Device Transparency: a New Model for Mobile Storage

Jacob Strauss Chris Lesniewski-Laas Justin Mazzola Paluska
Bryan Ford[‡] Robert Morris Frans Kaashoek

Massachusetts Institute of Technology [‡]Max Planck Institute for Software Systems

ABSTRACT

This paper proposes a new storage model, *device transparency*, in which users view and manage their entire data collection from any of their devices, even from disconnected storage-limited devices holding only a subset of the entire collection.

1. INTRODUCTION

Users increasingly own multiple devices running various media applications, but managing data objects across devices remains largely manual and error-prone. Collections easily become disorganized and drift out of sync.

To address this problem, we propose a new principle, *device transparency*, for the design of storage systems. We argue that all of a user's devices should present a single coherent view of the user's data collection, even when each of those devices can only store a fraction of the objects.

We propose that device transparency can be achieved by distributing all application metadata globally, so that each device knows about all objects, including both those stored locally and not. This "metadata-everywhere" technique enables a system to provide the appearance of a unified collection which can be organized using any device. When some devices are unavailable, because they are powered off or disconnected from the network, the system can also use the global metadata to inform the user where data objects are actually stored.

2. A CASE FOR DEVICE TRANSPARENCY

Users often own many devices that combine storage, networking, and multiple applications managing different types of data, such as photos, music tracks, videos, and email messages. In this paper, we focus on situations where some small devices, such as mobile phones, pocket cameras, or media players, may not have enough storage for all of a user's data objects. Generally, the user manually partitions the collection across the devices, choosing which objects will be replicated where.

Many existing storage systems designate one more-powerful device, such as a laptop or network server, as a central hub which stores master replicas of the complete collection. Mobile devices store partial replicas and propagate all updates via the hub. This model permits consistent access to replicated objects, but generally does not provide a unified view to the user: a mobile device can only manage those objects which are stored locally. In addition, a centralized system becomes unavailable when the hub device is unreachable or if it runs out of space to store the entire collection.

Adding only a few additional devices results in a much more difficult task. Consider a user with a laptop, the mobile devices mentioned above, a desktop computer at home and at work, and perhaps a DVR, a networked storage device, and some space in a highly-available cloud service. This set of devices no longer has an obvious master device that can store a single complete copy and mediate all updates. Any of the potential master devices may often be powered off, lie beyond slow network links, or have no convenient object management interface.

Since a hub storage system won't suffice, the user is forced to manage her collection in an *ad hoc*, manual way. As a result, the user sees a storage abstraction that looks like "object a on device x", "object b on device y," etc.: it is up to the user to keep track of where her objects live and whether a and b are different objects, copies of the same object, or different versions of the same object. Because her devices have limited resources, the user also bears the burden of organizing her object collection, larger than a single device's storage, into coherent groups that span all of the devices.

We believe that users would be better served by systems that present a single storage collection across multiple devices. Inspired by location transparency [27] for network file systems, we call this principle *device transparency*: a user should be able to manage all of her objects from any of her devices. As much as possible, a user should be able to think of her data as a unified collection rather than as isolated fragments on various devices. For example, when a user needs an object which is not stored on the local device, she should be able to explore the entire object collection (including objects not stored locally) to find it. The system should tell her what devices are responsible for the object's contents, and should download a local copy automatically.

At present we are unaware of any practical device-transparent storage system. A number of systems replicate and synchronize all data on all devices [11, 26], thus providing device transparency at the cost of requiring that all devices be large enough to hold all data. Other systems support partial replication for storage-limited devices but do not provide device transparency [13, 15, 20, 23] because disconnected devices do not present the user with a view of the entire data collection.

