saving them to a temporary folder on D1. This is because, D1 creates picture files locally and only adds their names into the home server's folder. When the first 25 files are accessed at D2, they are all supplied from D1 via the 5Mbps link in both scenarios. Subsequently, when files are evicted from D1, "home" takes much longer, as D1 migrates a total of 37 1MB files to the remote home server H via its 1Mbps link. In the "peer" case, D1 first migrates custody of the temporary folder to D2, as D2 cached it recently. It gets rid of the files as follows. Since 13 of the first 25 files have been renamed by D2 to a home site folder, it migrates them to home, incurring roughly 110 secs of latency. It migrates the remaining 25 unread pictures to D2, since it is the current custodian for the temporary picture folder.

In effect, in the "peer" case, as the user moved from D1 to D2, data followed him. The benefit of pushing the remaining 25 files to D2 is that when the user subsequently browses them on D3, the files are supplied by D2 in the "peer" case. On the other hand, in the "home" case, the eviction pushed all files to home and D3 has to obtain them from home via the slow 1Mbps link, incurring roughly 5x slowdown in access.

4.3 Data Sharing and Collaboration

In this section, we further explore the data sharing and consistency capabilities of datastations under two distinct usage scenarios: personal file access in the presence of roaming, and collaboration among multiple users.

First, in Section 4.3.1, we consider the case of a single client and server connected via a high-speed LAN. This study shows that the inherent inefficiencies of adding an extra level of middleware to a LAN file system implementation has very little impact on performance. Our second and third experiments focus on sharing files across a WAN. In Section 4.3.2 we consider the case where a set of files are accessed sequentially on series of widely separated datastations. This study shows that datastations' ability to satisfy file requests from the "closest" replica can significantly improve performance. Finally, in Section 4.3.3 we consider the case where a shared RCS repository is accessed in parallel by a collection of widely separated developers. This study shows that datastations' access privilege caching not only provides correct file locking semantics required by RCS, but can also enable datastations to fully exploit available locality by caching files near frequent sharers.

For all these experiments, we ran Redhat Linux 7.3 on all the computers.

4.3.1 Local Client-Server Performance

To provide a performance baseline, we first study the performance of datastations' fedfs file system and several other representative distributed file systems in a simple single-server, single-client topology. In this experiment, a single client runs Andrew-tcl, and scaled up version of the Andrew benchmark on a file directory hosted by a remote file server, starting with a warm cache. Andrew-tcl consists of five phases, each of which stresses a different aspect of the system (e.g., read/write performance, metadata operation performance, etc.). For this experiment, there is no inter-node sharing, so eventual consistency suffices.

We ran the Andrew-tcl benchmark on four file systems: the Redhat Linux 7.2 local file system, NFS, Coda, and fedfs. For fedfs, we considered two modes: *dpeer*, where file creation requests are satisfied locally, *dcache*, where the datastation is configured to forward file creation requests to the home file server and merely cache remotely homed files. Figure 4 shows the relative performance of each system where the client and server are separated by a 100-Mbps LAN, broken down by where the time is spent. Execution time in all cases was dominated by the compute bound compile phase, which is identical for all systems. Figure 5 focuses on the other phases. As expected, the best performance is achieved running the benchmark directly on the client's local file system. Among the distributed file systems, NFS performed best, followed by dpeer, but all distributed file systems performed within 10%. dpeer performed particularly well during the data-intensive `copy` and `mkdir` phases of the benchmark, because files created by the benchmark are homed on the client node[1]. Coda's file copy over LAN takes twice as long as dcache due to its eager flushes of newly created files to the server.

4.3.2 Sequential Wide Area Access (Roaming)

In our second experiment we focus on a scenario in which files are shared in migratory fashion across a WAN. For instance, this could illustrate a researcher visiting a series of remote campuses on an extended trip, while accessing his home files. Another applicable scenario is where documents are circulated in a migratory fashion among widespread peers for review. For this experiment, we assume a network topology like that illustrated in Figure 6 consisting of widely distributed five sites, each of which contains two nodes connected via a 100Mbps LAN.

