Putting Home Data Management into Perspective

BRANDON WATTS SALMON

December 2009
CMU–PDL–09–113

Dept. of Electrical and Computer Engineering
Carnegie Mellon University
Pittsburgh, PA 15213

Submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

Thesis committee

Prof. Gregory R. Ganger, Chair (Carnegie Mellon University)
Prof. Lorrie Faith Cranor (Carnegie Mellon University)
Prof. Anind K. Dey (Carnegie Mellon University)
Prof. Mahadev Satyanarayanan (Carnegie Mellon University)
Dr. Steven W. Schlosser (Avere Systems)

For Mary. I love you.

Abstract

Distributed storage is coming home. An increasing number of home and personal electronic devices create, use, and display digitized forms of music, images, videos, as well as more conventional files (e.g., financial records and contact lists). In-home networks enable these devices to communicate, and a variety of device-specific and datatype-specific tools are emerging. The transition to digital homes gives exciting new capabilities to users, but it also makes them responsible for administration tasks usually handled by dedicated professionals in other settings. It is unclear that traditional data management practices will work for "normal people" reluctant to put time into administration.

This dissertation presents a number of studies of the way home users deal with their storage. One intriguing finding of these studies is that home users rarely organize and access their data via traditional folder-based naming—usually, they do so based on data attributes. Computing researchers have long talked about attribute-based data navigation, while continuing to use folder-based approaches. However, users of home and personal storage live it. Popular interfaces (e.g., iTunes, iPhoto, and even drop-down lists of recently-opened Word documents) allow users to navigate file collections via attributes like publisher-provided metadata, extracted keywords, and date/time. In contrast, the abstractions provided by filesystems and associated tools for managing files have remained tightly tied to namespaces built on folders.

To correct the disconnect between semantic data access and folder-based replica management, this dissertation presents a new primitive that I call a

"view", as a replacement for the traditional volume abstraction. A *view* is a compact description of a set of files, expressed much like a search query, and a device on which that data should be stored. For example, one view might be *"all files with type=music and artist=Beatles stored on Liz's iPod"* and another *"all files with owner=Liz stored on Liz's laptop"*. Each device participating in a view-based filesystem maintains and publishes one or more views to describe the files that it stores. A view-based filesystem ensures that any file that matches a view will eventually be stored on the device named in the view. Since views describe sets of files using the same attribute-based style as users' other tools, view-based management replica management should be easier than folder-based file management.

In this dissertation I present the design of Perspective, a view-based filesystem, and Insight, a set of view-based management tools. User studies, deployments and benchmarks using these prototypes show that view-based management simplifies some important tasks for non-technical users and can be supported efficiently by a distributed filesystem.

Acknowlegements

I would especially like to thank Greg, my advisor, for believing in me. He allowed me to drive into an area which was new, not only to our group, but also to the field of systems researchers. It took a lot of faith and patience on his part to allow me the time and flexibility to do so. I cannot thank my parents, Linton and Lisa Salmon, enough. They believed in me when I did not believe in myself.

I would also thank all the members of the Perspective development team: Lujo Bauer, Hardik Doshi, Jared Goerner, Nitin Gupta, Rohit Harchandani, Christina Johns, Michelle Mazurek, Prerak Mehta, Abdur Pathan, Steve Schlosser, Zoheb Shivani and Eric Toan. It has been a pleasure working with you guys, and I look forward to great things from you in the future.

As a system person moving into the world of usability studies, I have benefitted greatly from the tutilage of a variety of usability experts. I would especially like to thank Jay Hasbrouck and Jay Melican, who worked with me at Intel and beyond, Rob Reeder and Kami Vaniea, who helped me through the details of my first lab study, and my faculty advisors Lorrie Cranor and Anind Dey, who have helped me craft the studies in this dissertation.

Thanks are also in order for the many participants of the user studies in this dissertation; I've learned a lot from you. I'd also like to thank the writers, actors and directors of Lost, Life, Psych, The Mentalist, and Better off Ted for making hours of DVR development much, much more enjoyable.

I would also like to thank the members and companies of the PDL Consortium (including APC, Cisco, DataDomain, EMC, Facebook, Google, HP, Hitachi, IBM, Intel, LSI, Microsoft, NEC, NetApp, Oracle, Seagate, Sun,

Symantec, and VMware) for their interest, insights, feedback, and support. This material is based on research sponsored in part by the National Science Foundation, via grants #CNS-0326453 and #CNS-0831407, and by the Army Research Office, under agreement number DAAD19–02–1–0389. I have also been supported by an NSF Fellowship and an Intel Fellowship during my studies.

Contents

Figures

Tables

1 Introduction

Distributed storage is coming home. An increasing number of home and personal electronic devices create, use, and display digitized forms of music, images, videos, as well as more conventional files (e.g., financial records and contact lists). In-home networks enable these devices to communicate, and a variety of device-specific and datatype-specific tools are emerging. The transition to digital homes gives exciting new capabilities to users, but it also makes them responsible for administration tasks usually handled by dedicated professionals in other settings. It is unclear that traditional data management practices will work for "normal people" reluctant to put time into administration.

An intriguing aspect of home storage is that home users rarely organize and access their data via traditional folder-based naming—usually, they do so based on data attributes. Computing researchers have long talked about attribute-based data navigation (e.g., semantic filesystems [22, 72]), while continuing to use folder-based approaches. However, users of home and personal storage live it. Popular interfaces (e.g., iTunes, iPhoto, and even drop-down lists of recently-opened Word documents) allow users to navigate file collections via attributes like publisher-provided metadata, extracted keywords, and date/time. Usually, files are still stored in underlying folders in the file system, but users often are insulated from naming at that level and are oblivious to where in the namespace given files end up.

Users have readily adopted these higher-level navigation interfaces, leading to a proliferation of semantic data location tools [82, 6, 23, 73, 38]. In contrast, the abstractions provided by filesystems and associated tools for

managing files have remained tightly tied to namespaces built on folders. For example, most tools require that specific subtrees be identified, by name or by "volumes" containing them, in order to perform *replica management tasks*, such as partitioning data across computers for capacity management or specifying that multiple copies of certain data be kept for reliability. Since home users double as their own system administrators, this disconnect between interface styles (semantic for data access activities and folders for management tasks) naturally creates difficulties. This dissertation presents *views* as an abstraction to mitigate this disconnect.

1.1 Thesis statement

The view abstraction, which characterizes files using attributes rather than folders, simplifies replica management tasks in the unique area of home storage. The view abstraction can be efficiently implemented by augmenting and adapting distributed filesystem mechanisms.

1.2 Views

To correct the disconnect between semantic data access and folder-based replica management, I propose replacing the traditional volume abstraction with a new primitive that I call a view. A *view* is a compact description of a set of files, expressed much like a search query, and a device on which that data should be stored. For example, one view might be *"all files with type=music and artist=Beatles stored on Liz's iPod"* and another *"all files with owner=Liz stored on Liz's laptop"*. Each device participating in a view-based filesystem maintains and publishes one or more views to describe the files that it stores. A view-based filesystem ensures that any file which matches a view will eventually be stored on the device named in the view.

Since views describe sets of files using the same attribute-based style as users' other tools, view-based management replica management should be easier than folder-based file management. A user can see what is stored where, in a human-readable fashion, by examining the set of views in the

system. She can control replication and data placement by changing the views of one or more devices. Views allow sets of files to overlap and to be described independently of namespace structure, removing the need for users to worry about application-internal file naming decisions or unfortunate volume boundaries. Semantic management can also be useful for local management tasks, such as setting access control and other policies on files, in addition to replica management.

The view abstraction can also be used as an organizational structure for distributed filesystems to allow devices to share storage and provide fault-tolerance and mobility guarantees without requiring a central server. Each device holds a subset of the data and can access data stored on any other (currently connected) device.

1.3 Methodology

I demonstrate the advantages and feasibility of views in several ways. Since this dissertation is focused on providing manageable home storage, my evaluation is primarily focused on user studies of management and usability.

First, I performed an initial *contextual analysis* and test system deployment to increase our understanding of the unique needs of the home environment and the way that distributed filesystems might affect this environment. I describe observations from these initial explorations and how they support a view-based approach to replica-management.

Second, I built Perspective[1], a prototype view-based filesystem that runs on Linux and OS X, to show the feasibility of implementing such a system. and provide a platform for deployment studies. In deployments, Perspective provides normal file storage as well as being the backing store for iTunes and MythTV in several households, in addition to our research lounge.

Third, I performed users studies to evaluate the impact of view-based management. I created a user interface for replica management and placed this interface on top of a view-based system and two more conventional approaches. In order to minimize the effects of the interface itself, I kept the

[1] In seeing many views, one gains Perspective.

interface as similar as possible, while varying the architecture underneath it. I then asked non-technical users to perform a set of sample management tasks in the lab using each interface. Results showed up to a 6x improvement in the number of users able to correctly complete these tasks.

Fourth, I present performance experiments with the Perspective prototype. While performance is not the focus of this dissertation, these experiments confirm that Perspective can provide consistent, decentralized storage with reasonable performance. Even with its application-level implementation (connected to the OS via FUSE [20]), Perspective performance is within 3% of native filesystem performance for activities of interest.

Fifth, I have deployed Perspective into the homes of several lab members, and I present an initial exploration into the advantages and challenges of a view-based system in actual long-term usage.

1.4 Contributions

This dissertation contains a number of contributions to the literature. First, I present two exploratory studies into home storage. While the literature contains a rich history of home studies, my studies each have unique focus. The deployment study is unique in focusing on the way non-technical users react to a distributed filesystem. The contextual analysis of home users is unique in its focus on device upgrade and reliability of home data.

Second, I present the first semantic abstraction for replica management, the view, and a user study evaluating the usability impact of semantic management in contrast with more conventional approaches.

Third, I present a group of algorithms needed to provide the view abstraction in a distributed filesystem. These algorithms include the first consistency protocol that allows for semantically-defined partial replication in an eventually consistent, heterogeneous, decentralized filesystem environment. These algorithms also include methods to provide efficient search in a view-based system.

Fourth, I present Perspective, the first filesystem to provide view-based replica management. Perspective is fully functional and is in use in several

household in the Pittsburgh area. In addition to showing the feasibility of a view-based filesystem, the Perspective prototype provides a platform for continuing research into distributed semantic storage, and distributed home storage.

Fifth, I present a group of tools which allow users to manipulate file replicas using the view abstraction. I introduce *overlap trees* as a mechanism for efficiently reasoning about how many replicas exist of a particular dataset, and where these files are stored, even when no view exactly matches the attributes of the dataset. I present the *view manager interface* as an interface for semantic replica management. I also present *customizable faceted metadata* as a way to browse semantic filesystem data.

Sixth, I present results from an initial deployment of the Perspective filesystem. These results explore some of the initial advantages and challenges of view-based management in practice.

1.5 Roadmap

Chapter 2 presents related work. Chapter 3 presents a deployment study of a home server system. Chapter 4 presents a contextual analysis of home data management. Chapter 5 presents the design points gleaned from these studies and a view-based architecture based on these studies. Chapter 6 presents Perspective, our view-based filesystem prototype. Chapter 7 presents Insight, a set of prototype view-based tools. Chapter 8 presents evaluations of the usability of the views-abstraction, the performance of Perspective, and initial findings from a deployment of Perspective. Chapter 9 presents conclusions and future work. The appendices present details from several of the user studies.

2 Background and related work

In this section I outline the wealth of previous work on which this dissertation is built. First I outline work calling for or using system-focused user-oriented design. Second, I describe studies that utilized similar methods or investiaged similar topics as the user studies in this dissertation. Third I describe distributed filesystems that have similar design to that of Perspective. Fourth, I describe semantic filesystems and semantic interfaces that relate to the Insight toolset.

2.1 Usable system design methodology

This dissertation presents an example of the application of design techniques from HCI to the systems challenge of filesystem design. While the fields of system design and HCI have developed different design and evaluation methodologies for good reasons, there are many problems which require a combination of such techniques. The home is one such environment, where a challenging environment requires sophisticated, and efficient distributed algorithms, but management by novice users also requires careful attention to usability.

In such environments, it is not enough to design an interface without a knowledge of how the system must be built or to design a system without an understanding of the user needs and workflows. Instead, usable system design problems require a system design influenced by user needs and workflows.

This thesis presents a combined methodology and its application to distributed home storage. I first performed exploratory studies of how users

interacted with distributed filesystems, then performed a targeted contextual analysis to study the home constraints. I used the information from these studies to design the Perspective filesystem. Finally, I tested the system using a lab usability study and an actual filesystem deployment, studied using regular in-situ semi-structured interviews.

While this approach is uncommon in the systems community, the HCI community has a long history of using these kinds of design methodologies [44, 31, 19]. The novelty is not the specific methodologies themselves, but the application to specific system problems in filesystem design.

There has also been related work on architecting systems for usability. The most closely related is by Ebling et al. [15], who performed a user study showing how extra visability into the Coda caching scheme improved accuracy of technical users on critical tasks. John et al. [35] argue that architectures are critical for usability and present a set of example scenarios to support this claim.

2.2 Home storage studies

Another contribution of this dissertation is the information gained from the specific user studies I performed. While I am not aware of previous research that examines users' behavioral responses and adjustments to shared home data systems in the same manner as the studies in this dissertation, a number of studies have employed similar techniques or investigated closely related issues.

2.2.1 Contextual analysis

A *contextual analysis* or *contextual inquiry* is a set of in-situ, semi-structured interviews with users to explore the way in which they think particular tasks or technologies [8]. This approach allows interviewers to learn about the problems users find difficult and the way in which they go about problems in a space. There is a rich history of contextual analysis and similar techniques in home technologies [44, 31].

Grinter et al. [26] studied households with advanced networking setups in the home, exploring the management of the physical networks (e.g., wireless routers) installed by the owners themselves. They noted that data sharing between devices was a problem that households found vexing and did not know how to address. Our findings concur with many of the observations in the Grinter study.

Brush et al. [12] conducted a study of how home users share technology devices in the home. Our studies support and enhance many of the results from this study, including challenges in dealing with logins, the lack of password usage, and the specialization of technological devices.

A number of studies have also studied how home technologies interact with home social norms and utilization of locations within the home [18, 19]. Our studies have many similar findings, but also provide a counterpoint. Like Brush et al. [12], we found that the number of home storage devices had dramatically increased, leading to less contention for usage of each resource.

Marshall has performed a large number of contextually analysis investigations into the way users manage their data over long-term periods, an area she calls *personal archiving* [41, 42, 43, 39]. Many of the observations from our contextual analysis support findings in these studies, including the frequency of data death due to device upgrade or failure. Marshall has also studied how tagging schemes differ from other metadata on Flickr [40].

A collection of studies have analyzed the filesystem behavior of enterprise settings. Oulasvirta et al. [49] studied how users switch between different devices at work, and noted that users found many reasons to use a range of devices at work, and observed a variety of different methods for handling device swapping. While not focused specifically on either home or business, several studies have also explored the rich social implications of music sharing in distributed environments [78].

There have also been a variety of related studies of portable technologies [16, 52, 66]. For example, Aipperspach et al. studied the locations where notebook computers are used in the home, using a similar combination of tracing and trace-prompted interview techniques, noting stable patterns of usage [3]. O'Hara et al. [30] studied the social implications of portable video,

following up a rich set of prior work on TV usage [37, 11]. This study reinforced the fact that even within the home, users may employ multiple devices to access the same content.

A variety of researchers have also studied home automation technologies and their impact on the family [75, 7]. For example, Woodruff et al. described how home automation systems in Orthodox Jewish households play an important role in family worship [83].

The home data management contextual analysis described in Section 4 borrows methodologies from these studies and touches on many themes explored in these studies. However, it is unique in focusing on users' reactions to device upgrade, device failure, and data preservation.

2.2.2 Deployment

A second approach is to deploy a new technology into users' homes and to study the impact this technology via interviews like those of contextual analysis. I performed two of these studies, one using a home server and one using Perspective. The HCI literature also has a rich history of these kinds of deployments.

O'Brien et al. described the impacts of a new desktop box on a set of households [48]. They found that users were not necessarily happy with having expanded functionality in a single device, and explored many of the social issues involved. Our study kept the devices separate, but allowed for data sharing between them. Still, many of our findings reinforce the social factors mentioned in the O'Brien study.

Another deployment study built a "smart home" prototype that included control of things like locks and windows, asked several families to live in it, and studied the results [60]. Many of the tensions found in this study are present in our studies as well. For example, families and system designers struggled to balance the need for security and customization with ease of usage.

A similar deployment of a home storage system was performed at Intel by Hady et al. [29], in which I was initially involved. They deployed a sample

Union-FS based approach into consumer homes and published a white-paper on the anecdotal results. As with the home server study in this thesis, they found that immediate data access was greatly liked by end users.

This dissertation describes two deployment studies. They are both unique in that they study the social impact of the deployment of a distributed filesystem in study households. In the home server study described in Section 3, I deployed an off-the-shelf distributed filesystem in non-technical households. Ian the Perspective deployment described in 8.2, I deployed the Perspective semantic distributed filesystem into technical households.

2.3 Distributed filesystems

Another contribution of this dissertation is the Perspective filesystem itself. A primary contribution of Perspective is the use of semantic queries to *manage the replication of data*. Specifically, Perspective provides accessibility and reliability guarantees over semantic, partially replicated data. This builds on previous semantic systems that used queries to *locate* data and folders to manage data. Our user study evaluation shows that, by supporting semantic management, Perspective can simplify important management tasks for end users.

Another contribution is a filesystem design based on in-situ analysis of the home environment. This overall design could be implemented on top of a variety of underlying filesystem implementations, but we believe that a fully view-based system provides simplicity to both user and designer by keeping the primitives similar throughout the system. While no current system provides all of the features of Perspective, Perspective builds on a wealth of previous work in data placement, consistency, search and publish/subscribe event notification. In this section, I discuss this related work.

2.3.1 Data placement

Views allow flexible data placement used to provide both reliability and mobility. Views are another step in a long progression of increasingly flexible

data placement schemes.

The most basic approach to storing data in the home is to put all of the data on a single server and make all other devices in the home act as clients of this server. Variations of this approach centralize control, while allowing data to be cached on devices [36, 71].

To provide better reliability, AFS [69], LOCUS [79], and Deceit [70] expanded the single server model to include a tier of replicated servers, each connected in a peer-to-peer fashion. Each server stores a copy of some number of *volumes*, which server as the first element of the file path. However, clients cannot access data when they are out of contact with the servers. Coda [67] addressed this problem by allowing devices to enter a disconnected mode, in which devices use locally cached data defined by user hoarding priorities. However, hoarded replicas do not provide the reliability guarantees allowed by volumes because devices make no guarantee about what data resides on what devices, or how long they will keep the data they currently store. Views extend this notion by allowing volume-style reliability guarantees along with the flexibility of hoarding in the same abstraction.

A few filesystems suggested even more flexible methods of organizing data. BlueFS [47] extended the hoarding primitive to allow client devices to access data hoarded on portable storage devices, in addition to the local device, but did not explore the use of this primitive for accessibility or reliability beyond that provided by Coda. Footloose [50] proposed allowing individual devices to register for data types in this kind of system as an alternative to hoarding files, but did not expand it to general publish/subscribe-style queries or explore how to use this primitive for mobility, reliability, management or distributed search.

2.3.2 Consistency

Perspective supports decentralized, topology-independent consistency for semantically-defined, partially replicated data, a critical feature for the home environment. While no previous system provides these properties out of the box, PRACTI [14] also provides a framework for topology-independent con-

sistency of partially replicated data over directories, in addition to allowing a group of sophisticated consistency guarantees. PRACTI could probably be extended to use semantic groupings fairly simply and, thus, provide consistency properties like Perspective. Recently, Cimbiosis [59] has also built on a view-style system of partial replication and topology independence, with a different consistency model.

Cimbiosis also presents a *sync tree*, which provides a distributed algorithm to ensure connectedness and routes updates in a more flexible manner. This sync tree could be layered on top of the consistency model of Perspective or PRACTI to provide these advantages.