The ideal device transparent storage system would, at all times and on all devices, present an identical, complete collection of objects to the user; changes would propagate instantly and atomically. Of course, the constraints imposed by the physical world on devices

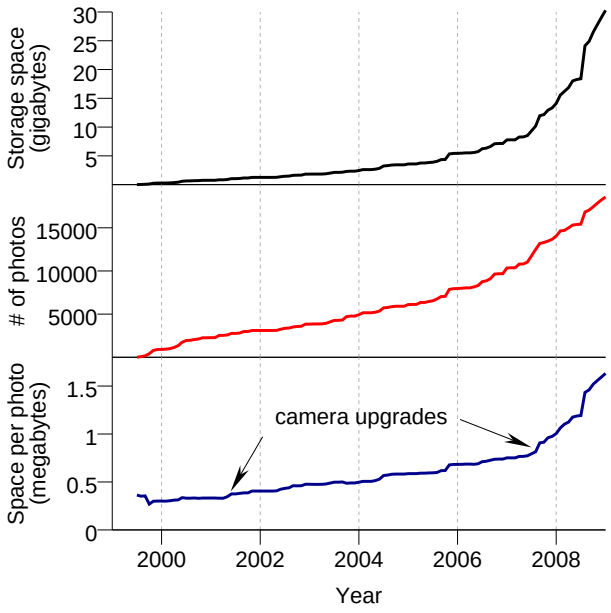

Figure 1: Growth of one user's photograph collection over 10 years. As digital cameras improved, the total storage consumed (top) increased exponentially, but the cumulative number of photographs taken (middle) grew more gradually. As a result, the average content size of a photograph in the collection (bottom) increased, while typical metadata per photograph remained the same.

and networks make it impossible to achieve this ideal completely. Some devices may be only intermittently connected, so they may not know about recently created or modified objects, and independent modifications on disconnected devices may lead to later conflicts. Devices may not have enough local storage to hold a copy of every object; if a device is temporarily disconnected, some objects may become inaccessible for the interim. Even subject to these fundamental limitations, we believe that it is possible to come substantially closer to the ideal of device transparency than existing systems do, and that a personal storage system built around this goal will greatly improve a user's ability to manage a collection of devices and data.

3. METADATA EVERYWHERE

Each device in a device-transparent storage system must know about every object in the storage collection in order to show the user the complete collection even when that device cannot communicate with the rest of the group. As such, each device must store *metadata* about all objects in the collection: this metadata includes storage system attributes describing which devices hold which objects, as well as application-specific attributes such as names, playlists, and types, enabling the user to identify and search for objects in the collection. The system should proactively pass information about updates to these objects between devices such that the user sees the same up-to-date collection on each device as soon as network connectivity permits.

The storage system does not need to keep the contents of all objects on each device, however. Instead, it should allow applications and users to set policies defining which objects should reside where, subject to network bandwidth and storage capacities.

object creation and manipulation:
$(objectID, versionID) \leftarrow$ create()
$(objectID, versionID)[] \leftarrow$ lookup($query$)
$versionID[] \leftarrow$ getVersions($objectID$)
$(key, value)[] \leftarrow$ getMetadata($objectID$, $versionID$)
$contentID \leftarrow$ open($objectID$, $versionID$)
$contents \leftarrow$ read($contentID$, $offset$, $length$)
$versionID \leftarrow$ newVersion($objectID$, $versionID[]$, $metadata$, $contents$)
$versionID \leftarrow$ deleteObject($objectID$)
placement rules:
$ruleID \leftarrow$ addRule($name$, $query$, $devices$, $priority$)
$(ruleID, query, devices, priority) \leftarrow$ getRule($name$)
$(ruleID, query, devices, priority)[] \leftarrow$ getAllRules()
removeRule($ruleID$)
event notifications:
$watchID \leftarrow$ addWatch($query$, $watchFlags$, $callback$)
removeWatch($watchID$)
$callback$($watchID$, $event$)

Figure 2: Device-transparent API summary. None of these calls block on network communication.

This distinction between object metadata and object content is a key mechanism that guides the design of a device-transparent storage system. Since every device knows about all objects, an application running on a given device can inform the user that an object has a recent and possibly conflicting update, even if that device hasn't yet obtained (or will never obtain) the object's content. Similarly, when an application needs an object whose content is not on the same device, it can use the metadata to help the user find a device that does have the content—even if the device storing the object is currently offline (e.g., a backup device sitting in a closet or safe deposit box)—so the user can bring the right device online if necessary and synchronize the devices.