[1]The performance of fedfs metadata operations, which are used heavily in the `grep` and `stat` phases of the benchmark, is poor due to a flaw in our current fedfs implementation. We do not fully exploit the Coda in-kernel module's ability to cache directory data. When we enhance fedfs to incorporate this optimization, fedfs performance during the metadata-intensive phases of the benchmark will match Coda's performance.

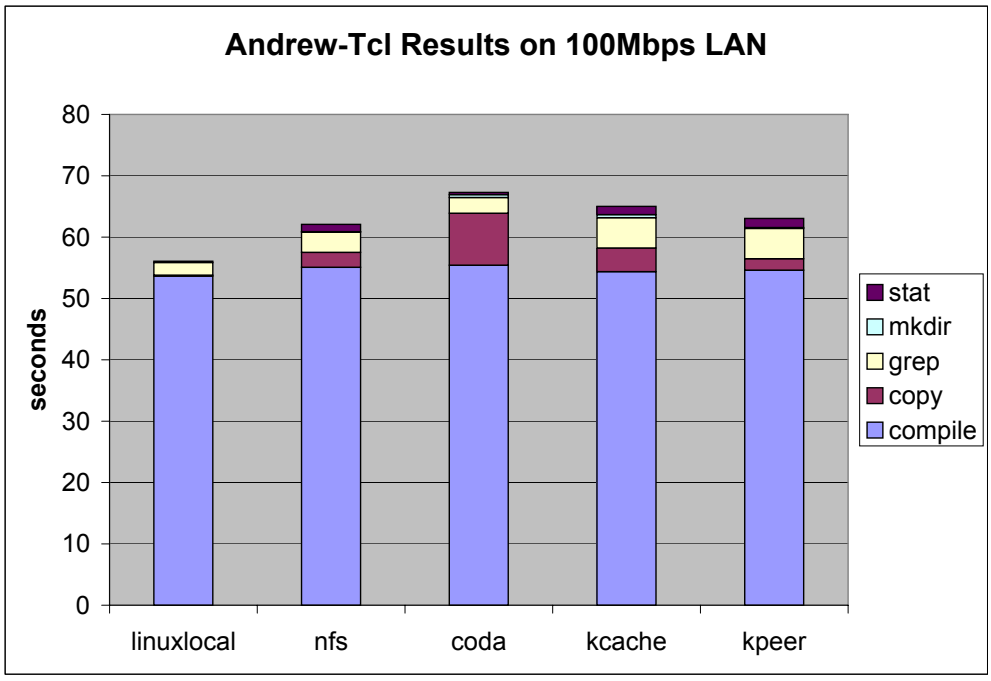

Figure 4: Andrew-Tcl Results on 100Mbps LAN

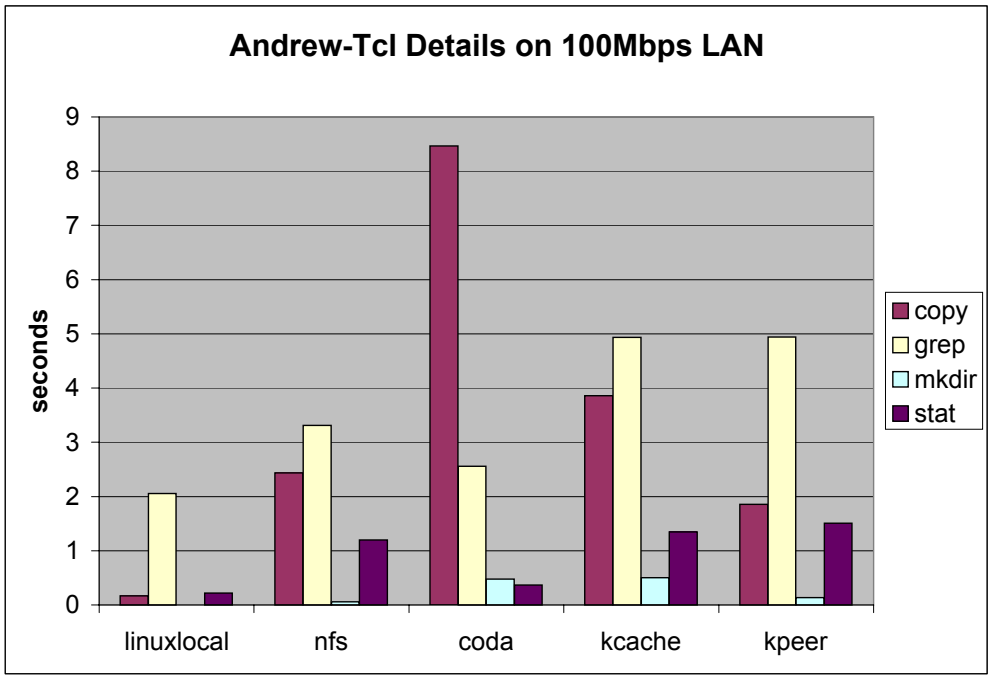

Figure 5: Andrew-Tcl Details on 100Mbps LAN

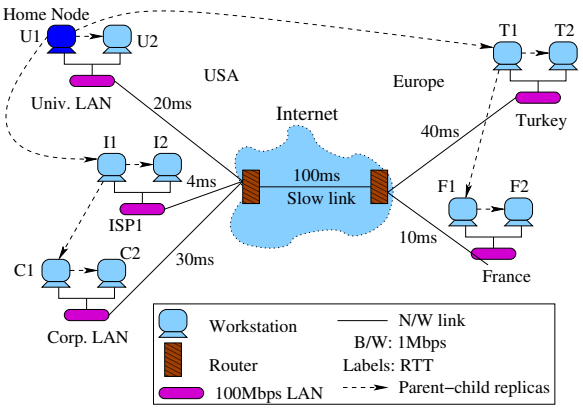

Figure 6: Topology for Sharing Experiments

Each of the ten nodes run the compile phase of the Andrew-tcl benchmark in turn on a shared directory to completion, followed by a "`make clean`". First one node on the University LAN runs the benchmark, then its second node, then the first node in ISP1, etc., in the order University (U) → ISP1 (I) → Corporate (C) → Turkey (T) → France (F). Each node starts with a cold file cache. The primary copy of the Andrew-tcl benchmark tree is hosted by a fedfs agent on the node marked "Home Agent" on the University LAN.

We performed this experiment using three distributed file systems: fedfs in peer-to-peer mode, Coda in strongly-connected mode (Coda-s), and Coda in adaptive mode (Coda-w). fedfs was configured to provide close-to-open consistency. Coda-s provides strong consistency, whereas Coda-w quickly switched to weakly connected operation due to the high link latencies. During weakly connected operation, coda-w employed trickle reintegration to write back updates to the server "eventually".

Figure 7 shows the time each node took to perform the compile phase of the Andrew-tcl benchmark. As reported in the previous section, both fedfs and Coda perform comparably when the file server is on the local LAN, as is the case on nodes U1 and U2. However, there are two major differences between fedfs and Coda when the benchmark is run on other nodes. *First, fedfs always pulls source files from a nearby replica,* whereas Coda clients always pull file updates through the home server incurring WAN roundtrips from every client. As a result, Coda clients suffered 2x-5x higher file access latency than fedfs clients. *Second, fedfs was able to provide "just enough" consistency to implement this benchmark efficiently but correctly, whereas the two Coda solutions were either overly conservative (leading to poor performance for coda-s) or overly optimistic (leading to incorrect results for coda-w).*

Fedfs had a number of advantages over coda-s. One was the aforementioned ability to read a source file from any replica, not just the home node. This flexibility was especially important when the benchmark was run either on the second node of a LAN or run for the second (or subsequent) time in "Europe". Also, file creation in coda-s is mediated by the home server, which leads to poor performance when the latency to the server is high, such as is the case for C1-F2. The net result is that the benchmark ran 2-4X faster on fedfs than on coda-s on the WAN clients, as might be expected given that coda-s is not intended for WAN use.