I chose the Perspective approach over Cimbiosis, because it does not require any device to store all files, while Cimbiosis has this requirement. Many of the households in our contextual analysis did not have any such master device, and appeared unwilling to spend the money required to have such a device, leading us to believe that requiring it could be a problem. Perspective also does not require small devices to track any information about the data stored on other devices, while PRACTI requires them to store imprecise summaries. However, there are advantages to each of these approaches as well. For example, PRACTI provides a more flexible consistency model than Perspective, and Cimbiosis a more compact log structure. A full comparison of the differences between these approaches, and the relative importance of these differences, is not the focus of this dissertation. I present Perspective's algorithms to show that it is possible to build a simple, efficient consistency protocol for a view-based system. However, section 8.4.2 outlines a qualitative analysis of these differences.

Previous peer-to-peer systems such as Bayou [76], FICUS [27] and Pangaea [64] extended synchronization and consistency algorithms to accommodate mobile devices, allowing these systems to blur or eliminate the distinction between server and client [58, 54]. However, none of these systems fully support topology-independent consistency with partial replication. EnsemBlue [52] takes a middle ground, providing support for groups of client devices to form device ensembles [68], which can share data separately from a server through the creation of a temporary pseudo-server, but requiring a

central server for consistency and reliability.

Podbase [57] extends an eventual consistency model by automatically inferring that identical files are replicas of one another, an approach which could be useful on top of a number of eventual consistency systems.

2.3.3 Search

Perspective uses views to provide efficient distributed search, by guiding searches to appropriate devices. The most similar work is HomeViews [21], which uses a primitive similar to Perspective's views to allow users to share read-only data. HomeViews combines capabilities with persistent queries to provide an extended version of search over data, but it does not use them to target replica management tasks like reliability. View-based search also builds on previous work like Diamond [32, 63], which pushes search into storage devices, but does not attempt to selectively choose which devices to search. A full performance comparison is highly dependent upon the workload and data layout, but section **??** contains a qualitative analysis of the advantages and drawbacks of view-based search.

2.3.4 Replica indices and publish/subscribe

In order to provide replica coherence and remote data access, filesystems need a replica indexing system that forwards updates to the correct file replicas and locates the replicas of a given file when it is accessed remotely. Previous systems have used volumes to index replicas [67, 69], but did not support replica indexing in a partially replicated peer-ensemble. EnsemBlue [52] extended the volume model to support partially replicated peer-ensembles by allowing devices to store a single copy of all replica locations onto a temporarily elected *pseudo-server* device. EnsemBlue also showed how its replica indexing system could be leveraged to provide more general application-level event notification. Perspective takes an inverse approach. It uses a publish/subscribe model to implement replica indexing and, thus, application-level event notification. This matches the semantic nature of views.

This work does not propose algorithms beyond the current publish/subscribe literature [2, 9, 13, 55, 74]. Instead, it applies publish/subscribe algorithms to the new area of file system replica indices. Using a publish/subscribe method for replica indexing provides advantages over a pseudo-server scheme, such as efficient ensemble creation, but also disadvantages, such as requiring view changes to move replicas. Again, a full comparison of alternative approaches is not the focus of this dissertation. I present Perspective's algorithms to show that replica indexing can be performed efficiently using views. Section 8.4.4 provides a qualitative comparison between various replica indexing schemes.

2.4 Semantic interfaces

I believe that effective home data management will use search on data attributes to allow flexible access to data across heterogeneous devices. Perspective takes the naming techniques of semantic systems and applies them to the replica management tasks of mobility and reliability as well.

2.4.1 Attribute-based naming

The term *semantic* has been applied to a wide range of differing technologies. To be more specific, I will use the term *attribute-based naming* to describe the naming methodologies Perspective espouses. These techniques assign some number of *attributes* or *tags* to an item and, then, allow the user to search for all items with a matching tag. The format of these tags may differ from system to system. For example, some systems use single-value tags, while others use name-value pairs. Some tag names are flat, while others are hierarchical.

This is in contrast to the traditional *files-and-folders* filesystem naming methodology, which places each file in a folder, which can then be placed into another folder, etc. In this system, each file has a *path* that uniquely identifies it in the system. The user can browse or set policies on folders and all their subfolders.

A huge percentage of applications for the home use attribute-based naming. These include photo applications [33, 53, 4], music players [34, 45], online data sharing sites [1, 17], email, and even document editing [24]. Operating systems themselves are also bypassing traditional files-and-folders organizations by using recently-accessed document lists, search tools, and semantic folders.

Despite this plethora of attribute-based tools for locating and organizing data, we are unaware of management tools (file replication, backup, copying, etc.) that utilize attribute-based groupings. Instead these interfaces are tightly coupled to files-and-folders. We adapted the Expandable Grids toolkit [62], first developed for managing permissions in a files-and-folder system, to build our interface for view manipulation.

While the lab study in section **??** of this dissertation shows that some tasks are harder to perform in a files-and-folders naming than attribute-based naming, the most important challenge the study illustrates is the mismatch between the type of naming used by applications and that used by management tools. When asked to use an attribute-based application (and most home applications are attribute-based), users found it very difficult to manage that same data using a files-and-folders based management tool. As user applications have become more attribute-based, not less, over time, it is important for management tools and structures to allow users to manage their data with attribute-based naming as well. Unfortunately, the two techniques are difficult to mix.

2.4.2 Comparing attributes and folders

Attribute-base naming schemes and files-and-folders have several fundamental differences that make it difficult to convert from one naming scheme into the other. While there are certainly cases where a conversion is easily possible, there are many important cases where it is not.

Attributes allow for multiple groupings: Attributes are specifically built to allow for multiple ways of grouping the same set of files, while folders are not. A file may show up in many different searches, but only a single

directory path. While the single-path-single-file approach can be helpful in programmatic access to data, it makes it difficult to be flexible in how data is accessed and found by an end user. For example, a photo may be part of a roll of photos imported at once from a camera, several photo albums, photos taken at a specific location, and photos containing various people.

Attributes are easily hideable: Attribute-based systems allow users and the system to add large numbers of attributes to files while only requiring end users to interact with the attributes they find valuable. While a file may have many attributes associated with it, only those attributes that interact with the current search are exposed to the user. In contrast, there is no easy way to contract a file path in a folder-based system; all the elements of the path must be considered when accessing a file. This forces users and systems to be careful to only add attributes that are known to be valuable, since extra attributes add complexity to all operations on the file.

This makes attributes a better fit for the metadata-rich data in the home. For example, audio files come with a large number of attributes such as album, artist, record label, etc. These attributes can be very valuable, but many of them are not utilized by most users in most queries. Thus, most folder-based systems only put a small number of these attributes into the folder structure, making the additional attributes unusable in organizing the data from the filesystem.

Attributes are easily composable: A search can filter based on any number of combinations of attributes. In contrast, in a folder-based scheme only attributes in subtrees are composable. Even if a file has multiple hard-links there is no way to compose the elements contained in them. This can be especially important when the user is unsure exactly how tags will be used when they initially associate the tags with the file. For example, when creating a photo a user is likely to know the outing on which the photo was taken, the people in the photo, and the time the photo was taken, but is unlikely to know whether she will want to organize the photos by year and person, outing and year, year and person, etc. Attributes allow her to decide how to organize the photos later and, indeed, to switch from one structure to another when needed.

Attributes are easily transposable: It is simple to sort a group of files by either genre or artist, for example. In contrast, folder-based systems have an inherent ordering in the path elements that cannot be undone. So, for example, if a folder tree is organized by genre and then by artist, it is difficult to sort the files by artist, even if the information is contained in the tree. This is especially critical when different tasks require different first-level groupings. For example, many current trees are divided by user followed by file type. This may be perfect for backing up all of one user's data, but a poor fit for a home music player, which would like to see all Music, regardless of the owner.

Attributes are bound to files: Attribute-based systems assume that tags belong to a given file. They are edited, created and destroyed on files, copied with files, etc. In contrast, files-and-folder systems assume that names are part of the folder, not the file itself. If the file is copied it's path does not follow it, and there is no way to edit other paths that may point to the same file.

Links: Hard links and symbolic links allow folder-based approaches to give a file multiple names in multiple folders. This extends a files-and-folders based system to include multiple groupings and name hiding. However, it still does not provide composition, transposition or file-binding. Following are example limitations of links that make them incompatible with an attribute-based system.

- Filesystems provide no way to list and edit all links to a given file. Each link is only accessible from the directory in which it resides. This prevents file-binding.

- Both hard and symbolic links provide hazy control over deletion of a file separate from removing and adding links. For symbolic links, removals on all paths but the "real" path are link removal, while removal on the "real" path is a deletion. For hard links, removing all links is a deletion, but removing any individual link is a link removal. This is further complicated by the lack of a good interface to determine what

other links exist to the file. This again is due to the lack of file-binding of names.

– Links provide no easy way to compose link paths (for example, the same file may have the name /Music/foo.mp3 and the name /Brandon/foo.mp3, but there is no way to find all files that are in both the /Music and /Brandon folders). This prevents composition and transposition.

These omissions make links difficult to manage, as evidenced by their limited usage in most filesystems. For example, Windows does not provide hard links through any standard interface, the Linux finder and terminal provide limited support, and almost no applications use links for items they expect to be manipulated by end users.

That said, links could be used as the basis of an attribute-style system if they were extended to provide the above features. Once added, a link style system would be indistinguishable from an attribute-based system with hierarchical tags, with a particular viewing methodology. There may be value in this implementation of tags for efficiency and/or initial adoption, although it is not immediately clear how exactly to extend links to provide these features.

2.4.3 Perspective's attribute-based naming

Perspective uses name-value pair attributes, because I believe the extra structure is important when using queries for file placement and security policies, which are less tolerant to false positives than search. Tags in Perspective are flat, with no hierarchy. It would not be difficult to augment Perspective to use hierarchical tags, but it is unclear how to modify faceted metadata, which Perspective uses for browsing and setting policy, to accommodate hierarchical tags.

2.4.4 Faceted metadata

Perspective uses *faceted metadata* to access and modify file metadata. In a faceted-metadata approach, the system allows the user to first choose an attribute of interest from a list of attributes. It then displays all the values for that attribute. After selecting a particular value the user is then allowed to either view all matching files or choose another attribute to use to refine their search. This technique has a long history of related work.

The Semantic Filesystem [22] proposed the use of attributes to locate data in a file system, and proposed a faceted-metadata style approach. Subsequent systems showed how these techniques could be extended to include personalization of the search queries [25].

Flamenco [84] uses faceted metadata to browse documents in a static corpus. Microsoft's proposed WinFS filesystem also incorporated semantic naming [82]. Other researchers have explored ways to expose a semantic structure in a method similar to folders [80]. Microsoft's Windows 7 provides *libraries* that allow the user to sort their data using different attributes, in a stripped down version of faceted metadata [81].

Perspective uses a novel form of faceted metadata called *customizable faceted metadata*, which helps tune what attributes are displayed to the user when a large number of attributes exist in the filesystem. Customizable faceted metadata is described in Section 7.1.1.

2.4.5 Keyword search

Other systems provide search functionality to files-and-folder systems [6, 23, 38, 73]. While all of these systems use name-value pairs internally, the interface they expose is *keyword search*. In keyword search the user simply types a word and the interface displays all files with tag values that contain the keyword. While keyword search has more false positives than strict tag comparisons, ranking algorithms can reduce the problem for the end user. Keyword search is very familiar to users from the web, and is powerful in its simplicity. However, these approaches are specifically tuned to providing search on automatically-extracted file attributes. Unlike faceted metadata

approaches, they do not provide easy interfaces for users to browse files via tags, modify tags, or create their own tags. Keyword search can be implemented using Perspective's name structures, although the currently implementation is not tuned for this kind of usage. However, we believe that the extra structure provided by faceted metadata is important not only for browsing and modifying tags, but also when using queries for file placement and security policies, which are less tolerant to false positives than search.

3 Home server deployment study

While there is a history in systems literature of deploying filesystems in technical labs and groups, there are no studies of which I am aware that explore non-technical users' response to these kinds of distributed systems. My studies have confirmed that non-technical users often have very different reactions to distributed filesystems than highly technical users.

In fall 2006 I deployed a file sharing system using off-the-shelf components into two non-technical households. This provided me an opportunity to study non-technical users reactions to this system.

This chapter outlines the methodology of this study and the findings. [65] These findings can be grouped into three main observations. First, immediate data access had a large and positive impact on users' usage of their data. Second, the devices, users, and management styles of these household were dynamic and surprisingly complex compared to the number of devices involved. Third, privacy was important to home users, but their notions of privacy and security did not match well with traditional filesystem access controls.

3.1 Methodology

To explore the effects of shared data on a household, I recruited two households from the Portland, Oregon area, installed a home data-sharing system in their homes, and observed them over two and a half weeks. My analysis methods consisted of both system-level tracing and ethnographically-inspired interviews. While the short length of this study limits my ability

to generalize all findings, I was able to observe several high-level challenges involved in adopting such a system.

I installed the system in each household and provided technical support as needed throughout the study. At the end of the study, I removed the technology and returned things to their pre-study state. Each household was compensated monetarily for their participation.

3.1.1 System description

The system that I installed used a single Windows machine with Windows file sharing as the data server. I also introduced two Linux-based digital video recorders (DVRs) into the house, which I connected to existing TVs. I also added a laptop-based stereo system running iTunes. In addition, I pointed the "My Documents" folder of all of the household computers into the data server.

Each device in the household stored all of its data on the single server machine over Windows file sharing. This allowed all devices to access any data stored in the system. The following list explains the directory structure on the server and device targeting.

- **Music:** All music files were stored in this folder. The stereo, DVRs and iTunes applications on household computers pointed to this directory.
- **Video:**
  - **Movies:** All movies were stored in this folder. The DVRs had access to this directory for movies.
  - **TV:** The DVRs placed recordings of TV shows in this folder, which other devices could also access as mpeg streams.
- **Users:** Each user had their own sub-folder in this folder where the files from their "My Documents" were placed.

I provided users with 250GB of storage and backed up the full system in case of system failure. To avoid problems with network bandwidth, I put the server and DVRs on a wired gigabit network and connected this wired network to the household's wireless router.

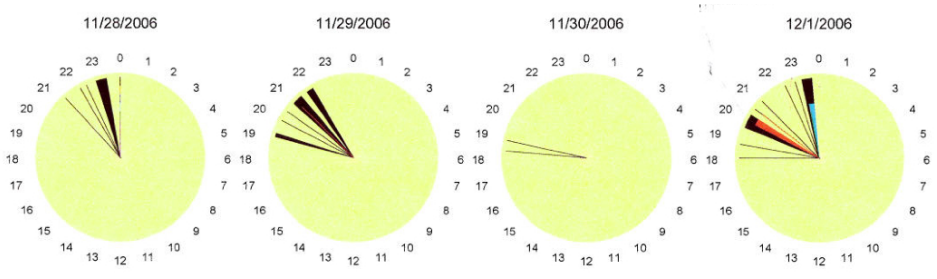

Figure 3.1. **System usage form.** This shows an example of the forms I used to review system usage with each household. Each circle represents one day, across time, and different colors represent different types of access. This shows four days of access on Phoebe's laptop.

To allow household members to control access to their personal files, I let them specify a private folder only available to a user-specified set of devices and a public folder available to all machines.

3.1.2 Investigation methods

Questionnaires: To begin the study, I conducted a group interview with the collective memberships of each household. I also held one-on-one interviews during which I asked each household member to help me fill out a form detailing the devices they used, the data they stored on each, and their reliability, mobility, and privacy preferences for those data.

Technology tour: At the outset of the study, I had the household members walk me through their homes, showing me each digital storage or playback device, and describing what data was stored on it and how it was used. This allowed me to get a sense of the technology they used regularly and the perceived differences between devices.

Tracing: I added a tracing module to the server to allow me to see which data users accessed from which devices at which times, and provide some insight into daily and weekly patterns in access behaviors.

Weekly interviews: To get a better sense of users' interactions with the system, I held weekly interviews with each household. For these interviews,

Figure 3.2. **The Bradys.** Left to right: Mike, Carol, Oliver, Bobby and Peter.

I brought visual representations of their weekly traces, to start the conversation about usage behavior, as shown in Figure 3.1. Both the diagramming technique and its application as an interview prompt were borrowed from Aipperspach's study of laptop usage [3].

3.1.3 Household selection

The goal of this study was to explore the effect of data sharing across devices in households. In order to minimize the learning curve involved in adopting such a system, I recruited households that listened to music and that already had cable and Internet access. However, I avoided self-described computer experts, to avoid biasing the study towards this population.

I screened for households with at least three adult members, so that the behaviors and patterns would be more involved than what might be found in a two-person household. Since the system did not support taking devices and data out of the household, I targeted households that did not frequently take devices outside of their home.

I recruited two households representing different configurations by posting flyers and through snowball sampling following personal connections. One household consisted of a blended family, and the second consisted of a set of adult roommates.

Household 1 - The Bradys: Figure 3.2 shows the Brady[1] family. Mike and Carol Brady married 10 years ago, and both had children from previous marriages. The Bradys lived in a suburb of Portland. Two of Carol's sons still lived at home. Peter was in high school and was interested in computers. Bobby was in junior high and was perhaps the least interested in technology in the family, although he played video games with his brothers. Carol's nephew Oliver also lived with the Bradys. Oliver was attending college and had recently started studying information management. Oliver was living with the Bradys for a year and often worked somewhat separately from the family. Carol's oldest son, Greg, did not live with them, but was a frequent visitor, along with his fiancé, Marcia.

The Brady's had two TVs that they connected to the system in the house: the upstairs TV in the living room and the downstairs TV in the den. A stereo system with large speakers was located in the family's living room. At their request, we hooked our new laptop stereo system through their existing system. Carol had a new laptop that she used to run a business out of her home and that the rest of the family used on occasion for personal activities. Oliver had his own laptop that he used for school and that his cousins used to play games. Peter was the primary user of an old desktop computer, which was located in the downstairs den. Bobby had an even older desktop in his room, which he only used for games, that was not connected to the Internet.

Household 2 - Piper, Paige and Phoebe: Figure 3.3 shows Piper, Paige and Pheobe. All three were young professional women in their mid to late 20s. Piper owned their house in another suburb of Portland and had family nearby. She taught film and health at a nearby high school. Phoebe had lived with Piper for about a year and was an interior designer. Both Piper and Phoebe liked technology and enjoyed tinkering with it. They did a lot of filming and photography, including a large number of home videos and images modified in Photoshop. Paige moved in a few months before the study. Paige had little interest in technology and used it sparingly. The

[1] All names have been changed to preserve anonymity.

Figure 3.3. **Left to right, Paige, Piper and Phoebe.**

Figure 3.4. **System setup.** This photo shows one DVR and the stereo setup in Piper's living room.

three roommates had many common friends and spent a lot of time together. They often threw parties and had friends over to their house.

Piper, Paige and Phoebe chose two TVs in the house to connect to the system: one in the main living room and one in the study (see Figure 3.4). At their request, we put the laptop-based stereo system in the living room. Piper owned a desktop computer that was placed in the study. Phoebe had a fairly modern laptop that she used throughout the house, including in her bedroom. Paige owned an older laptop that was not connected to the Internet and that she usually used in her bedroom. Piper used a portable hard drive to store and transport her films.

3.2 Instant data access

One key observation from the study is that instantaneous data access has a big impact on the way users interact with their data. While they could carry media from device to device before having the system, the effort required inhibited the data usage of both the households. Once they were able to access the data immediately on any device, they told us they used their data much more frequently. They also reported that they spent more time in public places, as their private data was now connected with public devices. This section outlines the data sharing methods that the households used previous to the installation and the impact of the deployment on their usage.

Pre-study file sharing methods: Before installation of the system, household members primarily used email and physical media to move data from device to device. They used CDs and DVDs (and, in one case, a portable hard drive) to transport large media, and USB drives and email for smaller files.

The common element of all these methods is the need to explicitly copy data from one device to another, often by moving a piece of media or a device to the location where the data is desired. We found that making data transparently available, thus removing the need to move data to the viewing device, improved the usage experience.

Despite the heavy use of distributed file systems in the enterprise, no one in either household used such file sharing systems at home or, for the most part, outside the home. Piper used two computers at work that were supposed to share data, but because of technical problems they did not. She explained that she "gets a lot of exercise" running data from one machine to the other.

Instant data access increased data usage: Intra-device data sharing led to a reported increase in data usage over the methods listed above. Study participants expressed great enthusiasm for having data from multiple users aggregated at shared devices, like the stereo and TV.