Storing all metadata everywhere may have once been impractical, but portable devices' capabilities have increased rapidly. While this has led to a proportionate increase in the amount of data, such as high-fidelity music, photos, and video, it has not been matched by a corresponding increase in metadata. Applications use some base set of attributes they define and modify, which often only changes later in consequence to user actions.

Figure 1 illustrates this pattern by tracking the growth of a single user's photograph collection over a decade and three cameras. While the user gradually became a more prolific photographer, the increase in number of photos was outstripped by the technology-driven growth in the sizes of those photos. If the same types of applications manipulate each of the photos, and thus the amount of metadata is roughly equal for each photo, then the total fraction of storage space devoted to metadata decreases over time.

4. STORAGE API

Traditional file system APIs do not provide the facilities or semantics applications need to manage data objects in a device transparent model. Only applications understand how to present object names and collections to the user—most users do not know, or need to know, how their e-mail or media applications store files. Applications often must be involved in reconciling concurrent updates to the same object, since they can present or hide conflicts in an application-specific way.

Figure 2 shows a storage system API that provides applications with the necessary information to show users a device-transparent

collection. It is broken into three groups of operations: object creation and manipulation, placement rules, and event notifications. Each object has a single permanent unique identifier, and each update to an object has a version identifier, which allows the system to describe the recent history of an object even in the face of concurrent updates from multiple devices. If two or more current versions of an object exist, an application can resolve that conflict by creating a new version that specifies multiple predecessor versions. Any application can choose to ignore or resolve any conflict; objects are not tied to a particular application. Applications can attach arbitrary sets of tag/value pairs as metadata to an object: instead of a single pathname applications use logical queries against the tag/value metadata to find objects and manage collections of objects. Applications could also use this interface to present a traditional folder hierarchy.

Placement rules allow applications to indicate which objects to store on which devices. We do not expect users to write placement rules directly: instead, applications define placement rules on users' behalf. Users can group objects onto devices by common metadata attributes: e.g., songs go onto the media player. Alternatively, users can browse collections and specify particular objects and the devices on which they should be placed. The latter mode is important because previous work has shown that users often do not predict which objects they will need nor describe those collections with rules very well [23]. When mistakes prevent the user from accessing an object while disconnected, searching the metadata will indicate which devices currently have copies of the object. Placement rules contain a priority so that devices can gather important objects even when rules match more objects than fit in a device's local storage.

A device-transparent storage system must be able to synchronize devices over any topology: any two of a user's devices that can communicate must also be able to exchange updates and objects. After synchronizing metadata, devices transfer object contents in the background, without blocking the user's interaction with the data collection. Because updates can arrive at any time, the final component of the storage API is a watch mechanism that uses persistent queries to notify applications of updates whenever they arrive.

5. HOW MUCH METADATA?

A primary factor that will determine whether a device-transparent storage system is feasible in practice is the amount of metadata each device must hold, as this overhead occurs on every device in a user's group.

To explore the expected size of metadata collections in a device-transparent storage system, we extracted metadata for three modest personal data sets: the email, music, and photo collections a single user gathered over the past decade. Table 1 shows the resulting metadata sizes. To extract metadata, we parsed the email messages to extract useful headers, imported the user's media player attribute database, and used `exiftags` or `dcraw` to extract attributes from photos.

The most important feature this data set illustrates is that the size of the metadata associated with each object is roughly constant regardless of content type or size of each object. As such, the total size of the store is bounded by the number of objects, not the type of objects in the store. Consequently, collections of many small objects, such as email messages, represent the hardest use case for a device-transparent storage system.

Even if extra user and application attributes or identifiers and database indexes led to a metadata collection a factor of two or three larger than the sizes shown in the table, the total amount of metadata for this example (less than 600 MB) would be reasonable for today's current portable devices. The total content size (153 GB) would not fit on a laptop only a few years old, and not on many current portable devices. Adding video objects to the content collection would only increase the disparity between metadata and content store sizes, even if newer applications use more and larger metadata attributes.

email	
number of messages	724230
total content size	4.3 GB
median message size	4188 bytes
metadata size	169.3 MB
metadata/content overhead	3.8%
metadata size per message	245 bytes
music	
number of tracks/playlists	5278/21
total content size	26.0 GB
mean track size	5.1 MB
metadata size	2.6 MB
metadata/content overhead	0.01%
metadata size per object	511 bytes
photos	
number of JPEG/RAW objects	61740/10640
total number of objects	72380
JPEG/RAW content size	32.7/90.1 GB
total content size	122.8 GB
metadata size	22.6 MB
metadata/content overhead	0.02%
metadata size per object	328 bytes

Table 1: Metadata sizes for example datasets.