The comparison between fedfs and coda-w illustrates the importance of having user-configurable consistency policies. In adaptive mode, if Coda determines that the client and server are weakly connected, it switches to 'eventual consistency" mode, wherein changes to files are lazily propagated to the home server, from which they are propagated to other replicas. Unfortunately, in this scenario, that degree of consistency is insufficient to ensure correctness. Reintegrating a large number of object files and directory updates over a WAN link takes

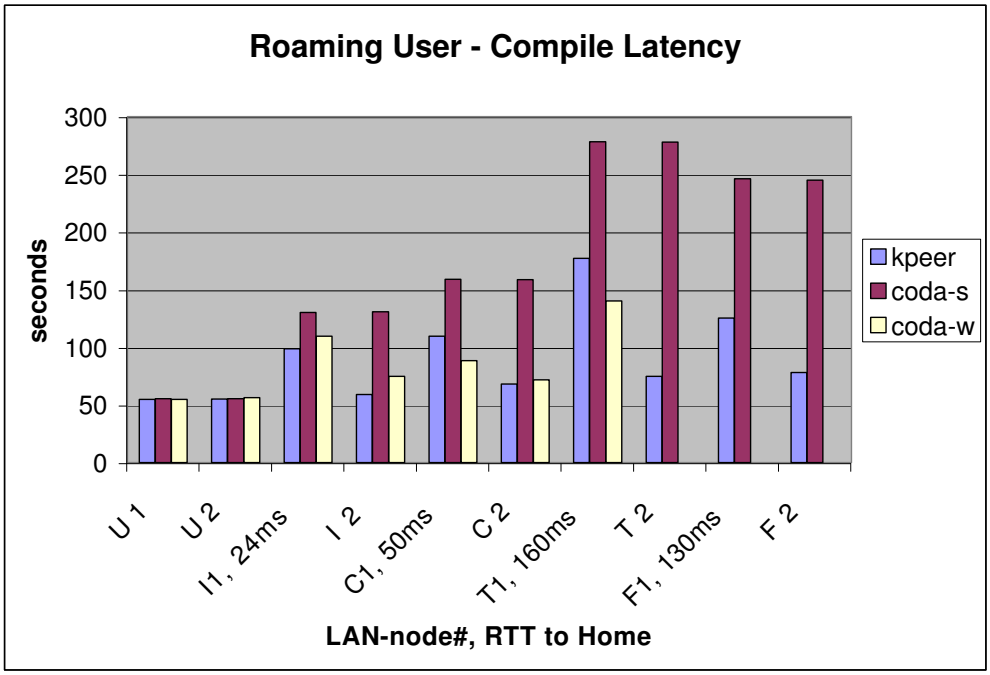

Figure 7: Compile Phase: Fedfs pulls files from nearby replicas. Strong-mode CODA behaves correctly, but exhibits poor performance. Weak-mode CODA performs well, but generates incorrect results on the final three nodes.

time. If a second benchmark run starts before all changes from the previous run have been pushed to the currently active client, conflicts occur. In this case, Coda reported an update conflict, which requires manual intervention. If these conflicts are ignored, delete messages associated with intermediate files created by earlier nodes are not integrated in time, which leads later nodes to incorrectly assume that they do not need to recompile the associated source files. In contrast, Khazana enforces fedfs's desired close-to-open consistency policy on each file, thereby ensuring correct operation regardless of contention.

4.3.3 Simultaneous WAN Access

We next explore the value of datastations for collaboration among multiple users over the wide area. An example scenario is multiple widely separated colleagues working on a paper or closely collaborating on a software development project. A version control system is typically employed to coordinate sharing in such cases. Version control systems rely on reliable file locking or atomic file/directory operations to synchronize concurrent read/write accesses. However, the atomicity guarantees required by these operations are not provided by most wide area file systems across replicas. As a result, such applications cannot benefit from caching, even if they exhibit high degrees of access locality.