However, despite this enthusiasm for sharing data between devices, the study reinforced the fact that current methods limit the amount of data

people will use. Phoebe explained that having the data readily accessible made them think about it more.

Phoebe: *Our homemade movies we used more, pictures we were all able to view those. It was just something, because it was there, we thought about it more. You know, like I said before it was more accessible. So it's in front of you more.*

Factors that contributed to users' perceived contrast in the accessibility of their files — and, therefore, of their expressed likelihood of using them given the aggregated storage systems — included the simple removal of the need to compile media selections and physically retrieve storage media, the danger of damaging (and mess of managing) physical media, and the immediacy of shared data resources. Piper explained that the hassle of moving media made a large difference in the amount they used the stereo. She explained that they did not use the stereo much without the sharing system.

Piper: *It's SO much easier when it's all just files on there. I mean the difference is ridiculous. 'Cause if you want to just switch out you can. And now you actually have to open up the thing, and take out the thing, put a new one in.*

For user-generated content, another inhibitor to sharing was the need to create the media in the first place. For example, despite her excitement at sharing her videos, Piper explained that even after a year of work, she had yet finish the DVD needed to share them with others.

Carol's treatment of her photos reinforces this point. She was very proud of her photos: In fact during our tour of her home, when we simply mentioned her digital camera, Carol stopped and spent five minutes showing photos to the interviewer on a small camera screen. While she had bought CDs to transfer the photos over, she was still showing people the photos on the tiny camera screen because of the hassle of transferring them to a computer.

Carrying media around also led to conflicts between household members. One of the big reasons Mike didn't want a TV in the living room was the way the kids handled the removable media.

Mike: *[Downstairs the kids] will have 20 stacks of videos out, we don't understand that, why don't you put the video away after you finish? ... they*

still do the same thing with videos in [the living room]. No matter, nine years of telling them to put it away, they still pop 'em out and leave 'em out on ... the floor.

In contrast, Mike and Carol were happy to have the stereo in the living room, because they did not feel that it brought clutter. But, even with the stereo, moving CDs around caused some amount of discomfort. Mike explained that he knew the kids occasionally used the stereo because:

Mike: *We'll flip the CD in [the stereo] and go "uh oh, this must be someone elses' ".*

Household members also talked about the importance of having the media easily available right when needed.

Mike: *So that's nice, you get an immediate resource to something you are talking about. You know, if you go "oh, let's go downstairs to look at the TV," and sometimes, people when they are tired, they don't wanna go. But if you have it right there, you can just pop it up. ... because it was more convenient. Like I say, if I hadn't had my laptop there I probably wouldn't have.*

The timing was a critical aspect of media use because many of the items were very context-dependent. Piper, Paige, and Phoebe's discussion of Internet downloads further illustrates this point:

Piper: *You know, we'll pull up stuff. Things from our email we get from people. [laughs] Oh my gosh. Some really funny stuff. ...*

Paige: *That's normally at 2am though, when we're really tired. [laughs]*

Phoebe: *Yeah, you think "that's so funny" but the next day you look at it and,*

Piper: *that's not so funny.*

This extra convenience was important enough to both households that they expressed dismay at having to give up the system at the end of the study. Mike even asked us to tell him how the stereo system worked, because he wanted to try to build his own.

Instant data access increased socialization: Interestingly, users reported that having data shared between devices increased the time spent in public, social places. Because household members felt that they could access

all of their data from the homes' communal spaces, they were more likely to spend time in these spaces. In addition, because they considered their private technology and the shared technology connected, they spent more time near the shared technology. When asked whether and how the system had changed her lifestyle, Phoebe replied:

Phoebe: *... I used my computer out [in the living room] more, probably, than in my room. You know, not at my desk so much. Out in the public area, just because we had so much out in the public area to use. So... I could sit there and download something onto the screen, and then everyone could pull it up, and that kind of thing. Yeah, it centralized things more, so I think we all spent a little more time there [in the living room], than in the office.*

Similarly, we noticed that the printer associated with Carol's laptop moved from the Bradys' downstairs office up to the family room part way through the study, and the laptop spent more time upstairs.

Discussion: This suggests that it is important for home storage system designers to make data access easy and immediate when possible. Even small increases in the amount of effort required to access data in our study led to a large drop in data usage. Users were enthusiastic about immediate data access and reported that it led to an increased usage of their data, and an increase in the amount of time they spent in social places.

3.3 Decentralized and dynamic

Another high-level finding from the study was that the home environment is quite dynamic. Users would frequently switch between various devices, often in the course of a single task. Users managed their devices in complex patterns involving both multiple administrators for each device based on who was home, and multiple administrative domains, to mitigate faults and provide autonomy for household members. Even the household membership itself was often a fluid construct, with a number of even the heavy technology users in each household not actually living within the home.

3.3.1 Decentralized devices

During the study, household members were regularly choosing what device to use for a task from a large set of potential devices. In this section, we discuss the various factors affecting this decision. While the characteristics of the device itself were important, a variety of social and location factors were also crucial in making these decisions.

Hardware: One basic reason for data sharing was the difference in capabilities and interfaces for different devices. For example, the TV has a much bigger screen than the laptop and, thus, lends itself to showing photos to large groups of people in a way that the laptop does not.

However, even within the same device type, capabilities varied widely. For example, the boys in the Brady household played games on Oliver's laptop because it was much faster than their desktop machines, which could not run their games. This was exacerbated by the fact that devices lived for a long time in both households, both for financial and convenience reasons [26]. For example, the Brady household contained three generations of family desktops, with the older versions going to each son.

Software: Software also played a large role in differentiating computers. While in theory software can be moved from device to device, in practicality this was rarely the case. In many cases, the software was considered an extension of the hardware of the machine itself. For example, Phoebe would send some work documents to her home laptop, because it had a more up-to-date version of Autocad than that on her work machine. While she was able to eventually convince her boss to upgrade the work version, this was a long and uncertain process.

Piper did her video editing work on a computer at her parents' home, because this machine had Adobe Premier Pro installed, unlike her machine. Piper still used this computer partially because she had data and programs on the computer that she didn't want to take the trouble to move to another device. Instead, she had been working on convincing her parents to give the computer to her.

Piper: *And so, they might give it to me, because they got a new computer*

[crosses fingers and laughs] But then they were saying that we had to clean it and everything, and I was like, "Nooo, just give it to me," 'cause I have SO much stuff on there.

Location: Another differentiator for devices was their location. For example, nearby devices were easier to use. Paige and Phoebe both had small TVs in their rooms, which they used when getting ready in the morning or while working in their room. However, in general, they watched the much larger TV in the living room. Similarly, Piper would sometimes check her email on Phoebe's laptop, because it was close by in the living room, and she didn't have to go to back to her desktop machine in the study.

The atmosphere of the location also influenced these decisions. For example, the Bradys were careful to keep the living room TV different from the downstairs TV, to keep the living room neat. In contrast, they were happy to have the stereo in the living room, because they did not feel it brought clutter. Similarly, the office was primarily a work place for Piper, Paige and Phoebe. So, when someone was working in the office, they would not disturb them with entertainment.

User assigned function: Members of our households also differentiated otherwise similar devices by assigning particular devices to particular tasks. For example, Carol's laptop was purchased specifically for her business. While it was used for other purposes as well, all members knew that this was the device's primary function. Carol also wanted to purchase another laptop, to separate the work and personal devices. This finding is consistent with previous studies that showed that users preferred to use technology marked as specialized over those marked as general, even if there was no difference in the technology itself [46].

Household members would often use their personal devices over devices owned by other users. So, for example, Peter would primarily use his desktop machine for homework. However, when possible, they sometimes switched to devices with better capabilities. To continue our previous example, Peter would switch to Carol's laptop to print documents, because it was much faster. In addition, household members would sometimes avoid performing

tasks on certain machines for fear of breaking the device: For example, gaming in the Brady household was not allowed on Carol's laptop.

Failure: Household members would also use alternative devices when technical difficulties arose. For example, part way through the study, Piper had trouble with her computer, so she temporarily used Phoebe's laptop as a replacement device until she could get the desktop fixed.

In a different take, when Oliver accidentally broke one of the DVRs while working with the wiring, Mike used the second (working) DVR as a template and was able to discover and repair the problem.

Discussion: This suggests that home-focused storage systems must be flexible enough for users to customize how and when they use the devices participating in the system. The system should also accommodate the role of devices changing over time, including for short disruptions due to temporary circumstances.

3.3.2 Decentralized administration

Administration was a challenge during the study. It took a full day or more of my time to install the system in each household, even after setting up several working installations in the lab. Even with a PhD student as a system administrator, both households ran into challenges in understanding and keeping the system running.

In the enterprise, *administrators* are the employees responsible for setting up and repairing technological devices, and average users do not involve themselves in these repairs. While not institutionalized, we found similar distinctions in the home, where some users were administrators and repaired technology, while others did not.

However, in contrast to the enterprise, we found that each household, and many devices, had multiple administrators who shared management tasks in an often ad-hoc fashion. This finding is somewhat at odds with previous studies who observed a single household "technology czar" [12, 26]. This difference may be due to our focus on non-technical users, where these

previous studies did no such screening. A technical household may be more likely to have a highly-technical individual to fill this role exclusively.

Multiple administrators: For example, we found that both of our households had several administrators, despite only having a handful of devices. For example, Piper and Phoebe would work together to figure out problems, while Paige was not involved and usually only heard about problems after they were fixed.

Similarly, in the Brady household, Mike, Oliver and Peter all took turns at working with the technology when challenges arose. Carol and Bobby did not do administration and instead would call for whichever of the administrators happened to currently be at home. This meant not only that the same device might be repaired by multiple administrators, but that previous repairs might have occurred without the current administrator's knowledge.

Multiple areas of authority: In addition to having multiple administrators, each home also had different administrators for different devices. The public devices were usually managed in combination, but many devices had private owners and were managed separately. For example, Mike Brady explicitly gave each son his own computer so that they could play their games on these devices without breaking the family computer.

Mike: *I have lost programs in the past because of a crashed computer so it's always a concern. Especially when they get these games going on. That's what scares me, these things that they do, you know to performance optimize the ability of the computer. ... We kinda keep as much separated as possible.*

Phoebe explained that this was another reason she password protected her laptop during college; a fellow student used her laptop for the Internet, because it was available, and accidentally contaminated the machine with a virus. Piper even made an administrative division between her C: and D: drives.

Piper: *So, I keep most of the stuff on the D: drive if I know that I don't want anyone to access it, cause I have had my C: drive crash before and lost a lot of stuff that way. So usually when I download I use the C: drive, the D: drive is never touched by outside sources. So, I never have a problem with that. That's usually where I keep my digital photos.*

Discussion: The homes in my study contained multiple managers and multiple administrative domains, even in the context of comparatively small homes. Many devices were administered by multiple people, at different times, in loose coordination. My users also made explicit decisions to insulate the administration of some devices from that of other devices, for both reliability and autonomy.

3.3.3 Dynamic user base

Household membership was a complex thing in this study. Each household had members who did not actually live in the home, but were heavy technology users, and household members who were temporary in various ways.

Occasional household members: In addition to the residents of the household, each household had instances of *occasional* household members, who did not reside in the home but were frequent visitors and users of the technology in the home.

For example, Piper was still a household member at her parents' house. During the study, Piper had trouble with the licensing for a piece of software on her computer, since she had inherited the computer from her parents but no longer lived at home. Their discussion reveals the ambiguity in Piper's status as a "household member."

Piper: *It was my family's, ok, I'm still part of my family.*

Paige: *And that household. [laughs]*

Piper: *Hey, I still have stuff over there.*

Paige: *You still have a ROOM, and a spot at the dinner table, I still think that counts.*

Similarly, while Greg and Marcia did not live with the Bradys, they were frequently in the home and were large users of the Brady's technology, including Carol's laptop. Some of these occasional members were also temporary visitors, such as Carol and Mike's children who returned and visited over the holidays.

Temporary household members: In addition to occasional household members, a number of the households members were also only in the home

temporarily. For example, Oliver was living with the Brady's but would be moving out after finishing the year of school. Piper, Paige and Phoebe lived together, but were unsure of when each of the roommates would move out.

Discussion: Because the user base in a home is so dynamic, it is important for system designers to make it easy for users to enter and leave the home, along with their associated devices. Designs that require time-consuming integration and dissociation processes may be difficult for users to adopt.

3.4 Privacy

One important task in managing data systems is *access control*, or controlling who can access which files. This is important to provide privacy for sensitive data and security for important data.

On the first day the system was installed, a lack of adequate access control caused problems. While each user could choose the privacy of his local files, music to be shared on other devices had to be shared on all devices. Peter listened to a variety of music on his desktop machine downstairs. Once the system was installed, all of this music was accessible from the DVRs and the stereo upstairs. Mike quickly noticed that this music contained songs with titles that concerned him, especially songs by several rappers Peter enjoyed. Mike immediately began deleting these songs, initiating a power struggle between Mike and Peter. Mike was concerned about the type of music Peter was listening to and felt an obligation to enforce this rule. Peter expressed frustration about the event and felt that Mike was unfairly judging his music.

This problem had been less prevalent before the system. The addition of data sharing sharing disrupted the previous social balance. When asked if the system had made this problem worse, Peter said:

Peter: *Yeah, kinda, he just went on the TV and just like went through all the music and just saw what he didn't want and deleted it. And again on the stereo, too, cause he didn't want to see them on the stereo, I guess.*

While access control was important in the home, it was also a very different environment than that found in most enterprises. We discuss these differences in the next sections.

3.4.1 Users and passwords

Current computer systems, both for the home and enterprise, rely on the abstraction of a *user*. Under this abstraction, each household member would log into the computer with a username and password. In a distributed setting, users carry the same username across devices. This allows users to sub-divide devices, customize their experience, and control access to important information. We found that this abstraction was not used in our households. These findings support findings by Brush et al. [12], who also observed challenges in the user abstraction for the home.

No user accounts: Neither of our households used the user abstraction on any of their devices. The Brady desktop, which was primarily used by Peter, had multiple users defined, but all family members signed on as Mike. They were surprised to learn (during the course of the study) that there were other users defined on the machine. All other machines had a single user defined.

No passwords: Of the eight participants in our study, only Phoebe used a password on her laptop. When asked why she was concerned about password protecting her machine, Phoebe explained she has always been a private person. However, she also revealed that this was a habit she developed while working in a computer lab setting at school and carried it over from this environment because she had "got used to it."

Discussion: This suggests that the use of passwords, while prevalent in the enterprise, may not be a natural fit for the home environment. In the remainder of this section, I explore why this may be the case and discuss alternative methods home users employed to protect their data.

3.4.2 Why is the home different?

Threat model: The biggest difference appears to be in the threat model assumed in the home. In a business setting, adversaries are assumed to be malicious, requiring strict mandatory control. However, in our households, there already existed a high degree of trust between household members, causing them to take a different approach to access control. While some households may not have such trust, it is reasonable to assume that a large number of households will fit the pattern I observed.

Consider Piper's reply when asked why she was not concerned about her data on the desktop being unprotected:

Piper: *'Cause I trust [my roommates], I don't think they'd do that, go through stuff that was mine that I wouldn't want them to look at. And even if they could I doubt they could find anything I don't want them to see on there. [laughs] You know, so I don't really have much to hide. So it doesn't really bother me.*

This quote hits many of the key points we found in home access control: When asked about access control, many of the household members gave similar replies to Piper's. In general, household members were fine with their data being accessible to other household members, because they did not consider their information overly sensitive, and they had a high level of trust with other household members. While many household members had certain, specific items that they did not want shared, in general they defaulted to "all data shared" for most data.

While almost all household members said they had nothing to hide, up front, most discovered that there was indeed data that they did not want shared when actually moving data in the system.

For example, Carol initially said she was fine with all the data on her laptop being shared with all household members. However, as we worked with Carol to import her data into the system, we ran across a set of journal entries that she stored on her laptop. She was very uncomfortable with this data being shared on other machines and made sure to make it private to the machine. In fact, she was uncomfortable even talking about it with her

husband nearby. Similarly, when questioned more closely about her data, Piper, who had previously said she was not concerned, confessed she had private data as well, which she did not put on her desktop machine.

Social pressures: Social pressures also played a very different role in the home. Especially in the Brady household, where the dynamic of parent and teenager was important, these effects were nuanced and complex. For example, when asked if he would allow Peter to have a file space over which Mike did not have control, Mike was not entirely sure how to answer.

Mike: *[long pause] It depends on what files. ... I'd have to go by their honor in some case or another. ... I'd want them to have some privacy to writing letters to girlfriends and things like that. ... I'm just more concerned about the content, like the music. If there was some way of finding things by content, like some things of a sexual nature, ... I'd probably do that. I should trust them more but, you always should, but if something came up that I was concerned about it, I'd want to be able to check it up and discuss it or something.*

Mike was balancing Peter's need for privacy against his responsibility as a parent to watch out for him. This policy was nuanced and required great thought on his part. However, like the other users in the study, he was more adept at manipulating locations and physical devices than computer-centric abstractions, like users and permissions, and so used these approaches to create policy.

3.4.3 Home access control methods

The remainder of this section discusses the alternative way ownership was handled in our households and the implications for system designers.

Device as security boundary: While users did not utilize passwords to protect their data, they were skilled at using other techniques to protect their sensitive information. For example, while Carol was very concerned about the privacy of her data, she was comfortable with it being stored on her laptop. Although the laptop belonged to Carol, it was also shared with

a variety of household members, including occasional members like Marcia, without any kind of password protection.

The device boundary was sufficient for Carol to feel comfortable about her data. She owned the device and could control who had access to it. This was the way users in both of our households controlled access to their data. They were concerned with the devices that had access to the data, but preferred to control access to these devices using location and control of the physical device, rather than passwords and users.

Hiding: Much of the access control we observed involved hiding files. This method relied on the assumption that other users would not, or could not, find the data stored on their devices. Piper's previous comment states that she does not think her roommates can find her documents, later comments about her private data further elaborated:

Piper: *Yeah, there are a couple things [I didn't want people to see] [laughs] There are some pictures I don't want people to see.*

Interviewer: *Would you be comfortable putting them in [the standard location for her files]?*

Piper: *[still laughing] Probably not.*

Interviewer: *So where do you keep those?*

Piper: *[still laughing] I just keep them on my D: drive. It's not in [the standard location for her files] even on my C: drive.*

Interviewer: *Hidden.*

Piper: *Very hidden!*

This assumption is not always true, of course. For example, Mike explained how he once caught a file that Peter tried to hide from him:

Mike: *All these kids think they're pretty computer smart. I'm not computer dumb [laughs] and I've got enough orneriness into me that I almost know what these guys do, and so its like he one time hid something under another file and I found it. "How did you know that?" well I'm not stupid, I'd probably do that myself.*

However, despite his ability to find individual files when looking for them, Mike did not search through Peter's desktop. When asked why he deleted Peter's music files, which he had not deleted previously, he replied:

Mike: *Well, I never had access to 'em, see I wouldn't know what he was listening to. ... On a system like this at least I know what they're listening to.*

This suggests that though he knew that he could search through Peter's data, he did not consider doing so an option. Later comments suggest that he was interested in looking for specific, banned files, but didn't want to peruse Peter's data. This also seems included in the initial quote by Piper in this section; while she didn't necessarily think her data would be safe from a dedicated attacker, she assumed her roommates would not intentionally search for her private data, much as they would not rummage through her room for personal items.

Location: Location also had a significant impact on privacy concerns. For example, during one interview, Phoebe was uncomfortable discussing one video usage, where she watched a video on a laptop in her room, because she was watching it with a visitor who had dated Piper.

On the opposite side, Carol explains why Peter's desktop is in the main room downstairs and why she doesn't want to buy him a laptop.

Carol: *The boys need a new desktop or laptop downstairs. I'm kinda having a problem with Peter and games and attitudes. I'm NOT wanting to buy a laptop. On desktops, he kinda has to sit in the same place [rather than playing games in another room]. ... We've had to do some stuff [to limit his game playing]. Not that he's a bad kid, just the addictiveness of it, and the attitudes that come from it.*

Similarly, Peter said he would want his music available on the downstairs DVR, because it was only used frequently by the kids, but not on the upstairs DVR or stereo, since this area was primarily used by the parents.