The communication requirements for keeping multiple devices up to date should not be an undue burden. Small mobile devices already handle frequent email updates over poor-quality network links, and the only extra burden in a device transparent storage system is passing those updates to each device. The bandwidth required to move content to the correct devices would be no more in a device-transparent storage system than it is today.

6. DISCUSSION

The design of a device-transparent storage system must address many points beyond the few areas that this paper considers. For example, we assume that an overlay network combines all of a user's devices into a single group to locate and manage network paths between those devices, and that a complete discussion of placement rules would describe how the storage manager on each device ensures that no objects get lost. Here are some areas for discussion along with brief comments.

What happens when metadata doesn't fit? Although warning strategies can try to limit cases where metadata takes up too much space, adding too many objects or including a device that is too small can cause a device to run out of space. At this point, the full device stops receiving updates from the rest of the group, but other devices continue unimpeded. The user can either delete individual items or get a new device with more storage space, which is exactly the same choice the user faces today. An alternative approach might be useful for small devices with limited interfaces (e.g., photo frames): a read-only class of devices could only hold

metadata for some object types, but consequently be unable to pass updates between devices.

What about version histories? Won't those grow without bound even if the size of an individual object doesn't? Multiple versions need to be kept only while there is a possibility of conflicts. Once all devices know about a given version of an object, everything strictly older can be truncated. Waiting to truncate versions until all devices agree whether there were any updates would cost extra metadata storage space if devices are absent or turned off for a long time. If this problem arises in practice, an active device can detect and warn the user of the state buildup, and provide the option to evict the absent device from the group temporarily or permanently.

How much work will modifying applications be? This process should be similar to the changes organizations are already making to support systems such as Live Mesh [16] and MobileMe [1], which include some support for disconnected and concurrent updates. Many applications already implement an attribute storage system atop standard filesystems, and watch for remote changes from other applications; a common attribute system will simplify these applications. Dealing with placement rules will be new for many applications, though a generic file browser should be able to handle placement rules for many types of objects rather than require that every single application do so.

Why should all devices need to know about all objects when devices can only use some types of data? Merely acting as a conduit to carry content and update notifications between other devices is a useful service. Also, identifying what data types a device can and can not use is no longer a simple problem; adding new applications to mobile devices on the fly is now common. Devices which are truly single purpose, rarely mobile, and hence not useful for update propagation, could instead act as read-only edge devices as described in the photo frame example above.

Do users really care about being able to find data on other devices that are turned off or inaccessible? Having access to application metadata for an object, even if the underlying object is inaccessible, still allows many useful operations. For example, a user can group photos into albums, add songs to playlists, find missing items, and move these collections between devices, all without needing access to the underlying objects.

Can this model work for multiple users? Sharing data collections is probably feasible for family sized groups, though not larger organizations, as that would break the assumption that the total amount of metadata is small enough for any single to device to hold. One way to design a shared collection would be to separate the storage collection into unshared and shared items and run a separate instance of the storage system over each.

Will handling conflicts be a problem for normal users? People should almost never see conflicts in a device-transparent storage system. Media objects and mail messages tend to be written once, and read many times afterwards. The job of the storage system is to track enough version information so that the only time a person sees a conflict is when that person made a truly incompatible change to the same attribute field from two different, disconnected devices. Any other type of change should be merged automatically by the applications.

7. RELATED WORK

The systems that come closest to providing device-transparency utilize optimistic replication to reach eventual consistency. All of the existing systems we are aware of, however, fall in groups to either side of our goal by not supporting partial replicas, or by not providing a complete view of the collection while disconnected.