For example, the RCS version control system uses the exclusive file creation semantics provided by the POSIX open() system call's O_EXCL flag to gain exclusive access to repository files. During a checkout/checkin operation, RCS attempts to atomically create a lock file and relies on its pre-existence to determine if someone else is accessing the underlying repository file. Coda's close-to-open consistency semantics is inadequate to guarantee the exclusive file creation semantics required by RCS. Thus hosting an RCS repository in Coda could cause incorrect behavior. In contrast, datastations can provide strong consistency by caching locking privileges along with file data in a peer-to-peer fashion. This ensures correct semantics for repository directory and file updates required by RCS, while exploiting locality in file accesses for low latency. Thus datastations enable users to safely and efficiently share RCS files across a WAN, and facilitate collaboration.

We evaluated two versions of RCS, one for which the RCS repository resides in fedfs (*peer sharing mode*) and one in which the RCS repository resides on U1 and is accessed via ssh (*client-server/RPC mode*).

To illustrate how fedfs performs in the face of concurrent file sharing, we simulated concurrent development activities on a project source tree using RCS for version control. [2] For this set of experiments, we used a simplified version of the topology shown in Figure 6 without the ISP1 LAN (I). The "Home Node" initially hosts three project subdirectories from the Andrew-tcl benchmark: `unix` (39 files, 0.5MB), `mac` (43 files, 0.8MB), and `tests` (131 files, 2.1MB).

Our synthetic software development benchmark consists of six phases, each lasting 200 seconds. In Phase 1 (widespread development), all developers work concurrently on the `unix` module. In Phase 2 (clustered development), the developers on the University and Corporate LANs switch to the `tests` module, the developers in Turkey continue work on

[2] We chose RCS, rather than CVS, because RCS employs per-file locking for concurrency control and hence allows more parallelism than CVS, which locks the entire repository for every operation.

the `unix` module, and the developers in France switch to the `mac` module. In Phases 3-6 (migratory development), work is shifted between "cooperating" LANs – the `unix` module migrates between the University and Turkey, while the `mac` module migrates between Corporate LAN and France (e.g., to time shift developers). During each phase, a developer updates a random file every 0.5-2.5 seconds from the directory she is currently using. Each update consists of an RCS checkout, a file modification, and a checkin.

Figure 8 shows the checkout latencies observed from clients on the University LAN, where the master copy of the RCS repository is hosted. Figure 9 shows the checkout latencies observed from clients on the "Turkey" LAN. The checkout latencies were fairly consistent at each node in client-server mode. Therefore, we plotted the average latency curve for each node on both graphs. The checkout latencies in peer sharing mode were heavily dependent on where the nearest replica was located and the amount of work needed to maintain consistency between replicas, so we provide a scatter-plot of all checkout latencies in this mode.

Overall, our results indicate that *fedfs enables RCS developers to realize the performance benefits of caching when there is locality, while ensuring correct operation under all workloads and avoiding performance meltdowns when there is little to no locality*. This is shown by the fact that checkout latency under clustered development (i.e., phases 3 and 5 of Figure 8, and phases 2, 4 and 6 of Figure 9) quickly drop to that of local RCS performance observed by U1 (shown in Figure 8). At low locality (as in Phase 1 for all developers, and Phase 2 for U1-C2), RCS on fedfs still outperforms client-server RCS. RCS on fedfs' latency is close to two seconds or less for all developers, while that of client-server RCS degrades in proportion to the latency between the client and central server. This is because fedfs avoids using the slow WAN link as much as possible. Finally, fedfs responds quickly to changes in data locality.

When the set of nodes sharing a file changes, it is migrated to the new set of sharers fairly rapidly. This phenomenon is illustrated by the initial high checkout latency for each node during Phases 3-6, which rapidly drops to local checkout latency once datastations cache the new working set locally. The time at the beginning of each phase change when nodes see high checkout latency represents the hysteris in the system, whereby datastations do not migrate data with low locality.