Physical analogues: While device boundaries, hiding and location may seem weak in contrast to methods used for enterprise access control, they are analogous to the access control methods found for many other items in the household already. For example, Carol and Mike separate their movies into an approved collection, which is stored downstairs next to the TV, and a restricted collection that they want to control more closely, which is stored in Mike and Carol's bedroom. The kids are required to ask permission before

using one of these restricted movies. However, these movies are not locked away; Mike and Carol rely on a combination of trust that the kids will obey the rules, and the increased scrutiny that this location provides, to maintain access control.

Discussion: Our findings suggest that home system designers may want to support a weaker set of security primitives than might be provided in an enterprise setting. Just as much of the physical access control in the home is provided without locks, much of home digital access control appears to be provided without password protection. The high trust level inside the home may also suggest a "default to open" policy for sharing, in contrast to the enterprise.

3.5 Summary

In this section, I summarize the high-level findings from the home server deployment study.

Instant data access: Even small increases in the amount of effort required to access data in my study led to a large drop in data usage. Users were enthusiastic about immediate data access and reported that it led to an increased usage of their data and an increase in the amount of time they spent in social places.

Decentralized and Dynamic: The users in my study employed a wide variety of computers and devices among which they fluidly switched in order to accomplish their given tasks. While it was not uncommon for them to have a set of primary devices at any given point in time, the set changed rapidly, the boundaries between the devices were porous, and different data was "homed" on different devices with no central server. This matches with previous studies of data archiving [43]. This was exacerbated by frequent changes in the users living the household, who moved their devices with them.

Distributed management: The homes in my study contained multiple managers and multiple administrative domains, even in the context of comparatively small homes. Many devices were administered by multiple

people at different times in loose coordination. My users also made explicit decisions to insulate the administration of some devices from that of other devices, for both reliability and autonomy.

Privacy control: Privacy was important to the users in our study, and a distributed filesystem with conventional access control was not adequate for their needs. Just as much of the physical access control in the home is provided without locks, much of home digital access control appears to be provided without password protection.

4 Home data contextual analysis

4.1 Introduction

I performed a *contextual analysis*, or set of in-situ, semi-structured interviews to explore a set of data management tasks. A contextual analysis is an HCI research technique that provides a wealth of in-situ data, perspectives, and real-world anecdotes of the use of technology. It consists of interviews conducted in the context of the environment under study. These interviews are *semi-structured*; the interviewer brings questions to begin and guide discussion, but allows the interview to proceed as it unfolds, to avoid biasing the results towards a particular set of hypotheses.

I interviewed 8 non-technical households and asked questions about the way they replicate their data, the way they respond to device failure and upgrade, and the way they organize their data. This study gave me a deeper insight into the way these tasks are performed and the challenges associated with them. I used this information in turn in the Perspective filesystem and Insight management toolset.

This chapter outlines the study methodology and some high level findings from the study. Section 4.2 outlines the study methodology. Section 4.3 describes the dynamic nature of home devices in the study. Section 4.4 describes semantic naming and challenges with directory hierarchies. Section 4.5 describes placement policies. Section 4.6 describes the heterogeneity of policies. Section 4.7 describes the importance of cost. Section 4.8 describes the importance of control to home users. Section 4.9 summarizes the results.

4.2 Study methodology

I interviewed eight households for my study, covering a total of 24 users (not counting small children), 81 devices, and over 12 hours of video. I recruited households with three or more members around the Pittsburgh, PA, and Washington, DC, areas who used a variety of digital information, such as digitally recorded TV and digital music, but were not computer professionals. To recruit participants, I used a combination of direct email, personal contacts, and fliers posted in public locations.

For each household, I conducted an hour-long interview with the members as a group in their home with all of their digital storage devices present. For the first half hour, I asked open-ended questions about data backup, device failure, and data sharing. For the second half hour, I asked each participant to bring forward each of their digital storage devices individually, tell us the device's capacity and utilization, look through the data on the device, and then break down the data stored on that device into classes based on reliability preference. For each class of data, I asked the participants to tell us the size of the class and the relative importance of the data not being lost.

I believe that a study of eight households, 24 users and 81 devices provides a significant cross section from which I can draw useful insights. Table 4.1 provides some details about the households in the study. Table 4.2 describes the devices in the study. Studies of similar size are common in the HCI literature. [12, 48] However, the study is biased towards roommates and young couples with or without young children. I was unable to recruit families with older children for this particular study. The full list of questions from the study can be found in Appendix A.

4.3 Decentralized and dynamic

The devices, data organization, and management patterns changed regularly in the households in study. This section describes my findings about device dynamism, organizational dynamism, and distributed administration.

Num	Adults	Children	Household type	Devices
1	3	0	Graduate student roommates	13
2	3	0	Professional/student roommates	10
3	3	0	Professionals roommates	13
4	2	2	Young family	5
5	4	0	Professional/student roommates	15
6	2	2	Young family	4
7	5	0	Student roommates	10
8	2	1	Young family	3

Table 4.1. **Households.** This table describes the eight thouseholds in the study.

Type of device	Number
Laptop	15
USB drive	12
Desktop	10
External drive	9
Cell phone	9
Portable music player	9
Web sites	7
CD/DVD	4
PDA	4
DVR	3

Table 4.2. **Storage devices.** This table describes the types of devices used in the households interviewed.

4.3.1 Device churn

Our interviews paint a very dynamic picture of the devices in the home. Users' devices were constantly changing due to upgrades, changes in employment, failure, and living conditions. One of our interviewees, Marcy, had owned 13 different cell phones in the last few years.

For example, 4 of our 24 users were currently or had recently been without a computer for an extended period of time during which they stored data on friend or family machines, or kept it on CDs or DVDs until getting a new machine. Another two users were in the process of pulling personal data from work machines due to a change in employment. Moving data forward from an old machine was often a difficult task:

Vania: "You format a computer, you have everything fresh and clean and its easy, and then you start trying to get data back in and it's a mess."

The result was a trail of data strewn across old and new devices. Four users explicitly mentioned data stranded on old devices because of the difficulty moving it forward. One user even described calling home to ask her mother to try to access files on an old desktop machine and read the needed information back to her.

Aaron: "Stranded is a good word for [the data left on my old computers]. An electronic desert island out there. An e-Island."

Of course some of the data left behind on old devices should be left there; a device upgrade is an opportunity to clean out stale data [43]. However, most users did little to ensure the correct data moved forward.

Aaron: "It really has been haphazard what's made it forward from disks onto [the new computer.] Not all [of the data made it forward from the last computer], not even close."

Restoring information from backups was a similar challenge and also a pain point. The two users who did mention having to do a restore both referred to it as a painful experience. One had last dealt with a failure 12 years previously and said, "I still vividly remember the anger". Another user was still between computers, but didn't actually think she could find and restore all her data once she had a new one. She said:

Figure 4.1. **Jill's backup.** This picture shows Jill's data backup, which she doubts she will be able to use.

Jill: "I have it somewhere, I can find it again, but it would be a **beast** *hunting it all down."*

The continual change in devices also tended to sweep aside any configurations that required work to maintain. For example, Kyle set up a networked drive so that they could access the files on their desktop from the laptop and, thus, work while keeping track of the kids. However, when they upgraded the desktop, they lost this feature. Kyle explained:

Kyle: "One of the reasons [the network drive] is now retired is because I dreaded trying to set it up on the new computer. This may be a very easy task for someone who is computer savvy, but 'mapping a network drive', meaning having a shortcut on my desktop that I could simply drag the files to, that took me about four hours, and four calls to Bangalore, India."

4.3.2 Dynamic naming

Another interesting trend I found was the conflation of organization and other management tasks. Many users stated that they would go through and organize their data when they needed to move it from device to device, move data off of an overloaded device, or back data up. This event may serve as a good cue point for users to clean their file naming [41]. However, we also observed this trend inhibiting change, as users put off needed management because they did not have time to reorganize the data. For example, Vania explained:

Vania: "Most of the time I had a lot of stuff on the laptop that I wanted to keep. But [now I don't store any real data here] because I have in the back of my head, when I have the chance I want to upgrade my laptop, so right now this is temporary, hopefully."

This dynamism made static naming and partitioning difficult. For example, Vania explained that he had trouble with his naming schemes over time.

Vania: "Even when you develop a systematic way, you don't predict some of the situations you are going to find, and then at some point it doesn't fit, and you make changes to your systematic way, or just this once I put it here, because I don't really know where to put it, and then the whole thing crumbles."

Similarly, Vania complained about the need to set static partitions on his computer, which quickly drifted out of sync with the needed usage.

Vania: "I reformatted my computer a couple of times. The reason is mostly because you start, and you create this partition for non-program stuff, and you end up stuffing C: with a lot of things, and then C: runs out of space so that D: doesn't run out of space and then your computer gets slow, and then you have to reformat."

4.3.3 Distributed administration

Administrative patterns differed between roommates and young couples. All of the young couples in the study had a single administrator who coordi-

nated most of the management, while all of the roommates managed their own devices separately and shared administration of common devices. Unfortunately, it is unclear how this applies to families with older children. The one example in the home server deployment, the Brady household, had distributed management patterns.

4.4 Semantic naming and challenges with hierarchies

I found that many of our users had difficulty manging the folder structure of their filesystem. For example, many users found that application-level attribute naming was difficult to correlate with the folder structure of the filesystem. In addition, several users found it difficult to manage the different information they needed within the folder structure.

4.4.1 Semantic applications, files-and-folders filesystems

In our interviews, we found a fundamental disconnect between the way that users accessed data normally and the way in which they had to manage that same data. In normal data access, most users utilized semantic abstractions through applications like music players, frequently accessed files lists, and desktop search. Most users also used email and online photo sites, and they expressed little difficulty in navigating these primarily attribute-based systems.

In contrast, when managing this same data, users were required to use the filesystem files-and-folder scheme. We found that many users were completely unaware of the files-and-folder scheme that existed on their computers, since it was almost always hidden by applications. Of 24 users we interviewed, we found that at least 7 of them did not know how to use the file browser. Many users told us things like, "I don't know where my music is stored, I just access it through iTunes."

Even experienced users had trouble with the disconnect between application data access and the filesystem structure. Kyle described that the most frequent use of his backup drive was actually to find files that were lost on his computer hard drive. He explained:

Kyle: "I understand my backup drive because everything goes exactly where I put it. But, on my hard drive, every piece of software has someplace it downloads stuff, and towards the end of your computer's lifetime, you have so much junk installed, you don't know where you set it to download to. So, you go back to an application and you download something, and you don't know where it goes."

4.4.2 Challenges with files-and-folders

In addition to the mismatch between semantic applications and files-and-folders filesystem structure, users had other problems with the folder structure of the filesystem. While these challenges may be mitigated by the use of attribute-based naming, they do not exactly match attribute-based naming.

Many users had trouble finding files after saving them. For example, Marcy, a less experienced user, explained that she couldn't seem to find files once she had saved them, because she often wasn't sure where they went.

Marcy: "It reminds me of those drop down files, it goes C: Drive and then My Desktop, or My Computer, and then you save the file there, and then you can't find it again. And sometimes you can recover it by search, but I've definitely lost files on the computer."

Aaron's analysis of his wife's challenge in organizing data more succinctly describes the problem many users had with organization. They were careful about the name of files that they saved, but had much more trouble with the path where they saved the file.

Aaron: "I think you are fairly concerned about how you name your documents, but you don't pay much attention to where you save it."

Even users who were very attentive to organization struggled with the folder structure of their data. One user described his challenge in integrating different ways of organizing the same data:

Vania: "For example, we started by organizing photos by trips, but then we started just having photos for Brianna, then we do a trip and take pictures of Brianna. And then in Brianna you can't put photos randomly, so you start

organizing them by date. So now you have three ways of labeling the data that are intertwining."

This multiple-attribute problem was exacerbated by the fact that naming structures had to be useful for multiple purposes. We found that most users utilized backup and mobility policies that ran counter to the attributes used in their normal data access methods. By forcing them to use a single set of attributes, the hierarchies made both tasks more complex. For example, many used time to differentiate their data, putting old files on secondary machines and recent files on active ones. Users also utilized distinctions by task or period in the user's life (e.g., files from college or a particular project) or type (e.g., raw movie files vs. processed videos) to manage capacity.

Aaron expressed his frustration at having to dilute his carefully chosen file names by splitting his files up between various devices due to space constraints.

Aaron: "I'm very conscious about the way I name things; I have a coding system. But the thing is, that doesn't work if you have everything spread out. The coding system makes sense when there's a lot of other things around, but not when it's by itself."

A number of users indicated that the data they stored would not fit in its entirety on their portable devices, and they had difficulty carving off subsets to place on these devices. Vania described his frustration about his photos:

Vania: "At some point we just know this partition is photos and we back the whole thing up [and don't try to divide it], but it virtually becomes impossible to load it up on my laptop, because it doesn't have enough space."

4.5 Explicit, infrequent data placement

Home users carry a plethora of devices with them outside the home. All of the users we interviewed had cell phones, and most had music players that they carried outside of the home. A majority (13 out of 24) of the users interviewed also had computers that they frequently used outside of the home. It is critical that users be able to access their data on these portable devices.

While network access is frequently available, firewalls, corporate security policies and other barriers often make it difficult to connect to home devices from remote locations. For example, Kyle had to email data to himself from work and home, because his work firewall kept him from even checking his home mail, let alone connecting over the WAN. We believe that these observations argue for a decentralized storage solution that supports disconnection between devices, to allow users to carry data stored on devices from place to place, without requiring network access.

We observed that only 2 of 24 users had devices on which they regularly placed data in anticipation of needs in the near future. Instead, most users decided on a type of data that belonged on device (i.e. all my music, files for this semester), and only occasionally revisited these decisions when prompted by environmental changes. More frequent update operations usually involved copying all new files matching the device data criteria onto each device.

4.6 Heterogeneous policy

A one-size-fits-all approach for data placement would not be sufficient for the 24 users interviewed, as different users had very different opinions about mobility and reliability policies.

For example, some users kept most of their data on their laptop, while others kept recent documents on their laptop and older files on an external hard drive. Some households consolidated most of their data into a single grouping, while others kept their data separate between users.

Reliability concerns also varied between household. For example, while many users cared little about losing recorded TV shows, one household listed this data as some of the most important in the house — more important than any of their documents, and more important to one user than her photos. They were avid TV watchers, and during the regular TV season this loss would mean missing a show and having no way to catch up. When, a year earlier, their DVR had cut off the end of “American Idol” before the final decision, the experience was traumatic enough to warrant a heartfelt

Attributes	Example	Count
User/Type	Sara's pictures	63
Task	Andy's coursework	42
Type	Music	20
User	Vania's files	16
User/type/ad-hoc subset	Some of Tracy's documents	16
User/time	Jessie's old files	2
User/type/time	Rebecca's old documents	1
Task/time	Andy's recent teaching files	1

Table 4.3. **Data division attributes.** This table shows the frequency with which users in our contextual analysis used each attribute to define a set of their data.

mention during the interview. Another user in another household said that the data she missed most from a previous failure was the music she had purchased. This is in contrast to other users, who cared little for music or other media, but were vitally concerned about photos.

Mary: "The phone numbers are the worst. It's so embarrassing when someone calls and says 'Hey!' and you're like 'Hey?' and they're like 'What are you doing tonight?' and you're like 'Who are you?' "

Table 4.3 shows the various attributes that users utilized to sort their data. User and type together are the dominant attributes, but this may also be influenced by the fact that most OSes automatically choose this grouping by default.

Table 4.4 shows the percentage of data that each user wanted to protect from failure and how much they would care if the data went away. I was surprised at how much of their data users said they cared deeply about, although their replies are not tempered by any cost.

4.7 Money matters

Vania: "Yeah, we've been short on space on pretty much everything. We had like 150GB initially, we ran out of space, we bought 250GB, we ran out of

Reliability class	Percent of data
5 - Disastrous	53%
4 - Big loss, couldn't or couldn't want to replace	13%
3 - Considerable loss of time or money	17%
2 - Annoyance	8%
1 - Don't care	8%

Table 4.4. **Data reliability.** This table shows how much users would care if data was lost.

space again. Then we bought a 500GB external drive recently, now 350GB are taken."

With the increase in digital photography and video, and the crossover between home and work environments, it is common that users have data stored on home devices they are unwilling to lose. However, we found that users were loathe to spend money on more capacity than was needed and often explicitly chose to complicate their backup strategy to save money. During our interviews, 6 of our 24 interviewees explicitly mentioned cost as a reason they adopted more cumbersome data management strategies. Aaron summarized this point well:

Aaron: "I can't be as organized as I'd like because I have no central place to back up stuff. ... [but] if it's a choice between shoes [for the kids] and a nice external hard drive, then it will be the shoes."

Aaron: "I imagine it isn't too expensive to buy a large drive these days, but you know, too expensive is relative."

Even when resources were not as tight, users did not want to spend money on extra capacity if not needed. Sally explained:

Sally: "I don't have the money [for an external hard drive]. That's money I'd rather use to go to Spain."

The inverse of cost sensitivity is a drive to fully utilize the devices that users already owned.

Vania: "I don't like the idea of having an extra drive being just used to backup. It seems like a waste of space."

These concerns about cost suggest that uniformly backing up all of the data in a home will not be acceptable to many users. Cost will increase as a concern as more media (TV shows, movies, etc.) become prevalent in digital form. We found that media-heavy users especially needed to differentiate between data types. Even Kyle, who was very concerned with reliability and backed up the family data regularly on DVDs stored in a safe deposit box at the bank, was unwilling to back up all their data and had a separate policy for raw video footage and finished items to save space and time.

This cost-sensitivity leads to a number of users running short on space. We found that many users (11 of 24) had run out of space at some point. A number of these users (5 of 24) were heavy media users, either in video or audio. However, even average users spoke of running short on space and having to remove data or buy new capacity.

4.8 Need to feel in control

Cost concerns notwithstanding, over half of users (14 of 24) did use some sort of backup to protect their data. However, many users expressed a distaste for spending time on these tasks and had sporadic backup strategies. Despite the distaste for spending time on managing reliability, very few used automated backup tools (2 of 14 users), which many participants described as being too complex for their needs and difficult to understand. Instead, most users who backed up their data (12 of 14) copied data by hand from device to device. Ramona summarized the sentiment we heard from several users:

Ramona: "I don't know that I necessarily trust those programs that back up your information because I feel like I know what I want, and I don't know what that's doing. I don't necessarily know that it's getting everything that I need, or that it's not duplicating the information that I already have."

The fact that users crave visibility into what their tools are doing on their behalf is a particularly powerful insight. We found that users want to understand what is being done on their behalf and are very distrustful of tools that completely hide complexity, unless they are extremely confident that the tool does what they expect.

Condition	Users	Total users
Adopted harder management method to reduce cost	6	24
Without a primary storage device within last year	4	24
Did not know how to use the file browser	7	24
Regularly used computers outside of their home	13	24
Anticipated future usage and used to "cache" files on devices	2	13
Has run out of space	11	24
Backs up data in some way	14	24
Uses some sort of backup tool	2	24
Hand copies data between devices	12	24
Uses email for backup of some items	5	24

Table 4.5. **Interesting statistics.** This table shows a summary of interesting statistics from the study.

In contrast, we found that 9 of 24 users used the technique of emailing copies of important documents to themselves as a crude means of backup and synchronization for small files.

Other users relied on external vendors to restore data from crashed computers, turning machines in to computer stores for restoration or moving old data over to new machines. However, several users mentioned losing data in these exchanges, even with routine problems. Of those users who used outside vendors, most of them had unpleasant stories about the experience, citing either perceived ineptitude or a frightening loss of control over their information.

Janna: "I felt like [the store that restored data from my crashed laptop] totally took advantage of me at the most vulnerable time of my life. I wish I had more control over what happened to me. It was as if I am putting everything I have: things that cost money, or make money for other people, in their hands, and they're not taking my time seriously."

4.9 Summary of results

This section summarizes the key findings from the study. Table 4.5 also shows a summary of key interesting numbers from the study.

Decentralized and Dynamic: The users in my study employed a wide variety of computers and devices between which they fluidly switched in

order to accomplish their given tasks. While it was not uncommon for them to have a set of primary devices at any given point in time, the set changed frequently, the boundaries between the devices were porous, and different data was "homed" on different devices with no central server. This was exacerbated by regular changes in the users living the household, and the devices they used. This matches with previous studies of data archiving [43]. One household had set up a home server, at one point, but did not re-establish it when they upgraded the machine due to setup complexity.