Optimistic Replication Schemes:

Coda [13], Ficus [11], Ivy [17], and Pangaea [22] developed optimistic replication and consistency algorithms for file systems. Coda used a centralized set of servers with disconnected clients. Ficus and Ivy allow for updates between clients, but did not provide for partial replicas, and Pangaea handled disconnected servers, but not disconnected clients. A later extension to Ficus [21] added support for partial replicas, at the cost of no longer supporting arbitrary network topologies.

BlueFS [18] and EnsemBlue [19] extended Coda to permit a degree of decentralized updates along with more flexible placement rules and support for unmodified consumer devices, but retains a single central server.

Bayou [26] provided a device transparent view across multiple devices, but did not support partial replicas, and required all applications to provide merge procedures to resolve all conflicts.

PersonalRAID [24] tried to provide device transparency along with partial replicas. The approach taken, however, required users to move a single portable storage token physically between devices. Only one device can thus use the data collection at a given time.

TierStore [5], WinFS [15], and PRACTI [3] each support partial replicas, but limit the subsets to subtrees of a traditional hierarchical filesystems. TierStore targets Delay-Tolerant-Networking scenarios. WinFS aims to support large numbers of replicas. PRACTI also provided consistency guarantees between different objects in the collection. None of these systems provide device transparency over a complete collection.

Cimbiosys [20] and Perspective [23], come closest to the goals of device transparency; both support flexible placement rules over partial replicas. Neither attempts to provide a complete view over the data collection from disconnected devices or aid applications in reasoning about object version histories. Polygraph [14] extended Cimbiosys to recover from compromised devices.

Point to point synchronization:

Point-to-point synchronization protocols like rsync [28], tra [4], and Unison [2] provide on-demand and efficient replication of directory hierarchies. They do not, however, easily extend to a cluster of peer devices, handle partial replicas without extensive hand-written rules, or proactively pass updates whenever connectivity permits.

Version Control:

Version control systems such as Git [8] and Subversion [25] provide algorithms and models for reasoning about version histories, allowing developers to time-shift working sets back to arbitrary points.

Attribute Naming:

Storage system organization based on queries or attributes rather than strict hierarchical names have been studied in several single-device [7, 10] and multi-device [6] settings, in addition to the contemporary optimistic replication systems. More generally, the field of Personal Information Management [12] aims to organize data so

that users can find it easily, whereas device transparency aims to aid management of a data collection spread across multiple devices.

Centralized Topologies:
A number of newer systems provide tight application integration between multiple clients, such as MobileMe [1], Google Gears [9], and Live Mesh [16], but retain a centralized set of servers, so disconnected clients cannot share updates.

8. SUMMARY

Growing storage and network capabilities of mobile devices, combined with personal data collections that do not fit on all devices, leads to confusion caused by the object-on-a-device abstraction that traditional storage systems provide.

This paper describes a new abstraction, device transparency, that unifies the collections of objects on multiple devices into a single structure. Adopting the design principle of global metadata distribution would allow a storage system to provide the illusion of true device transparency in the face of intermittent communication and limited storage. We are currently working to implement and evaluate a personal storage system, *Eyo*, to examine these ideas in more detail.

9. ACKNOWLEDGMENTS

We would like to thank Ramesh Chandra, our shepherd Andrew Warfield, and the anonymous reviewers for their helpful comments. This research is sponsored by the T-Party Project, a joint research program between MIT and Quanta Computer Inc., Taiwan, and the NSF FIND program, with additional support from Nokia Research.