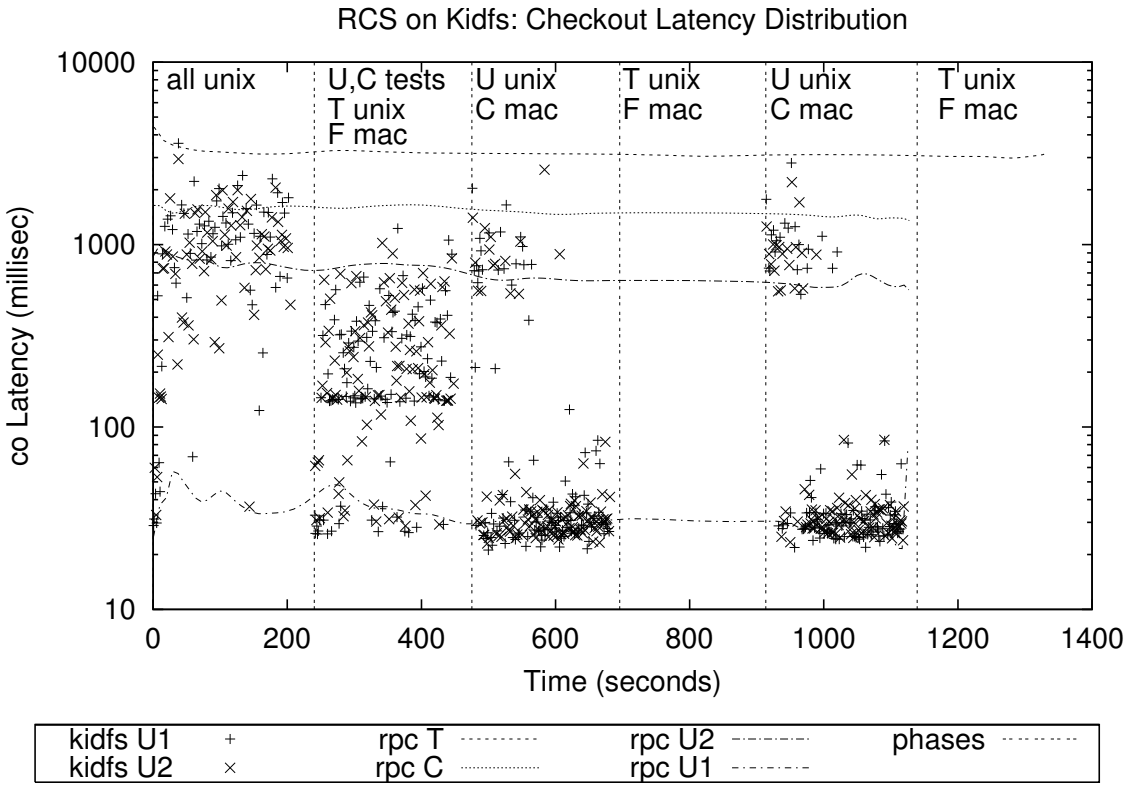

Figure 8: RCS on fedfs: Checkout Latencies on the “University LAN”