Heterogeneous policy: The users in the study had a variety of storage policies. While there were common approaches and themes, no set policy would accommodate all of their needs. Policies varied both for mobility concerns and reliability concerns.

Money matters: While the cost of storage continues to decrease, my interviews showed that cost remains a critical concern for home users. Note that my studies were conducted well before the Fall 2008 economic crisis. While the same is true of enterprises, home storage rarely has a clear "return on investment," and the cost is instead balanced against other needs (e.g., new shoes for the kids) or other forms of enjoyment. Thus, users replicate selectively, and many adopted cumbersome data management strategies to save money.

Semantic naming: Most users navigated their data via attribute-based naming schemes provided by their applications, such as iPhoto, iTunes, and the like. Of course, these applications stored the content into files in the underlying hierarchical file system, but users rarely knew where. This disconnect created problems when they needed to make manual copies or configure backup/synchronization tools.

Need to feel in control: Many approaches to manageability in the home tout automation as the answer. While automation is needed, the users expressed a need to understand and sometimes control the decisions being made. For example, only 2 of the 14 users who backed up data used backup tools. The most commonly cited reason was that they did not understand what the tool was doing and, thus, found it more difficult to use the tool than to do the task by hand. This suggests that automation techniques must

be careful about hiding complexity, unless they can convince the user that they will do the right thing in all cases.

Infrequent, explicit data placement: Only 2 of 24 users had devices on which they regularly placed data in anticipation of needs in the near future. Instead, most users decided on a type of data that belonged on a device (e.g., "all my music" or "files for this semester") and rarely revisited these decisions, usually only when prompted by environmental changes. Many did regularly copy new files matching each device's data criteria onto it.

5 View-based architecture

5.1 Storage for the home

The home is different from an enterprise. Most notably, there are no sysadmins—household members generally deal with administration (or don't) themselves. The users also interact with their home storage differently, since most of it is for convenience and enjoyment rather than employment. However, much of the data stored in home systems, such as family photos, is both important and irreplaceable, so home storage systems want high levels of reliability in spite of lax management practices. Not surprisingly, I believe that home storage's unique requirements would be best served by a design different than enterprise storage. This section outlines insights gained from the previous studies in real homes, design features suggested by them, and an overview of a *view-based filesystem architecture* that provides the design features suggested by my studies.

5.1.1 What users want

This section summarizes major findings from the studies described in the previous two chapters.

Instant data access: Even small increases in the amount of effort required to access data in my study led to a large drop in data usage. Users were enthusiastic about immediate data access and reported that it led to an increased usage of their data and an increase in the amount of time they spent in social places.

Decentralized and Dynamic: The users in my studies employed a wide variety of computers and devices between which they fluidly switched in order to accomplish their given tasks. While it was not uncommon for them to have a set of primary devices at any given point in time, the set changed frequently, the boundaries between the devices were porous, and different data was "homed" on different devices with no central server. This was exacerbated by regular changes in the users living the household, and the devices they used. This matches with previous studies of data archiving [43]. One household had set up a home server, at one point, but did not re-establish it when they upgraded the machine due to setup complexity.

Money matters: While the cost of storage continues to decrease, our interviews showed that cost remains a critical concern for home users. Note that our studies were conducted well before the Fall 2008 economic crisis. While the same is true of enterprises, home storage rarely has a clear "return on investment," and the cost is instead balanced against other needs (e.g., new shoes for the kids) or other forms of enjoyment. Thus, users replicate selectively, and many adopted cumbersome data management strategies to save money.

Semantic naming: Most users navigated their data via attribute-based naming schemes provided by their applications, such as iPhoto, iTunes, and the like. Of course, these applications stored the content into files in the underlying hierarchical file system, but users rarely knew where. This disconnect created problems when they needed to make manual copies or configure backup/synchronization tools.

Need to feel in control: Many approaches to manageability in the home tout automation as the answer. While automation is needed, the users expressed a need to understand and sometimes control the decisions being made. For example, only 2 of the 14 users who backed up data used backup tools. The most commonly cited reason was that they did not understand what the tool was doing and, thus, found it more difficult to use the tool than to do the task by hand.

Infrequent, explicit data placement: Only 2 of 24 users had devices on which they regularly placed data in anticipation of needs in the near fu-

ture. Instead, most users decided on a type of data that belonged on a device (e.g., "all my music" or "files for this semester") and rarely revisited these decisions, usually only when prompted by environmental changes. Many did regularly copy new files matching each device's data criteria onto it.

Distributed management: The homes in my studies contained multiple managers and multiple administrative domains, even in the context of comparatively small homes. Many devices were administered by multiple people at different times in loose coordination. My users also made explicit decisions to insulate the administration of some devices from that of other devices, for both reliability and autonomy.

Privacy control: Privacy was important to the users in my study, and a distributed filesystem with conventional access control was not adequate for their needs. Just as much of the physical access control in the home is provided without locks, much of home digital access control appears to be provided without password protection.

5.1.2 Designing home storage

From the insights above, I extract guidance that has informed my home filesystem design.

Global namespace: Because immediate data access is so important, a home filesystem should provide a global namespace. A global namespace allows data stored on any device in the home to be accessible from any other device in the home.

Peer-to-peer architecture: While centralization can be appealing from a system simplicity standpoint, and has been a key feature in many distributed filesystems, it seems to be a non-starter with home users. Not only do many users struggle with the concept of managing a central server, many will be unwilling to invest the money necessary to build a server with sufficient capacity and reliability. I believe that a decentralized, peer-to-peer architecture more cleanly matches the realities I encountered in my studies.

Single class of replicas: Many previous systems have differentiated between two classes: permanent replicas stored on server devices and tem-

porarily replicas stored on client devices (e.g., to support mobility) [67, 52]. While this distinction can simplify system design, it introduces extra complexity for users and prevents users from utilizing the capacity on client devices for reliability, which can be important for cost-conscious home consumers. Having only a single replica class removes the client-server distinction from the user's perception and allows all peers to contribute capacity to reliability.

Semantic naming for management: Using the same type of naming for both data access and management should be much easier for users who serve as their own administrators. Since home storage users have chosen semantic interfaces for data navigation, replica management tools should be adapted accordingly—users should be able to specify replica management policies applied to sets of files identified by semantic naming.

In theory, applications could limit the mismatch by aligning the underlying hierarchy to the application representation. But, this alternative seems untenable, in practice. It would limit the number of attributes that could be handled, lock the data into a representation for a particular application, and force the user to sort data in the way the application desires. Worse, for data shared across applications, vendors would have to agree on a common underlying namespace organization.

Rule-based data placement: Users want to be able to specify file types (e.g., "Jerry's music files") that should be stored on particular devices. The system should allow such rules to be expressed by users and enforced by the system as new files are created. In addition to helping users to get the right data onto the right devices, such support will help users to express specific replication rules at the right granularity, to balance their reliability and cost goals.

Transparent automation: Automation can simplify storage management, but many home users (like enterprise sysadmins) insist on understanding and being able to affect the decisions made. By having automation tools use the same flexible semantic naming schemes as users do normally, it should be possible to create interfaces that express human-readable policy descriptions and allow users to understand automated decisions.

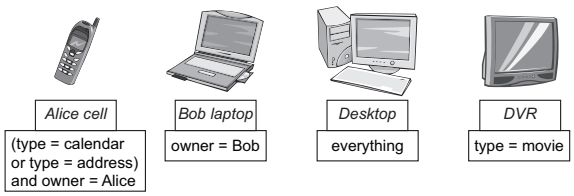

Figure 5.1. **An example set of devices and associated views.** This diagram, shows four devices, each with a customized view. Alice's cell phone is only interested in "calendar" and "address" objects belonging to Alice, Bob's laptop is interested in all objects owned by Bob, the house desktop is interested in all data, and the DVR is interested in all "movie" objects.

5.2 View-based architecture

To provide the design features from Section 5.1.2, I present a view-based filesystem architecture. It is decentralized, enables any device to store and access any data, and allows decisions about what is stored where to be expressed or viewed semantically.

A view-based filesystem provides flexible and comprehensible file organization through the use of *views*. A view is a concise description of the data stored on a given device. Each view describes a particular set of data, defined by a semantic query, and a device on which the data is stored. A view-based replica management system guarantees that any object that matches the view query will eventually be stored on the device named in the view. Figure 5.1 shows an example set of views. Example views include *"all files where artist='Aerosmith' and household='Smith"' stored on the Desktop* , and *"all files where owner='Brandon' and type='Contact' and household='Smith"' stored on Brandon's cell phone*, and *"all files where time created < Dec 2, 2007 and household='Smith"' stored on external hard drive*. I will describe the prototype's query language in detail in Section 6.1.1.

Figure 5.2 illustrates a combination of management tools and storage infrastructure that I envision, with views serving as the connection between the two layers. Users can set policies through management tools, such as those described in Chapter 7, from any device in the system at any time. Tools implement these changes by manipulating views, and the underlying

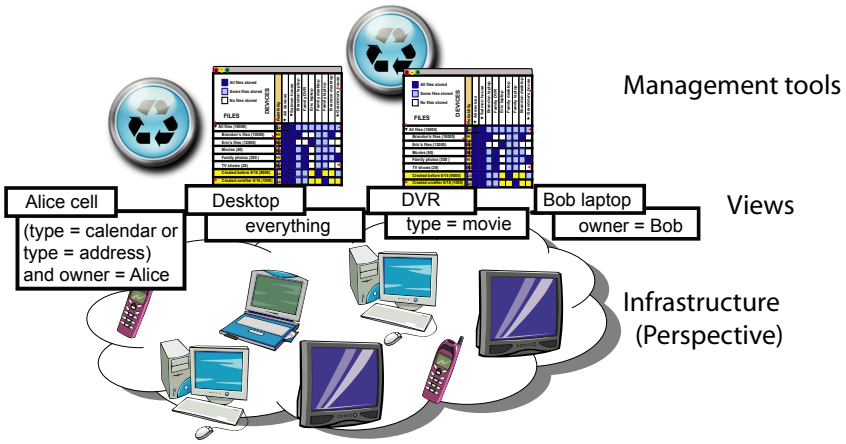

Figure 5.2. **View-based architecture.** Views are the interface between management tools and the underlying heterogeneous, disconnected infrastructure. By manipulating the views, management tools can specify data policies that are then enforced by the infrastructure.

infrastructure in turn enforces those policies by keeping files in sync among the devices according to the views. Views provide a clear division point between tools that allow users to manage data replicas and the underlying infrastructure that implements the policies.

View-based management enables the design points outlined in Section 5.1.2. Views provide primitive that allows users to specify meaningful rule-based placement policies. Because views are semantic, they unify the naming used for data access and data management. Views are also defined in a human-understandable fashion, providing a basis for transparent automation. A view-based system provides data reliability using views without restricting their flexibility, allowing it to use a single replica class.

5.2.1 Placing file replicas

In a view-based filesystem, the views specify the distribution of data among the devices in the system. When a file is created or updated, Perspective checks the attributes of the file against the current list of views in the system and sends an update message to each device with a view that contains that

file. Each device can then independently determine if and when to pull a copy of the update.

When a device, A, receives an update message from another device, B, it checks that the updated file does, indeed, match one or more views that A has registered. If the file does match, then A applies the update from B. If there is no match, which can occur if the attributes of a file are updated such that it is no longer covered by a view, then A ensures that there is no replica of the file stored locally.

This simple protocol automatically places new files and also keeps current files up to date according to the current views in the system. Simple rules described in Section 6.3 assure that files are never dropped due to view changes.

Each device is represented by a file in the filesystem that describes the device and its characteristics. Views themselves are also represented by files. Each device registers a view for all device and view files to ensure they are replicated on all participating devices. This allows applications to manage views through the standard filesystem interfaces, even if not all devices are currently connected. Note that this assumes that the number of views contained on the devices in each household and each device ensemble is relatively small (in the hundreds). This assumption is valid for the home, but might not be in the other settings.

5.2.2 View-based data management

This subsection presents three scenarios to illustrate view-based management.

Traveling: Harry is visiting Sally at her house and would like to play a new U2 album for her, while he is at her house. Before leaving, he checks the views defined on his wireless music player and notices that the songs are not stored on the device, though he can play them from his laptop where they are currently stored. He asks the music player to pull a copy of all U2 songs, which the player does by creating a new view for this data. When the

synchronization is complete, the filesystem marks the view as complete, and the music player informs Harry.

He takes the music player over to Sally's house. Because the views on his music player are defined only for his household and the views on Sally's devices for her household, no files are synchronized. But, queries for "all music" initiated from Sally's digital stereo can see the music files on Harry's music player, so they can listen to the new U2 album off of Harry's music player on the nice stereo speakers, while he is visiting.

Crash: Shawn's young nephew Gus accidentally pushes the family desktop off of the desk onto the floor and breaks it. Shawn and his wife Juliet have each configured the system to store their files both on their respective laptop and on the desktop, so their data is safe. When they set up the replacement computer, a setup tool pulls the device objects and views from other household devices. The setup tool gives them the option to replace an old device with this computer, and they choose the old desktop from the list of devices. The tool then creates views on the device that match the views on the old desktop and deletes the device object for the old computer. The data from Shawn and Juliet's laptops is transferred to the new desktop in the background over the weekend.

Short on space: Marge is working on a project for work on her laptop in the house. While she is working, a capacity automation tool on her laptop alerts her that the laptop is short on space. It recommends that files created over two years ago be moved to the family desktop, which has spare space. Marge, who is busy with her project, decides to allow the capacity tool to make the change. She later decides to keep her older files on the external hard drive instead and makes the change using a view-editing interface on the desktop.

6 Perspective: view-based filesystem

This section describes Perspective, a view-based filesystem that provides view-based replica management without requiring centralization. Our studies suggest that centralized approaches are a poor fit for the home environment, where the devices sets change regularly. This section describes the high level design of Perspective and the changes to core filesystem functions involved in supporting the more flexible view-based location scheme.

6.1 Search and naming

All naming in Perspective uses semantic metadata. Therefore, search is a very common operation both for users and for many system operations. Metadata queries (i.e. searches) can be made from any device, and Perspective will return references to all matching files on devices currently accessible (e.g., on the local subnet), which we will call the current *device ensemble* [68]. Views allow Perspective to route queries to devices containing all needed files and, when other devices suffice, avoid sending queries to power-limited devices. While specialized applications may use the Perspective API directly, we expect most applications to access files through the standard VFS layer, just as they access other filesystems. Perspective provides this access using *frontends* that support a variety of user-facing naming schemes. These frontends convert their lookups to Perspective searches. Our current prototype system implements four frontends that each support a different organization: directory hierarchies, faceted metadata, simple search, and hierarchies synthesized from the values of specific tags.

This section describes the Perspective query language, device ensemble creation, query routing, query caching, and frontends.

6.1.1 Query language and operations

Perspective uses a query language that includes logic for comparing attributes to literal values with equality, standard mathematical operators, string search, and an operator to determine if a document contains a given attribute. Clauses can be combined with the logical operators *and*, *or*, and *not*. Each attribute is allowed to have only a single value, but multi-value attributes can be expressed in terms of single value attributes, if necessary. We require all comparisons to be between attribute values and constant values.

While I could allow any attribute to have multiple values, allowing only a single value can be exploited to improve overlap computation. For this reason, I chose to support single value attributes as the core attribute.

However, attributes with multiple values can be implemented by overloading attribute names. For example, to create a keyword tag for a file with multiple values, we put a number of name-value pairs into a tag name. For example, to create a multiple-value tag with the name "keyword" and the value "Tree", the system can create the tag: "keywordTree=1". This conversion could be done at the system level by marking certain attributes as "multi-value" and then converting these attributes internally to multiple attributes of this form. Perspective relies on the application to do the conversion, as needed.

In addition to standard queries, we support two operations needed for efficient naming and reliability analysis. The first is the *enumerate values* query, which returns all values of an attribute found in files matching a given query. The second is the *enumerate attributes* query, which returns all the unique attributes found in files matching a given query. These operations must be efficient. Fortunately, we can support them at the database level using indices, which negate the need for full enumeration of the files matching the query.

Rule		Description
Query:-	object[Stmt]	Returns all the object matching the statement.
	object[Stmt]:Attribute	Returns all the values of attribute for objects matching the statement.
	object[Stmt]:*	Returns all the unique attributes for objects matching the statement.
	exists[Stmt]	Returns a single object if some object matches the Stmt.
Stmt:-	(Stmt)	True if statement is true
	Stmt and Stmt	True if both statements are true
	Stmt or Stmt	True if either statement is true
	not Stmt	True if statement is false
	Clause	True if the clause is true
Clause:-	Attribute=Value	True if named attribute equals value
	Attribute!=Value	True if named attribute not equal to value
	Attribute<Value	True if named attribute less than value
	Attribute<=Value	True if named attribute less than or equal to value
	Attribute>Value	True if named attribute greater than value
	Attribute>=Value	True if named attribute greater than or equal to value
	Attribute contains Value	True if named attribute contains string value
	Attribute	True if object contains the named attribute.
Attribute:-	string *(with no spaces or /)*	Name of an attribute.
Value:-	'string'	
	string *(with no spaces)*	Value strings without spaces can drop the quotes.

Table 6.1. **Perspective query language.** This table describes the Perspective query language in detail.

Query	Description
/object[type='Music' and artist='U2']	Find all files that have the type "Music" and the artist "U2".
/object[type='Picture']:album	Find all values of album for files with type "Picture".
/object[owner='Bob']:*	Find all the attributes found in files with owner "Bob".
/object[not(favorite)]	Find all files without a tag called favorite.

Table 6.2. **Perspective query examples.** This table shows several example queries in the Perspective query language.

This language is expressive enough to capture many common data organization schemes (e.g., directories, unions [56], faceted metadata [84], and keyword search) but is still simple enough to allow Perspective to efficiently reason about the overlap of queries. Perspective can support any of the replica management functions described in my dissertation for any naming scheme that can be converted into this language. Table 6.1 shows a full description of Perspective's query language, and Table 6.2 shows example queries.

Query comparison: Overlap evaluation is commonly used to compare two queries. The comparison of q1 and q2 returns one of three values. The comparison of q1 and q2 returns *subsumes*, if the system can determine that all the files matching q2 also match q1. The comparison of q1 and q2 returns *no-overlap*, if the system can determine that none of the files that match q1 match q2. The comparison of q1 and q2 returns *unknown*, if the system cannot determine the relationship between these two queries. Note that the comparison operator is used for efficiency but not for correctness, allowing for a trade-off between language complexity and efficiency.

For example, Perspective can determine that the query *all files where date < January, 2008* is subsumed by the query *all files where date < June, 2008*, and that the query *all files where owner=Brandon* does not overlap with the query *all files where owner=Greg*. However, it cannot determine the relationship between the queries *all files where type=Music* and *all files where album=The Joshua Tree*. Perspective will correctly handle operations on the latter two queries, but possibly at some cost in efficiency.

6.1.2 Device detection

Perspective queries find all files matching the query on any devices in the current device ensemble. A device ensemble is a group of devices that communicate with one other to share data [68]. For example, in the home, computers, digital video recorders, and devices from visitors will share data. Alternately, a family on a road trip may have laptops and music players share data.

Both of these cases are considered ensembles and are handled the same way in Perspective. Views allow Perspective to automatically adjust to the types of devices found in the ensemble, whether they are capable devices found in the home or an ad-hoc collection of devices outside the home.

Perspective uses Bonjour [10] and Avahi [5] to detect devices that arrive and leave the ensemble, in addition to using it to find IP addresses. Each device publishes itself as an instance of a Perspective service using Bonjour, with the device ID used as the name for the instance. However, an industrial research group has also easily ported Perspective to work with UPnP [77] as well. The data on a device is available from all other devices in the ensemble as soon as it has synchronized the view and device objects.

6.1.3 Routing queries

Because search is the core of all naming operations in Perspective, it is important to make it efficient. Rich metadata provides a great deal of useful content for search, and views can enable some useful optimizations.

By comparing a search query to the views in the system, the system can exclude devices that could not store the desired data. For example, the search for calendar data illustrated in Figure 6.1 is posed from the laptop. Using the views in the system, Perspective is able to exclude the DVR from the search, since calendar entries do not match its view.