10. REFERENCES

[1] Apple Inc. MobileMe. http://www.apple.com/mobileme/.
[2] S. Balasubramanian and Benjamin C. Pierce. What is a file synchronizer? In *Proceedings of MobiCom '98*, October 1998.
[3] N. Belaramani, M. Dahlin, L. Gao, A. Nayate, A. Venkataramani, P. Yalagandula, and J. Zheng. PRACTI replication. In *Proceedings of NSDI'06*, 2006.
[4] R. Cox and W. Josephson. File Synchronization with Vector Time Pairs. Technical Report MIT-CSAIL-TR-2005-014, MIT, 2005.
[5] Michael Demmer, Bowei Du, and Eric Brewer. TierStore: A Distributed File-System for Challenged Networks. In *Proceedings of FAST*, 2008.
[6] Roxana Geambasu, Magdalena Balazinska, Steven D. Gribble, and Henry M. Levy. Homeviews: peer-to-peer middleware for personal data sharing applications. In *Proceedings of SIGMOD*, 2007.
[7] David K. Gifford, Pierre Jouvelot, Mark A. Sheldon, and Jr. James W. O'Toole. Semantic File Systems. In *Proceedings of SOSP*, 1991.
[8] Git. http://git.or.cz/.
[9] Google Gears. http://gears.google.com.
[10] Burra Gopal and Udi Manber. Integrating content-based access mechanisms with hierarchical file systems. In *Proceedings of OSDI*, 1999.
[11] Richard G. Guy, John S. Heidemann, Wai Mak, Thomas W. Page, Jr., Gerald J. Popek, and Dieter Rothmeir. Implementation of the Ficus Replicated File System. In *Proceedings of the USENIX Summer Conference*, June 1990.
[12] William Jones and Jaime Teevan, editors. *Personal Information Management*. University of Washington Press, Seattle, WA, 2007.
[13] James J. Kistler and M. Satyanarayanan. Disconnected Operation in the Coda File System. In *Proceedings of SOSP*, 1991.
[14] Prince Mahajan, Ramakrishna Kotla, Catherine Marshall, Venugopalan Ramasubramanian, Thomas Rodeheffer, Douglas Terry, and Ted Wobber. Effective and Efficient Compromise Recovery for Weakly Consistent Replication. In *Proceedings of EuroSys*, 2009.
[15] Dahlia Malkhi, Lev Novik, and Chris Purcell. P2P replica synchronization with vector sets. *SIGOPS Oper. Syst. Rev.*, 41(2):68–74, 2007.
[16] Microsoft. Live Mesh. http://www.livemesh.com.
[17] A. Muthitacharoen, R. Morris, T. Gil, and B. Chen. Ivy: A Read/Write Peer-to-peer File System. In *Proceedings of OSDI*, 2002.
[18] Edmund B. Nightingale and Jason Flinn. Energy-efficiency and storage flexibility in the blue file system. In *Proceedings of OSDI*, 2004.
[19] Daniel Peek and Jason Flinn. EnsemBlue: Integrating Distributed Storage and Consumer Electronics. In *Proceedings of OSDI*, 2006.
[20] V. Ramasubramanian, T. Rodeheffer, D. Terry, M. Walraed-Sullivan, T. Wobber, C. Marshall, and A. Vahdat. Cimbiosys: A Platform for Content-based Partial Replication. In *Proceedings of NSDI*, 2009.
[21] David Ratner, Peter L. Reiher, Gerald J. Popek, and Richard G. Guy. Peer Replication with Selective Control. In *Proceedings of the First International Conference on Mobile Data Access*, 1999.
[22] Yasushi Saito, Christos Karamanolis, Magnus Karlsson, and Mallik Mahalingam. Taming aggressive replication in the Pangaea wide-area file system. In *Proceedings of OSDI*, 2002.
[23] Brandon Salmon, Steven W. Schlosser, Lorrie Faith Cranor, and Gregory R. Ganger. Perspective: Semantic Data Management for the Home. In *Proceedings of FAST '09*, San Francisco, CA, February 2009.
[24] Sumeet Sobti, Nitin Garg, Chi Zhang, Xiang Yu, Arvind Krishnamurthy, and Randolph Y. Wang. PersonalRAID: Mobile Storage for Distributed and Disconnected Computers. In *Proceedings of FAST*, 2002.
[25] Subversion. http://subversion.tigris.org.
[26] Douglas B. Terry, Marvin M. Theimer, Karin Petersen, and Alan J. Demers. Managing Update Conflicts in Bayou, a Weakly Connected Replicated Storage System. In *Proceedings of SOSP*, 1995.
[27] Irving L. Traiger, Jim Gray, Cesare A. Galtieri, and Bruce G. Lindsay. Transactions and consistency in distributed database systems. *ACM Trans. Database Syst.*, 7(3):323–342, 1982.
[28] Andrew Tridgell. *Efficient Algorithms for Sorting and Synchronization*. PhD thesis, Australian National University, April 2000.