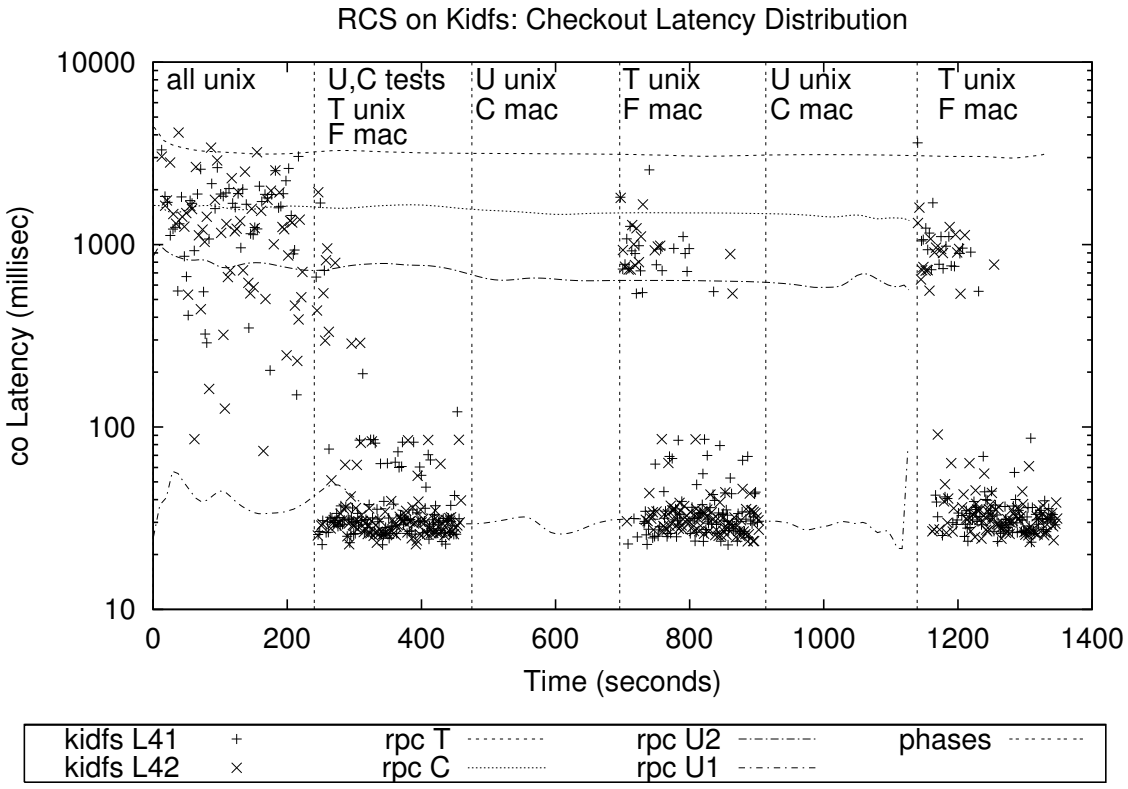

Figure 9: RCS on fedfs: Checkout Latencies from “Turkey”

5 Related Work

Many existing distributed file systems provide low latency file access to users over the wide area [8, 12, 10]. However, they provide these benefits only as long as the mobile user roams within their administrative domain. Several existing solutions provide data staging at untrusted intermediary machines to improve latency of remote file access by mobile users. However, our goals extend beyond caching remote data near a a mobile user to also providing coherent file storage via autonomously managed distributed file servers. Flinn et al [3] describe a way for mobile clients to cache remote files on nearby untrusted surrogate machines. Their surrogates do not take responsibility for updates. Hence a mobile client still has to perform reconciliation to remote servers by itself. Their scheme forces reconciliation always to home servers, which is restrictive. Though their approach reduces trust requirements on surrogates, we believe that offloading reconciliation to a surrogate enables much more oppurtunities for sharing and mobility.

Waystations in Fluid replication [7] perform a similar data staging role and also accept file updates for background reconciliation. However, waystations reconcile only with the remote file server but not with each other. A user migrating to a nearby waystation has to pull file modifications through the server, potentially incurring high latency.

Oceanstore [11] provides floating replicas of data with the ability to migrate to nearby servers. Oceanstore classifies servers into those that can be trusted to perform replication protocols and those that cannot be trusted. Though untrusted servers can accept updates, they cannot be committed until the client directly contacts the trusted servers and confirms them. Oceanstore's main goals are secure sharing as well as long-term durability of data, and for this they incur poor write performance. In contrast, our goal is to provide transient storage with efficient ad-hoc sharing.

Several systems (ROAM [9], Bayou [1]) support epidemic replication where mobile clients can synchronize directly with each other without having to go through central servers much of the time. We believe that providing wired hosts such as datastations with good Internet connectivity enhances the data sharing capabilities of mobile hosts, while at the same time, relieving them from having to rely on remote servers.

6 Conclusions

In this paper we described a novel architecture called Datastations that provides ubiquitous transient storage to arbitrary mobile applications. A Datastation can be utilized by mobile users as a proxy cache for their remote file servers, as a file server to meet transient storage needs, and as a platform to share data and collaborate with other mobile users. The unique aspects of datastations are its decentralized but uniform file name space, locality-aware peer replication with complete autonomy to peers and flexible consistency choices to support a variety of sharing modes. We demonstrated several usage scenarios enabled by Datastations that are difficult to achieve with existing systems.