Complete views can simplify the search process even further. Because a device with a complete view is guaranteed to store all objects that match that view, if a complete view subsumes the search, the device only needs to forward the search to the corresponding device. If the Desktop had a

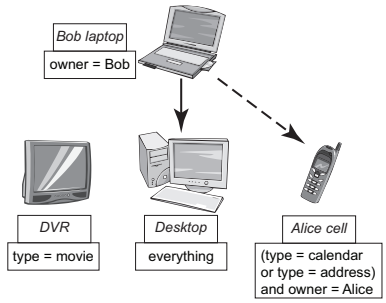

Figure 6.1. **Search.** Views can improve search efficiency. In this example, Bob's laptop performs a query for calendar entries for today. Because the laptop has a copy of all views in the system, it knows which devices could hold an object of that description. In this case, this would be both the desktop and the cell phone. However, if the desktop provided a complete view, then the laptop would only need to query the desktop, and not the cell phone.

complete view on "everything", the example search in Figure 6.1 could be executed just on the desktop machine and would not have to be propagated to the cell phone.

Efficient decentralized search is especially useful when dealing with devices from multiple administrative domains. One challenge in sharing data between domains is that they do not collaborate on name spaces, making it difficult to find data. However, by decentralizing the search process and using a semantic model, a device can see data on a device from another domain in exactly the same way it sees data from a local domain. It is free to do access control as it sees fit, and "domain" could be one part of the metadata.

6.1.4 Caching queries

To make search efficient, it is also important to be able to cache queries to avoid performing redundant queries multiple times. Perspective uses a lookup cache that maps the text of a query to a list of the metadata objects for matching files. If the same query is performed again, Perspective will use the result from the cache rather than actually re-performing the query.

The lookup cache is a write-through cache, all local updates are applied to the cache and then also applied to the backing store. If one of the views on the local device subsumes the query, then all required updates from remote devices will be forwarded to this device. If not, Perspective will mark the query as temporary, and will redo the query after a timeout period, similar to the approach used by NFS.

6.1.5 Frontends

While semantic naming provides a powerful way to find and access data, it is important to expose files to applications without requiring them to be modified for a custom interface. The *frontend core* is a framework that converts VFS calls into Perspective native calls. The core provides a common platform, onto which developers may place a number of frontends to customize the way semantic data is mapped into VFS. Applications and users may view the same data through multiple different frontends at the same time via this framework. Perspective uses FUSE to allow integration with the filesystem from userspace.

The frontend core maps open, close, stat, read, write and truncate FUSE calls into corresponding Perspective native calls. It maps getxattr and setxattr calls into readMetadata and writeMetadata function calls. Because there is not a clear one-to-one mapping between semantic names and a hierarchical representation, each frontend provides its own method of converting a path into a Perspective query. This is used by the frontend to complete readdir and getattr FUSE calls. Frontends are also given a callback on getattr and readdir calls, to allow for custom processing if needed. Other operations (i.e. unlink, create, rename, mkdir and rmdir, symlink and readlink) are highly specific to each frontend, so they are passed through directly to each frontend, although the core provides helper functions and common implementations to aid developers.

Each frontend is a C++ module written by a developer that can be linked against Perspective. Table 6.3 shows the API for a frontend. A frontend can

Function	Description
nameToQuery(*in* path, *in* queryType, *out* query)	Convert a given path and query type (i.e. file or directory op) into a Perspective query.
attrToName(*in* metadata, *out* names)	Convert the given file metadata object into name(s) for the results of a readdir call.
startingGetattr(*in* path, *in* query, *out* stat results, *out* handled)	A callback before a getattr call. The frontend can fill in the result if desired. If handled is set to true, the frontend will not continue with its normal path.
startingReaddir(*in* path, *in* query, *out* readdir results, *out* handled)	A callback before a readdir call. The frontend can fill in extra readdir names if desired. If handled is set to true, the frontend will not continue with its normal path.
rmdir, mkdir, rename, unlink	These calls are passed through directly to the frontend. The default implementation returns *not supported*.

Table 6.3. **Frontend API.** This table details the API for a frontend.

be used system-wide or might be customized to a particular application. Section 7.1 describes the frontends we have implemented so far.

Frontend queries: Frontend queries provide the central abstraction for frontends. Table 6.4 shows the format for a frontend query. The query specifies whether the query is *local* or *global*, a *base query* specified as a Perspective query, and a *frontend* that will define how the resultant files are displayed.

The individual frontend determines how the files are listed: as a flat list of files, as a set of directories, or via some other structure. The frontend core uses the frontend to convert from a directory into a Perspective query.

Each frontend is also responsible for implementing rename, mkdir and rmdir, and the implementation is highly dependent upon the frontend itself.

The unlink operator is dependent upon whether the frontend query is local or global. If it is a global query, then unlink will map to the unlink operator for Perspective. If it is a local query, unlink will map to the dropReplica operator in Perspective.

When a file is created, the directory in which the file is created is converted into a query by the frontend, and the frontend core then uses the query to apply a set of default attributes to the new file.

Mount table: The mount table maps from a set of root directory entries into a set of frontend queries. For example, the default Perspective mount table makes the root of the Perspective mount look like the facet metadata format by mapping it to the query */object[]facet*. Advanced applications

Rule		Description
Query:-	Basequery	A query on global data.
	local:Basequery	A query on local-only data.
Basequery:-	object[Stmt]	Query displayed as a flat list.
	object[Stmt]Frontend	Query displayed with the given frontend.
Stmt:-		defined in the Perspective query language.
Frontend:-	name	The name of the frontend.
	name{Arguments}	Frontends can have an options argument string.

Table 6.4. **Fontend query.** This table details the structure of a frontend query.

and users can access raw Perspective queries by listing a special formatted subdirectory in the .Raw directory. For example, to list all files using the directory frontend, an application could list the directory *.Raw/object[]dir*.

6.1.6 Application views

Much of a users' interaction with their data comes through applications, such as iTune or iPhoto, that package and manage specific data types. Thus, it is critical that file systems supporting these applications provide a way to keep the metadata in the filesystem coordinated with the metadata in the application, so that the user can manage their data with management tools using the same metadata as they are accustomed to using in their regular applications.

Perspective facilitates this through the use of *application views*. Application views provide application-level event notification, similar to the *persistent queries* proposed by Ensemblue [52]. An application can register an application view, which like a normal view is described by a query and a device, for the data in which it is interested. Perspective then ensures that every update matching the view will create a notification on the given device. The application can then query for new notifications using an ioctl-like interface to Perspective. The application can then use this information to keep its view of the data in sync with the filesystem.

Application views are implemented in a very similar fashion to normal views. Updates are routed in the same way as with any view. However, when

an update is received by the view manager, it places a notification in a local database, rather than applying the update. When the application asks for new updates, they are pulled from this temporary database. Perspective ensures an at-least-once semantic, but may provide the same update to an application multiple times. However, if a complete local view covers the application view, we can guarantee that, while updates may be received multiple times, an application will never receive an update less recent than any previous update it received for a file.

We have built a java framework on top of Perspective that simplifies the development of application-specific scripts using application-views. This framework is part of the Perspective library and is described in Section 7.2.3.

6.2 Partial replication

Perspective supports partial replication among the devices in a home. Devices in Perspective can each store disjoint sets of files — there is no requirement that any master device store all files or that any device mirror another in order to maintain reliability. Previous systems have supported either partial replication [27, 67] or topology independence [76], but not both. Recently, PRACTI [14] provided a combination of the two properties, tied to directories, but probably could be extended to work in the semantic case. Cimbiosis [59] has also provided partial replication with effective topology independence, although it requires all files to be stored on some master device. We present Perspective's algorithms to show that it is possible to build a simple, efficient consistency protocol for a view-based system, but a full comparison with previous techniques is beyond the scope of my dissertation. The related work section presents the differences and similarities with previous work.

6.2.1 Consistency

As with many file systems that support some form of eventual consistency, Perspective uses version vectors and epidemic propagation to ensure that

all file replicas eventually converge to the same version. Version vectors in Perspective are similar to those used in many systems; the vector contains a version for each replica that has been modified. Because Perspective does not have the concept of volumes, it does not support volume-level consistency like Bayou. Instead, it supports file-level consistency, like FICUS [27].

Full file copies: Like a number of other filesystems [67, 69, 76] Perspective stores a file in its entirety, or not at all. This greatly simplifies reasoning about the version of a file that is stored on a given device. However, this does cause challenges with remotely accessing large files, such as movie files. For these files, Perspective must wait until the full file is cached locally before allowing access. In the future, remote access to these kinds of files may require some sort of streaming support, although it would not require permanent storage of partial copies of files.

Version vectors: When an application updates an object in any way, it must increment the version vector associated with that object to reflect the change. A version vector stores a version number for each modified replica of the object in the system. It is used to determine which of two updates is more recent or to detect if updates have been made concurrently [51] .

We exploit extensible metadata to implement version vectors. We store each entry in the version vector as a metadata tag containing a replica ID and the corresponding version number. We can decrease the storage consumed by version vectors by simply omitting version numbers of zero, meaning that each object need only contain version numbers for replicas that have actually been modified [61].

Replica IDs are used to identify a replica's entry in the version vector. In many systems, the device ID is used as the replica ID. However, in Perspective, this is not adequate, because it allows devices to drop modified replicas from their local stores.

If a device acquires a replica, modifies it, drops the replica from cache, and then reacquires a replica at a later time, it cannot reuse the replica ID it used previously. Version vectors will only work correctly if a particular version of an object is unique. If a device reused a replica id, it could enter a corner case that would generate two object versions with the same version

vector; if it were to drop an object, later get an older version of the object and modify this older version, it might create a second version of the object with the same version vector, making it impossible to determine which update was more recent. Instead, we choose a new replica ID each time we create a new local replica.

The replica ID for a replica is stored in the metadata associated with the object. On a write to the object, the version manager uses the replica ID to determine the entry in the version vector that should be updated. When a device receives an update, it will clear the replica ID field and replace it with an appropriate local replica ID.

6.2.2 Synchronization

Devices that are not accessible at the time of an update will receive that update at synchronization time, when the two devices exchange information about updates that may have been missed by each. Device and view files are always synchronized before other files, to make sure the device does not miss files matching new views. Perspective employs a modified update log to limit the exchanges to only the needed information, much like the approach used in Bayou [76]. However, the flexibility of views makes this task more challenging.

For each update, the log contains the metadata for the file both before and after the update. Upon receiving a sync request, a device returns all updates that match the views for the calling device either before or after the update. As in Bayou, the update log is only an optimization; we can always fall back on full file-by-file synchronization.

Even occasional conventional full synchronization can be problematic for heterogeneous devices with partial replication, especially for resource- and battery-limited devices. For example, if a cell phone syncs with a desktop computer, it is not feasible for the cell phone to process all of the files on the desktop, even occasionally. To address this problem, Perspective includes a second synchronization option. Continuing the example, the cell phone first asks the desktop how many updates it would return. If this number is too

large, the cell phone can pass the metadata of all the files it owns to the desktop, along with the view query, and ask the desktop for updates for any files that match the view or are contained in the set of files currently on the cell phone. At each synchronization, the calling device can choose either of these two methods, reducing synchronization costs to $O(N_{smaller})$, where $N_{smaller}$ is the number of files stored on the smaller device.

Full synchronizations will only return the most recent version of a file, which may cause gaps in the update logs. If the update log has a gap in the updates for a file, recognizable by a gap in the versions of the before and after metadata, the calling device must pass this update back to other devices on synchronization, even if the metadata does not match the caller's views, to avoid missing updates to files that used to match a view, but now do not.

In addition, if a device gets an update, through a sync request, that is older than the version it currently stores, it must alert the device with which it was syncing of this new version. Otherwise, after a file is modified so that it no longer matches the views on a device, the device might pick up an old version of the file that does match the views.

Timestamp-based sync: The current implementation is simpler than the full design. It provides all the needed functionality, but requires n-to-n communication and does not allow filtering of updates. When a device applies an update, it creates a new local timestamp and puts this timestamp into the metadata for the file.

When a device receives a sync request, it returns the metadata for all files with timestamps later than the timestamp passed by the remote device. This does not allow for filtering of updates based on the views, since we don't keep intermediate updates, but it does assure correct operation and only requires devices to return new updates.

Device ensembles: In order to form an ensemble, devices exchange views with one another and sync with one another. When a device discovers a new device, it will send a copy of its views to the new device and sync any overlapping views with it.

When a device makes an update to an object or creates a new object, it keeps a notation and occasionally sends an *updatePending* RPC to all devices with matching views. Devices that receive the RPC in turn call *sync* on the source device.

Devices in an ensemble do not have to share an owner or central server in order to participate in an ensemble. An ensemble could contain devices from one household, and also devices owned by visitors, without requiring coordination through servers. Each device is free to decide whether to send searches to, or accept data or updates from, another device.

Each device is only required to store view and device objects from devices that contain replicas of files it stores, although they must also temporarily store view and device files for devices in the current ensemble in order to access their files. Because views are very small (hundreds of bytes), this is tractable, even for small devices like cell phones.

6.2.3 Update connectivity

To keep all file replicas consistent, we need to ensure that updates will eventually reach all replicas. If all devices in the household sync with one another occasionally, this property will be assured. While this is a reasonable assumption in many homes, we do not require full pair-wise device synchronization. Like many systems built on epidemic propagation, a variety of configurations satisfy this property. For example, even if some remote device (e.g., a work computer) never returns home, the property will still hold as long as some other device that syncs with the remote device and does return home (e.g., a laptop) contains all the data stored on the remote device. System tools might even create views on such devices to facilitate such data transfer, similar to the routing done in Footloose [50]. Alternately, a sync tree like that used in Cimbiosis [59] could be layered on top of Perspective to provide connectedness guarantees.

View freshness: View freshness timestamps allow Perspective to guarantee that all updates created before a given timestamp are safely stored in the correct locations and, thus, have the fault-tolerance implied by the

views and have appropriately up-to-date file replicas. Perspective calculates view freshness timestamps by tracking the local logical time and the wall clock time at which it last successfully synced with each remote device.

Each device stores this information in a *synchronization object* that is then replicated across all devices in the home, along with views and devices. Perspective uses these objects to provide a *freshness timestamp* for each view. Perspective can guarantee that all file versions created before the freshness timestamp for a view are stored on that view's device. It can also recommend sync operations needed to advance the freshness timestamp for any view.

The freshness timestamp of a view is the latest of a number of candidate timestamps computed from the synchronization objects.

- The earliest local sync timestamp for all devices with views overlapping this view.
- The freshness timestamp for a device D that subsumes this view, where the local device synced with D after the current freshness timestamp for D.

The current implementation only checks the first condition, which will work in the case where all devices sync with one another.

6.2.4 Conflicts

Any system that supports disconnected operation must deal with *conflicts*, where two devices modify the same file without knowledge of the other device's modification. We resolve conflicts first with a *pre-resolver*, which uses the metadata of the two versions to deterministically choose a winning and losing version. Our pre-resolver can be run on any device without any global agreement. It uses the file's modification time and then the sorted version vector in the case of a tie. But, instead of eliminating the losing version, the pre-resolver creates a new file, places the losing version in this new file. It then tags the new file with all metadata from the losing version, as well as tags marking it as a conflict file and associating it with the winning

version. Later, a *full resolver*, which may ask for user input or use more sophisticated logic, can search for conflict objects, remove duplicates, and adjust the resolution as desired. The current implementation of Perspective has only the pre-resolver implemented.

6.2.5 Capacity management

Pushing updates to other devices can be problematic, if those devices are at full capacity. In this case, the full device will refuse subsequent updates and mark the device file noting that the device is out of space. Until a user or tool corrects the problem, the device will continue to refuse updates, although other devices will be able to continue. However, management tools built on top of Perspective should help users address capacity problems before they arise.

6.2.6 File deletion

As in many other distributed filesystems, when a file is removed, Perspective keeps a *tombstone* marker that assures all replicas of the file in the system are deleted. The tombstone marker is ignored by all naming operations. To make the tombstone, Perspective truncates the file to zero length and sets a special deleted tag in the metadata. Note that deletion of a file removes all replicas of a file in the system, which is a different operation from dropping a particular replica of a file (done by manipulating views). This distinction also exists in any distributed filesystem allowing replication.

6.2.7 Garbage collection

Over time, tombstones will accumulate in the system and must be garbage collected. However, tombstones have little impact on the system and build up slowly over time. For this reason, garbage collection can be a background maintenance activity, much like filesystem defragmentation. The current implementation of Perspective does not include garbage collection, and has been storing my data for almost 2 years without a problem.

In order to ensure complete garbage collection, the system must use two phases of agreement, such as in ROAM [61]. However, it is possible to support garbage collection with a single phase. To do so, each Perspective device will track the local timestamp up to which each remote device has synced with it. Once the timestamp for each device with matching views is greater than the timestamp of the tombstone, the device can garbage collect the tombstone. This simplifies the garbage collection algorithm and allows the device to do garbage collection occasionally in the background.

It is possible for the tombstone to be reintroduced, if we roll off the edge of another device's log or if we haven't synced with a remote device past the tombstone on that device. However, this is invisible to the end user and, in the common case will not happen. So long as the tombstone is garbage collected eventually, it is fine for it to be reintroduced occasionally.

6.3 Reliability with partial replication

In order to manage data semantically, users must be able to achieve fault-tolerance on data split semantically across a distributed set of disconnected, eventually-consistent devices. Perspective ensures that data is never lost despite arbitrary and disconnected view manipulation using three simple distributed update rules. I will discuss a library that allows applications to reason efficiently about fault-tolerance in this environment in Section 7.2.4.

6.3.1 Update rules

Perspective maintains permanence by guaranteeing that files will never be lost by changes to views or by addition or removal of devices, regardless of the order, timing, or origin of the changes, freeing the user from worrying about these problems when making view changes. Perspective also provides a guarantee that, once a version of a file is stored on the devices associated with all overlapping views, it will always be stored in all overlapping views. This guarantee ensures a given number of copies in the system based on the current views.

These guarantees are assured by careful adherence to three simple rules:

(1) When a file replica is modified by a device, it is marked as "modified." Devices cannot evict modified replicas. Once a modified replica has been pulled by a device holding a view covering it, the file can be marked as unmodified and then removed.

(2) A newly created view cannot be considered complete until it has a valid freshness timestamp.

(3) When a view is removed, all replicas in it are marked as modified. The replicas are then removed when they conform to rule 1.

These rules ensure that devices will not evict modified replicas until they are safely on some "stable" location (i.e., in a completely created view). The rules also assure that a device will not drop a file until it has confirmed that another up-to-date replica of the file exists somewhere in the system. However, a user can force the system to drop a file replica without ensuring another replica exists, if she is confident that another replica exists and is willing to forgo this system protection. With these rules, Perspective can provide permanence guarantees without requiring central control or limiting when or where views can be changed.

6.4 Implementation

The Perspective prototype is implemented in C++ and runs at user-level using FUSE [20] to connect with the system. It currently runs on both Linux and Macintosh OS X. Perspective stores file data in files in each machine's local filesystem and metadata in a SQLite database with an XML wrapper. While this reference implementation utilizes a particular set of algorithms, the Perspective protocol is designed to allow a range of implementations based on the sophistication and needs of the device.

This section outlines the details of the Perspective filesystem. The first subsection outlines the Perspective protocols, separate from any specific implementation. The second subsection outlines in detail the components of our Perspective prototype. The third subsection outlines the local object

Rpc command	Description
query(*in* query, *out* list<metadata>)	Return the metadata for all objects stored on the device that match query q.
updatePending()	Inform a remote device that an update has occurred on the local device.
pull(*in* oid, *out* metadata, *out* data)	Get the data and metadata for the most recent version of named object stored on this device.
sync(*in* timestamp, *out* list<metadata>)	Return the metadata for all versions of files modified by this device since the passed local timestamp.
objectCommitted(*in* metadata)	After this RPC is called, a device is free to drop the replica if it matches a partial view but not a complete view.

Table 6.5. **Perspective RPC calls.** This table shows the five main Perspective RPC calls. Perspective also contains another one or two RPCs purely to check a device's status, but they are not needed for core system functionality.

store implementation. The fourth subsection outlines crash recovery. The fifth subsection outlines getting and using data from remote devices.

6.4.1 Perspective protocols

There are two major components of the Perspective protocols. The first is the RPC calls, which define the language that devices speak with one another. The second is the Perspective native application API, which allows applications to access information in the Perspective filesystem.