We believe that providing ubiquitous storage is an important step towards Pervasive computing [4], as it makes application state ubiquitously accessible, liberating computation from being dependent on a static data repository. Datastations enable mobile users to offload data and application state. Our next logical step is to leverage the ubiquitous accessibility of application state to facilitate offloading computation as well, by providing computing surrogates as an infrastructural utility. However, this raises security and resource management challenges more complex than raw storage. We plan to leverage the vast body of existing work in this area (e.g., the Rover toolkit[6]).

References

[1] A. Demers, K. Petersen, M. Spreitzer, D. Terry, M. Theimer, and B. Welch. The Bayou architecture: Support for data sharing among mobile users. In *Proceedings of the Workshop on Mobile Computing Systems and Applications*, Dec. 1994.

[2] Emulab. `http://www.emulab.net/`, 2001.

[3] J. Flinn, S. Sinnamohideen, N. Tolia, and M. Satyanarayanan. Data staging on untrusted surrogates. In *Proceedings of the 2nd USENIX Conference on File and Storage Technologies (FAST03)*, Mar. 2003.

[4] R. Grimm, J. Davis, B. Hendrickson, E. Lemar, A. MacBeth, S. Swanson, T. Anderson, B. Bershad, G. Borriello, S. Gribble, and D. Wetherall. Systems directions for pervasive computing. In *Submitted to the 8th Workshop on Hot Topics in Operating Systems (HotOS-VIII)*, 2001. `http://www.cs.washington.edu/homes/gribble/papers/one-hotos.pdf`.

[5] 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):51–82, Feb. 1988.

[6] A. Joseph, J. Tauber, and M. F. Kaashoek. Mobile computing with the Rover toolkit. *IEEE Transactions on Computers: Special issue on Mobile Computing*, 46(3), Mar. 1997.

[7] M. Kim, L. Cox, and B. Noble. Safety, visibility and performance in a wide-area file system. In *Proceedings of the 1st USENIX Conference on File and Storage Technologies (FAST02)*, Jan. 2002.

[8] J. Kistler and M. Satyanarayanan. Disconnected operation in the Coda file system. In *Proceedings of the 13th Symposium on Operating Systems Principles*, pages 213–225, Oct. 1991.

[9] D. H. Ratner. ROAM: A scalable replication system for mobile and distributed computing. Technical Report 970044, University of California, Los Angeles, 31, 1997.

[10] P. Reiher, J. Heidemann, D. Ratner, G. Skinner, and G. Popek. Resolving file conflicts in the Ficus file system. In *Proceedings of the 1994 Summer Usenix Conference*, 1994.

[11] S. Rhea et al. Pond: The oceanstore prototype. In *Proceedings of the 2nd USENIX Conference on File and Storage Technologies (FAST03)*, Mar. 2003.

[12] Y. Saito, C. Karamanolis, M. Karlsson, and M. Mahalingam. Taming aggressive replication in the Pangaea wide-area file system. In *Proceedings of the Fifth Symposium on Operating System Design and Implementation*, pages 15–30, 2002.

[13] R. Sandberg, D. Goldberg, S. Kleiman, D. Walsh, and B. Lyon. Design and implementation of the SUN Network Filesystem. In *Proceedings of the Summer 1985 USENIX Conference*, pages 119–130, 1985.

[14] V. Srinivasan and J. Mogul. Spritely NFS: experiments with cache-consistency protocols. In *Proceedings of the 12th Symposium on Operating Systems Principles*, pages 45–57, Dec. 1989.

[15] S. Susarla and J. Carter. Composable consistency for large-scale peer replication. Technical Report UUCS-03-025, University of Utah School of Computer Science, Nov. 2003.

[16] S. Susarla and J. Carter. Khazana: A flexible wide-area data store. Technical Report UUCS-03-020, University of Utah School of Computer Science, Oct. 2003.