Perspective RPC calls

The Perspective protocol only requires devices to support five main RPC calls. These calls allow devices to find data, manipulate data and views, and synchronize data. The Perspective RPC interface is kept simple to allow both sophisticated and simple implementations to use the same RPC protocol. Figure 6.5 shows a summary of these five RPCs. Below, we provide more detail on the requirements for each of these APIs.

query: A device responds to this RPC by returning the metadata for all files that match the given query. The calling device is responsible for assembling the results and keeping them up to date.

update: This RPC notifies a device that an object has been modified or created. Devices must guarantee that it will eventually call this RPC on

Command	Description
lookup(*in* query, *in* local-only, *out* list<metadata>)	Return the metadata for all objects which match query q.
open(*in* oid, [*in* metadata], *out* handle)	Open the given object. If the object is not stored locally, the optional metadata will improve the performance finding the object on a remote device.
close(*in* handle)	Close the object.
read(*in* handle, *in* offset, *in* size, *out* data)	Read from an object.
write(*in* handle, *in* offset, *in* size, *in* data)	Write to an object.
readMetadata(*in* handle, *out* metadata)	Read object metadata.
writeMetadata(*in* handle, *in* metadata)	Write object metadata.
truncate(*in* handle, *in* size)	Truncate the object.
create(*in* metadata, *out* oid)	Create a new object.
delete(*in* oid)	Delete the given object (removes from all devices).
dropReplica(*in* oid)	Drop the local replica of this object if allowed by the update rules. (Optionally could add a force option).

Table 6.6. **Perspective native API.** This figure lists the native Perspective calls. This API is used by frontends, which convert VFS-like calls into Perspective native calls.

a remote device after a local update to a file matching one of the remote device's views. Upon receiving this RPC, a device should eventually call sync on the sender of the RPC.

pull: A device responds to this RPC by returning the metadata and data for a given object if it stores a version at least as recent as the given metadata, or an error otherwise.

sync: A device responds to this RPC with the metadata of the current version of each file stored on the device that has been modified since the logical timestamp passed in the RPC. If the update log is enabled, this list may only contain files that could be relevant to the calling device.

objectCommitted: This RPC should be called on a remote device after the local device has committed a new version of a file that matches the remote device's views. Upon receiving this RPC, a device may clear the modified tag on the given file if the local version is no more recent than the version passed in the RPC.

Perspective native application API

The Perspective core provides a native API that is semantic and object-based. This API includes standard filesystem operations like open, close,

read and write, in addition to search and metadata operations. Figure 6.6 shows the functions in this API.

Because Perspective is a semantic object store, search is the primary way to locate data. In a traditional filesystem, an application accesses data by first scanning directories to find an inode number and location, which it uses to access a file. In Perspective, an application finds an object through a search on object metadata, which leads to an object ID and location that can be used to access the object.

While applications are free to program directly to the Perspective native API, we expect that most applications will access data through frontends, which map the Perspective native API into other interfaces, such as a standard VFS filesystem via FUSE. Section 7.1 describes the frontends in more detail. The advantage of separating the core filesystem layer from the frontends is the ability for frontends to visualize data (and policies) in different, customized ways, while still providing all of the system properties of availability, redundancy and security on a common, flexible core implementation. Frontends also allow Perspective to expose the more complex semantic data through the conventional VFS interface.

6.4.2 Components

Figure 6.2 shows the major components of Perspective. We have designed Perspective in a modular fashion, to allow alternate implementations (for experimentation purposes) of many of the components.

View manager: The view manager handles all operations on views. It sends update messages to the appropriate other devices when local changes are made and accepts messages from remote devices, handing them off to the object manager for processing. The view manager also decides which objects to return when another device asks for sync information. Search is also performed through the view manager, which passes the search on to the appropriate devices. Overall, the view manager manages the control messages for Perspective by communicating with view managers on remote devices.

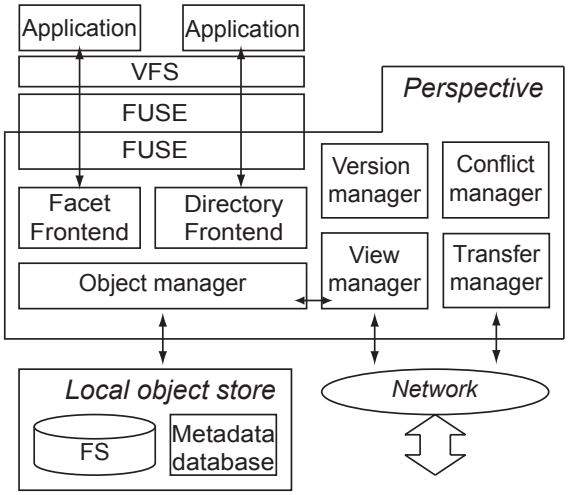

Figure 6.2. **Block diagram.** This block diagram shows the major components of the Perspective.

Transfer manager: The transfer manager is responsible for transferring data between devices. The transfer manager may contain a set of data transfer protocols from which it can choose an appropriate method, depending on the data, the device, and current connectivity.

Version manager: The version manager compares two versions of an object and decides which version is newest. If two versions conflict, it will ask the conflict manager to resolve it. Perspective uses version vectors to track object versions and identify conflicts.

Conflict manager: The conflict manager is responsible for resolving conflicts when the occur in the system. Perspective can use user directed conflict resolution or application level resolvers, just like Coda and Bayou [76, 67].

Object manager: The object manager coordinates updates to objects, applying updates to the local store, passing messages to the frontends and the view manager, and processing front end requests.

Frontend: A frontend is responsible for connecting an application to Perspective. The frontend customizes the way in which it exports objects to an application. A frontend could implement a standard file system interface

Column	Description
oid	The object id of this attribute
name	The name of the attribute
value	The value of the attribute
intvalue	The value converted to an integer for indexing
position	The position of the attribute in the xml doc

Table 6.7. **Database schema.** This table shows the schema of the local store SQL database.

by mapping object attributes into a file system hierarchy, as in the Semantic file system [22]. This allows an application to mount Perspective through a standard file system interface. The frontend can convert from filesystem directory operations into Perspective searches. Alternately, a frontend could communicate directly with an application to customize the interface.

Local object store: The local object store stores object replicas and metadata. We have implemented our own local object store, but any semantic store would suffice.

6.4.3 Local object store

Perspective uses the local filesystem to actually store the files stored on the device. Because our focus does not require innovation in on-disk layout, this allows us to simplify our design. Theoretically, Perspective could use any local filesystem that supports name-value pair metadata, although it is unclear that commodity semantic filesystems would be appropriately tuned to efficiently implement *enumerate values* and *enumerate attributes* queries.

The data for each file is stored in a file in a single directory in the underlying filesystem, named by object ID. If needed, this could be extended into a deeper hierarchy to decrease the size of each individual directory, but this has not been slow enough to warrant optimization yet.

Perspective stores the metadata associated with each file as an XML document, with the restrictions that each tag must be unique and no tag can have child tags, i.e. the XML document is flat.

The metadata is stored in a SQLite database that is also stored in the local filesystem. The SQLite database is surrounded by an XML wrapper

Format	Value
Perspective query	object[owner=Brandon and (sharing=Public or type)]
Disjunctive normal form	object[(owner=Brandon and sharing=Public) or (owner=Brandon and type)]
SQL	select s3.oid, s3.name, s3.value from store s1, store s2, store s3 where s1.name='owner' and s1.value='Brandon and s2.name='sharing' and s2.value='Public' union select s3.oid, s3.name, s3.value from store s1, store s2, store s3 where s1.name='owner' and s1.value='Brandon' and s2.name='type';

Table 6.8. **Perspective query to SQL.** This table shows the conversion of an example Perspective query into disjunctive normal form, and then into a SQL statement.

that converts between XML and SQL. The schema for the database, shown in Table 6.7, allows us to index data without requiring an understanding of the tags found in the system.

In order to convert a query into a SQL statement, we first apply De Morgan's laws and associativity to convert the query into *disjunctive normal form*. This creates a query with groups of clauses combined with the *and* operator, which are in turn combined with the *or* operator. We can then convert this into SQL by creating a join and select operation for each clause in the *and* groupings, and then unioning the results of each group across the *or* operations. Table 6.8 shows an example of the conversion from a Perspective query to SQL.

Enumerate values queries and enumerate attributes queries are implemented using the *distinct* option and clauses on the name and value columns. Table 6.9 shows an example of the SQL for an enumerate values query.

6.4.4 Crash recovery

After a crash, Perspective increments all the essential counters (the one used for oid, the one used for logical timestamps, and the one used for replica ids). To avoid having to keep a detailed write ahead log, Perspective uses the fact that these counters are extremely large (unsigned 64 bit numbers) and increments them by an amount guaranteed to be larger than the number of operations that could be in flight between two sync calls.

Keeping metadata and data in sync: When a device crashes, Perspective also needs to make sure that the metadata and data still match and

Perspective query	SQL
/object[]:album	select distinct 0 as oid, s0.name as name, s0.value as value, from store s0 where s0.name = 'album' union select distinct 0 as oid, s0.name as name, s0.value as value, from store s0 where not exists (select * from store s1 where s1.oid = s0.oid and s1.name = 'album limit 1);
/object[]:*	select distinct 0 as oid, s0.name as name, 0 as value from store s0;

Table 6.9. **Perspective enumerate values query to SQL.** This table shows example enumerate values and enumerate attributes queries converted to SQL.

that the version on the resulting file is not the same as a different version of the file. Perspective's local object store does so by using *commit timestamps* and an fsck on crash recovery. This process could be tuned using a write ahead log, as is done in most contemporary filesystems.

The commit timestamp is a logical timestamp that is guaranteed to increase monotonically, and always be distinguishable from a recent write timestamp. Perspective uses a timestamp years in the future. On each update Perspective will set the timestamp to the maximum of the current time several years in the future, or the previous version of the timestamp+1.

This timestamp is put into the metadata for the object. Changes to the file data will set the mtime to a time in the present, guaranteed not to equal the commit timestamp. On file close, the data file is flushed to disk, the database updates are committed, and then Perspective changes the mtime of the file to match the commit timestamp.

On crash, Perspective does an fsck of the drive. For each file, it checks the commit timestamp in the metadata against the mtime of the file. If the two do not match, it changes the replica id of the file to a new replica id and increments the version of the file.

This ensures that, if the original update was transferred to another device before the crash, this version will conflict with the other version, allowing

the user to decide what version to use. If the update was not copied to another device, then this new version will supercede the old version, just as would happen with a local filesystem.

The object store also has a *set* operation that acts like a rename in the oid space. This operation can also leave files with data not matching metadata. However, we do not want to make these into a new version of the object, since the interrupted operation was not actually modifying data in any way. To distinguish between set operations and local updates, the set operator uses a timestamp much farther in the future. Fsck throws any mismatching files with this timestamp out. Since we could only get an interrupted set operation if the update has not yet been committed to the store, we can be sure another version of the object exists on some other device in the system, due to the update rules described in Section 6.3. So we will pull the new version of this object again when we see it.

6.4.5 Remote data access

While devices in Perspective can access the data of a file whether it is stored locally or remotely, internally all data access in Perspective is local. In order to access a remote file, Perspective will pull a local copy of the file, and then treat all accesses as local accesses. The update rules described in Section 6.3 make sure that the local copy will be dropped when possible.

Pulling updates: When Perspective is asked to pull an update from a remote device, it first checks all devices that could contain the object, starting with the device that the current metadata came from, then trying all devices with matching views, then trying all devices currently accessible.

Once it has located a device with an appropriate copy of the object, it uses the pull RPC to get the data and metadata for the object. The update is put into a temporary object with a special low oid that is ignored by queries. If the remote object is modified during the download, it will restart. Once the update has completely downloaded, Perspective checks to make sure it is still more recent than the local replica, uses the set operation to rename the temporary object over any existing object with the given oid.

6.5 Accessing and moving data

This section outlines the execution of a number of common operations, as a way to review the various functions in Perspective. Applications access data through a frontend mounted in the local VFS filesystem. The frontend converts the VFS calls into the appropriate Perspective native calls and relays the results back to the user.

Readdir: When an application does a readdir in a directory, the front end converts the path of the directory into a Perspective query and forwards the search request to the object manager. The object manager then asks the view manager to forward the search to remote devices. The view manager checks with the system views and forwards the operation to the appropriate devices, possibly including the local device. These searches return the metadata for matching objects, which the view manager combines. This list of object metadata is passed back to the frontend, which then converts these metadata objects into filenames for the directory.

Create: When an application creates a new object, the front end converts the path to the file into a query and then uses this query to create a metadata object with the appropriate attributes set. It then passes the metadata for the object into the object manager. The object manager assigns a new object ID and creates the object in the local object store. The object manager returns the new object ID to the frontend for reference. It also forwards the update to the view manager to forward to remote devices with matching views.

Open: To open an object in Perspective, the frontend first looks up the metadata for the object by converting the path to the object into a search, just like a readdir request. It passes this metadata to the object manager. The object manager extracts the object ID from the metadata. The object manager then generates a new object manager handle which it attaches to this object id. Note that the open operation will not pull a replica of a remotely stored object. The *get replica* operation will be called to get a remotely stored replica, when it is needed for later operations.

Read: To read an object, Perspective ensures that a copy of the object

is stored locally and open using the *get replica* command and then simply reads data from this file.

Get replica: The object manager handle is associated with an object id, which allows the object manager to index into the local object store and look for the object. If the object is not found locally, the object manager uses the metadata and views to query for a replica of the object on a another device. It creates a local copy of the file which will be used until it is closed. The object manager then opens the local replica and ties this handle to the object manager handle.

Write: First, the object manager ensures that a local replica is open using *get replica*. The object manager asks the version manager to update the version vector for the given object. The object manager then writes the changed data and metadata through to the local object store and asks the view manager to send the update to the appropriate devices.

Read metadata: To read the metadata of an object, an application calls getxattr on a special Perspective attribute. To satisfy the request, the frontend looks up the metadata for the file, using the cached query associated with the directory in which the file is stored.

Write metadata: First, the object manager ensures that a local replica is open using *get replica*. The object manager asks the version manager to update the version vector for the given object. The object manager then writes the changed metadata to the local object store and asks the view manager to send the update to the appropriate devices.

Delete: The delete operation truncates the file to zero length and sets a special "deleted" attribute in the metadata. The metadata then serves as a *tombstone* that marks it as deleted and hides it from normal queries [28]. It then forwards the update to the view manager to forward to the appropriate devices.

Forwarding updates: Any local modification to an object must eventually be passed to remote devices with matching views. Each time a local file is modified, the view manager adds an entry for each device with views that match the file before or after the update to a list. Occasionally, the view

manager sends a *pendingUpdates* RPC to each device on the list, notifying the device to sync with the local device.

7 Insight: view-based management tools

Because I focused on usability, it was not sufficient to implement a filesystem; I also had to implement a set of tools to allow users to manipulate the filesystem. Insight is a set of management tools written to utilize the Perspective filesystem. Insight includes a set of frontends linked into Perspective that allow devices to customize the way data is presented to applications or users. Insight also includes a java toolkit containing core algorithms for applications that manipulate views. Insight also includes interfaces for manipulating views and object metadata.

7.1 Frontends

Perspective provides a semantic interface, but it is important to map this semantic interface through the standard VFS layer to allow commodity applications to access the data. Frontends allow applications to map the semantic naming of Perspective into other naming schemes, such as a hierarchical naming scheme. This section describes the frontends I have implemented in detail. Each query in Perspective has a *base query* in the native Perspective query language, and then a *frontend* defines how the files matching this query are displayed. Each frontend is a small C++ module which is linked with Perspective. Each module implements a way to convert from a VFS path into Perspective queries and implements name operations like rename, mkdir, rmdir, etc.

I implemented four frontends, each of which fills a different purpose. I expect that developers who work in Perspective will build other frontends to accommodate their needs as well.

7.1.1 Customizable faceted metadata frontend

Most naming in Perspective is done through a frontend called the *customizable faceted metadata* frontend.

One way of visualizing and accessing semantic data is through the use of *faceted metadata* [84]. Faceted metadata allows a user to choose a first attribute to use to divide the data and a value at which to divide. Then, the user can choose another attribute to divide on, and so on. Faceted metadata helps users browse semantic information by giving them the flexibility to divide the data as needed. But, it can present the user with a dizzying array of choices in environments with large numbers of attributes.

To curb this problem, I developed *customizable faceted metadata*, which exposes a small user-selected set of attributes as directories plus one additional *other groupings* directory that contains a full list of possible attributes. The user can customize which attributes are displayed in the original list by moving folders between the base directory and the *other groupings* directory. These preferences are saved in a customization object in the filesystem. By performing these customizations, the user can effectively build a set of hierarchies, while still having the advantage of being able to divide their data in the manner most convenient. Figure 7.1 illustrates customizable faceted metadata as browsed from the Finder.

Customizable faceted metadata provides three different types of directories. *Attribute* directories show a list of possible attributes and are translated into *enumerate attribute* queries. *Value* directories show the possible values of a particular attribute and are translated into *enumerate values* queries. *File* directories show the files in a particular group as a flat list and are translated into normal queries. Table 7.1 shows the translation of several paths into Perspective queries.

Challenges in accommodating applications: While it is conceptually fairly simple to expose faceted metadata through the VFS layer, some of the details of exposing faceted metadata over VFS to unmodified applications have been very tricky to get right. This section outlines some of these challenges.

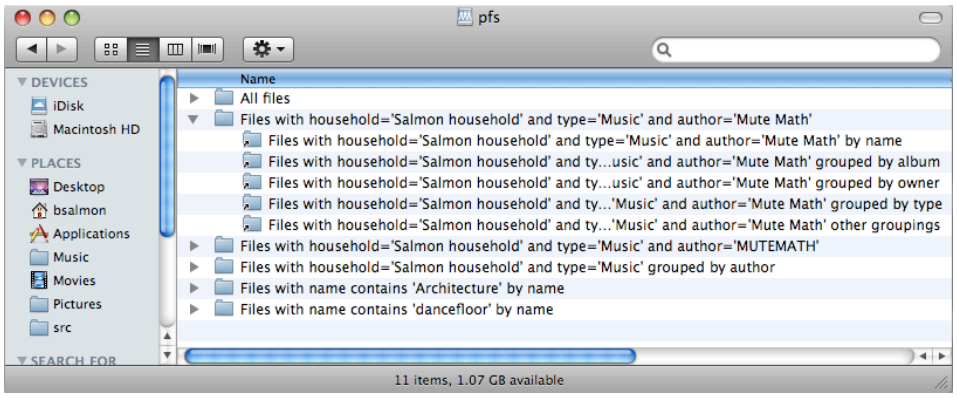

Figure 7.1. **Customizable faceted metadata frontend.** This figure shows a screenshot of customizable faceted metadata through the Finder.

Path	Perspective query
object[]facet/All files/All files grouped by owner/Files with owner='Bob'	object[owner='Bob']:*
object[]facet/All files/All files grouped by owner/Files with owner='Bob' /Files with owner='Bob' grouped by type	object[owner='Bob']:type
object[]facet/All files/All files grouped by owner/Files with owner='Bob' /Files with owner='Bob' by name	object[owner='Bob']

Table 7.1. **Mapping customizable faceted metadata to Perspective queries.** This table shows the way a number of paths in customizable faceted metadata are converted into Perspective queries.

Source	Destination
object[]facet/All files/All files grouped by owner/ Files with owner='Bob'	object[]facet/Files with owner='Bob'
object[]facet/All files/All files grouped by album/ Files with album='Pop'	object[]facet/Files with album='Pop'

Table 7.2. **Symbolic links in faceted metadata.** This table shows the destination of two example symlinks in the facet frontend.

Enumeration: One of the challenges is dealing with enumeration. Many applications attempt to recurse through file subtrees and analyze all files. While this works well in a standard files-and-folders system, the flexibility of faceted metadata makes it effectively an infinitely deep tree, with the same file appearing in multiple paths.

Full recursion of subtrees happens surprisingly frequently. For example, the OS X finder actually attempts to fully recurse a subtree any time a user attempts to move, remove or modify a file subtree.

To address this challenge, we expose all faceted directories as symbolic links rather than actual directories. Each link points to a directory in the root of the volume that contains the full path. Table 7.2 describes the source and destination of several example links. Well-behaved applications do not traverse symbolic links when doing recursion to avoid loops, which means they will not try to enumerate the full subtree. The sort frontend, described in the next section, allows applications to enumerate files when needed.

Stable paths: Applications also require stable file paths. If they access a file using a particular faceted path, they expect that path to continue to be valid, even if users in general use a different type of path. For this reason, we return success when calling getattr on a faceted sub-directory, even if it isn't in the commonly used set of attributes. Symbolic links also help this process, as they collapse a path into a more concise query.

Mkdir, rmdir: The mkdir and rmdir operations are also challenging to do correctly. For example, most finders create a new directory with a default name. This is a challenge for faceted metadata, as each directory must have a specific format. To accommodate these applications, Perspective allows an application to create a directory even if it doesn't match the query format.

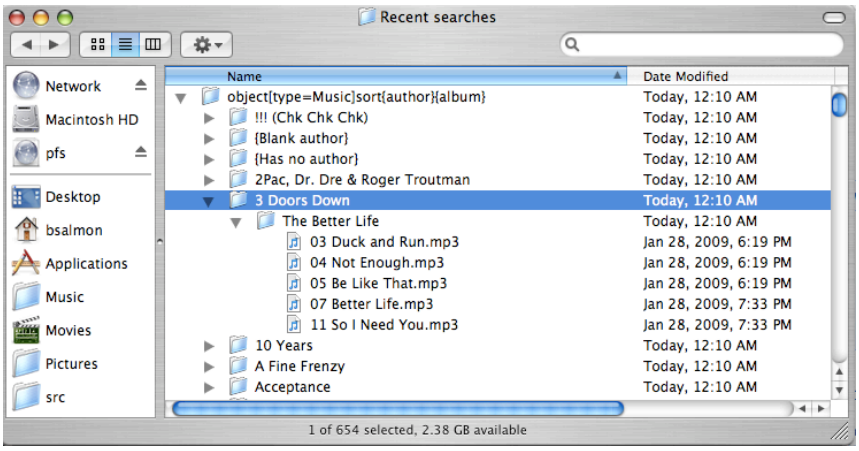

Figure 7.2. **Sort frontend.** This figure shows the sort frontend through the finder. Note that this sort query sorts based on author and album.

If it does not match this format, the faceted frontend will convert it into the correct format, allow the mkdir to succeed, and then begin displaying the symbolic link for the correct name.

7.1.2 Sort frontend

The sort frontend allows an application to choose a specific set of attributes to use to sort a given set of files. This allows applications to enumerate a set of files too large to deal with in a flat directory. The configuration string for this frontend lists a set of attributes. The first level of the hierarchy will be the values of the first attribute, the second level of the hierarchy the second attribute, etc. This allows a music program, for example, to divide the results of a query by *artist* and then by *album*.

To convert a sort frontend path to a Perspective query, the frontend determines if the given directory is of the depth of the number of attributes. If it is less than the number of attributes, then it uses an *enumerate values* query to find all values of the appropriate attribute. If the tree depth is the number of attributes, it uses a normal query to list the given files. Table 7.3 shows the translation of several paths into Perspective queries.

Path	Perspective query
object[]sort{artist}{album}	object[]:artist
object[]sort{artist}{album}/U2	object[artist='U2']:album
object[]sort{artist}{album}/U2/Pop	object[artist='U2' and album='Pop']

Table 7.3. **Mapping sort to Perspective queries.** This table shows the conversion from three sort paths into native Perspective queries.

7.1.3 Directory frontend

It is also necessary to provide some applications with an actual files-and-folders visualization of their data. The directory frontend provides this visualization and stores the directory structure in metadata tags.

This frontend is implemented by setting a special set of tags. Each file is tagged with an attribute for each level of the path, along with a depth in the path. Empty directories are represented by files with the correct attributes and a special name (“.”). To search for all files in a directory, the frontend creates a Perspective search for all files with the correct path. The frontend also searches for all subdirectories, by looking for all values of the attribute next in the path in files inside the current subtree. Table 7.4 shows an example of the conversion between a path in the hierarchy and Perspective native queries.

Note that this frontend actually implements a *union* style system. A subtree will exist if any file on any machine exists in that subtree. This makes it easy to construct the file tree as devices enter and leave the ensemble.

Applications that require a file hierarchy should also set appropriate tags to allow users to browse the data with faceted metadata. For example, iTunes actually writes new files into a directory tree, but scripting makes sure that files are also tagged with author, album, etc.

7.1.4 Search frontend

The simplest frontend is the search frontend. This frontend simply displays all files that match the base query in a flat list of files. This makes it easy for users to do individual searches in the filesystem, but may become intractable

Path	Perspective query
object[]dir/Music	object[pathDepth=2 and path1=Music] + object[pathDepth>2 and path1=Music]:path2

Table 7.4. **Mapping directories to Perspective queries.** This table shows how an example directory frontend path is converted into native Perspective queries.

Path	Perspective query
object[name contains 'Love']	object[name contains 'Love']
object[type=Music and size < 1000]/foo.mp3	object[name='foo.mp3' and type=Music and size < 1000]

Table 7.5. **Mapping search frontend paths into native Perspective queries.** This table shows how two example search frontend paths are converted into native Perspective queries.

when dealing with large numbers of files. This frontend is the default frontend applied to raw Perspective queries. Table 7.5 shows example conversions of search frontend paths into native Perspective queries.

7.2 Libperspective

Perspective also provides a library for applications that wish to manage data in Perspective. This library contains a C++ component linked with a set of Java libraries. The library contains methods to manipulate metadata on individual files, methods to manipulate views and devices, and methods to reason about the redundancy of data.

7.2.1 Manipulating file metadata

The metadata in files is accessible to applications using the *getxattr* and *setxattr* commands. Currently, Perspective exports all of a file's attributes as the text of an XML document in a single extended attribute with a special name ("perspective_mdata"). However, it would also be possible to expose each individual attribute through a separate extended attribute name. Since many versions of Java do not include extended attributes, libperspective contains functions to read and write metadata from files for Java applications.

Attribute name	Description
viewObject	If this tag is present, the file is a view.
name	Name of the view, constructed from other tags
query	The view query
deviceId	The id of the device this view is attached to.
viewType	The type of view (partial, complete, application)

Table 7.6. **View attributes.** This table shows the attributes in a view.

Attribute name	Description
deviceObject	If this tag is present, the file is a device.
name	Name of the device.
id	The id of the device.
owner	The owner of the device.
household	The household of the device.
capacity	The total capacity of the device.
availableCapacity	The free space on the device.

Table 7.7. **Device attributes.** This table shows the attributes in a device.

7.2.2 Manipulating views and devices

Views and devices show up as special files in the filesystem. The important information about each device or view is contained in the metadata for the corresponding file. Table 7.6 shows the attributes of views, and Table 7.7 shows the attributes of devices. Libperspective contains methods to create, delete and modify views, and methods to delete device objects after a device fails. The methods simply manipulate the corresponding files and their metadata.

7.2.3 Application view framework

The application view framework simplifies the use of application views to keep applications in sync with Perspective. Section 6.1.6 describes application views in detail. This section describes an application framework utilizing the feature. The framework includes a main java base class for an application and a central class that runs the framework for all installed applications. Each application only needs to implement a subclass of the base class.

The framework installs a cron job that executes every few seconds. It reads from a config file to discover the installed applications and the application views for each application. It then reads all of the outstanding updates for the application view and calls a subclass specific to the application to handle each update.

7.2.4 Reasoning about file replicas with overlap trees

Reasoning about the reliability of a storage system — put simply, determining the level of replication for each data item — is a challenge in a partially-replicated filesystem. Since devices can store arbitrary subsets of the data, there are no simple rules that allow all of the replicas to be counted. Libperspective provides an *overlap tree* library to facilitate reasoning about the number of replicas in the system.

A naïve solution would be to enumerate all of the files on each device and count replicas. Unfortunately, this would be prohibitively expensive and would be possible only if all devices are currently accessible. Fortunately, Perspective's views compactly and fully describe the location of files in terms of their attributes. Since there are far fewer views than there are file replicas in the system, it is cheaper to reason about the number of times a particular query is replicated among all of the views in the system than to enumerate all replicas. The files in question could be replicated exactly (e.g., all of the family's pictures are on two devices), they could be subsumed by multiple views (e.g., all files are on the desktop and all pictures are on the laptop), or they could be replicated in part on multiple devices but never in full on any one device (e.g., Alice's pictures are on her laptop and desktop, while Bob's pictures are on his laptop and desktop – among all devices, the entire family's pictures have two replicas).

To efficiently reason about how views overlap, Perspective uses *overlap trees.* An overlap tree encapsulates the location of general subsets of data in the system and, thus, simplifies the task of determining the location of the many data groupings needed by management tools. An overlap tree is

currently created each time a management application starts and then used throughout the application's runtime to answer needed overlap queries.

Overlap trees are created using enumeration queries. Each node contains a query that describes the set of data that the node represents. Each leaf node represents a subset of files whose location can be precisely quantified using the views and records the devices that store that subset. Each interior node of the tree encodes a subdivision of the attribute space and contains a list of child nodes, each of which represents a subset of the files that the parent node represents. We begin building the tree by enumerating all of the attributes that are used in the views found in the household.

We create a root node for the tree to represent all files, choose the first attribute in our attribute list, and use the enumerate values query to find all values of this attribute for the current node's query. We then create a child node from each value with a query of the form *<parent query> and attribute=value*. We compare the query for each child node against the complete views on all devices. If the compare operator can give a precise answer (i.e., not *unknown*) for whether or not the query for this node is stored on each device in the home, then this node is a leaf and we can stop dividing. Otherwise, we recursively perform this operation on the child node, dividing it by the next attribute in our list. Figure 7.3 shows an example overlap tree. The ordering of the attribute list could be optimized to improve performance of the overlap tree, but I leave it unordered in the current implementation.

When we create an overlap tree, we may not have all information needed to construct the tree. For example, if we currently only have access to Brian's files, we may incorrectly assume that all music files in the household are owned by Brian, when music files owned by Mary exist elsewhere in the system. The tree construction mechanism makes a notation in a node if it cannot guarantee that all matching files are available via the views. When checking for overlaps, if a marked node is required, the tree will return an *unknown* value, but it will still correctly compute overlaps for queries that do not require these nodes. To avoid this restriction, devices are free to cache and update an overlap tree, rather than recreating the overlap tree when

each management application starts. The tree is small, making caching it easy. To keep it up to date, a device can publish a view for all files and then use the updates to keep the cached tree up to date.

Once we have constructed the overlap tree, we can use it to determine the location and number of full copies in the system of the files for any given query. Because the tree caches much of the overlap processing, each individual query request can be processed efficiently. We do so by traversing all of the leaf nodes and finding those that overlap with the given view or query. We may occasionally need to perform more costly overlap detection, if the attribute in a leaf node does not match any of the attributes in the query. For example, in the overlap tree in Figure 7.3, if we were checking to see if the query *album=Joshua Tree* was contained in the node *owner=Mary and type=Music*, we would use the enumerate values query to determine the values of "type" for the query *album=Joshua Tree and owner=Mary*. If "Music" is the only value, then we can count this node as a full match in our computations. Otherwise, we cannot. This extra comparison is only valid if we can determine; via the views, that all files in the query for which we are computing overlaps are accessible

Attributes with larger cardinalities can be handled more efficiently by selectively expanding the tree. For example, if a view is defined on a query such as $date < T$, we need only expand the attribute date into three sub-nodes, one for $date < T$, one for $date \geq T$, and one for *has no date attribute*.

Note that the number of attributes used in views at any one time is likely to be much smaller than the total number of attributes in the system, and both of these will be much smaller than the total number of files or replicas. For example, in our contextual analysis, most households described settings requiring around 20 views and 5 attributes for views. None of households we interviewed described more than 30 views or more than 7 attributes for views. Because the number of relevant attributes is small, overlap tree computations are fast enough to allow us to compute them in real time as the user browses files. We will present a performance comparison of overlap trees to the naïve approach in Section 8.

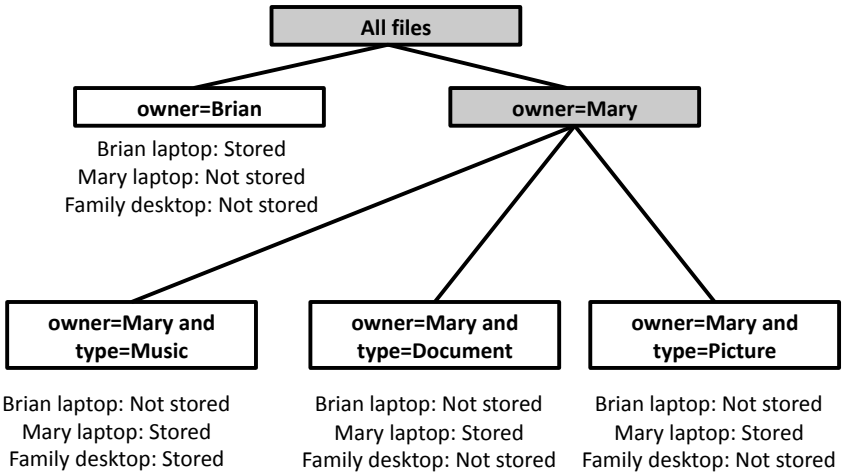

Figure 7.3. **Overlap tree.** This figure shows an example overlap tree, constructed from a three-device, three-view scenerio: Brian's files stored on Brian's laptop, Mary's files stored on Mary's laptop, and Mary's music stored on the Family desktop. Shaded nodes are interior nodes and unshaded nodes are leaf nodes. Each leaf node lists whether or not this query is stored on each device the household.

7.3 Interfaces

Insight also includes several interfaces for manipulating metadata and views. The viewmanager interface allows users to create and modify views, and the pchatr interface allows users to view and modify metadata assigned to files.

7.3.1 View manager interface

To explore view-based management, we built a *view manager* tool to allow users to manipulate views. The view manager interface (Figure 8.1) allows users to create and delete views on devices and to see the effects of these actions. This GUI is built in Java and makes calls into the view library of the underlying filesystem.

The GUI is built on Expandable Grids [62], a user interface concept initially developed to allow users to view and edit file system permissions. The interface significantly simplified these tasks by allowing users to view and edit effective permissions, rather than requiring the user to manipulate and understand a set of rules. We apply Expandable Grids to storage policies in our system. Each row in the grid represents a file or file group, and each column represents a device in the household. The color of a square represents whether or not the files in the row are stored on the device in the column. The files can be "all stored" on the device, "some stored" on the device, or "not stored" on the device. Each option is represented by a different color in the square. By clicking on a square a user can add or remove the given files from the given device. Similarly to file permissions, this allows users to manipulate actual storage decisions, instead of rule lists.

An extra column, labeled "Summary of failure protection," shows whether the given set of files is protected from one failure or not, which is true if there are at least two copies of each file in the set. By clicking on an unbacked-up square, the user can ask the system to ensure that two copies of the files are stored in the system, which it will do by placing any extra needed replicas on devices with free space.

An extra row contains all unique views and where they are stored, allowing a user to see precisely what data is stored on each device at a glance.

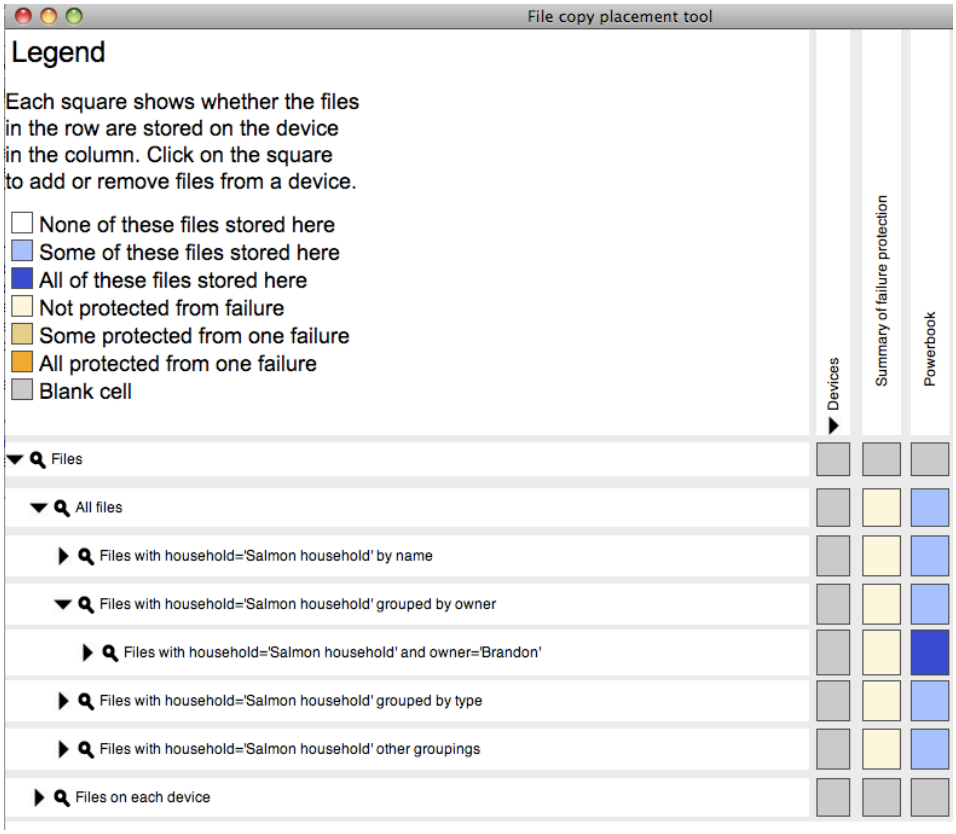

Figure 7.4. **View manager interface.** A screen shot of the view manager GUI. On the left are files, grouped using faceted metadata. Across the top are devices. Each square shows whether the files in the row are stored on the device in the column.

7.3.2 Pchatr interface

The pchatr interface allows users to view and modify the metadata associated with a file. It can be run in a GUI mode or in a command line mode.

To run the GUI mode, the user passes the program a list of files or directories to parse. The tool finds all of the common metadata attributes and displays them in a GUI format. The user can add attributes, remove attributes, or modify attributes with the GUI. When finished, the user pushes the "OK" button, and the tool applies the changes to all of the files. If the user pushes "Cancel," the updates are aborted.

The command line mode takes a list of files and a query to use to set attributes on these files. For example, to add the tags, *owner=Bob* and *type=Music*, the user would pass through the query, *object[owner=Bob and type=Music*. The tool then applies the query to each file in the set.

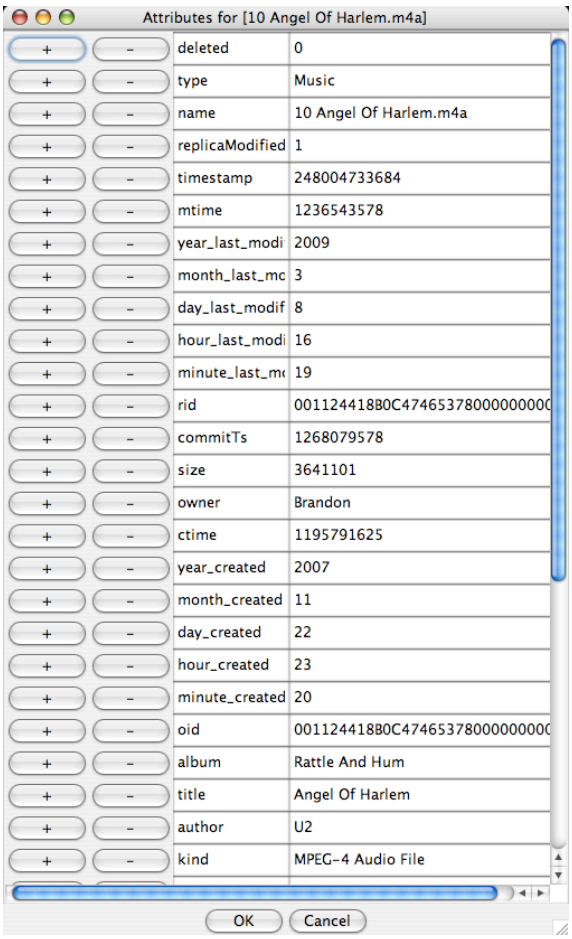

Figure 7.5. **Pchatr interface.** This figure shows the pchatr interface for manipulating file metdata.

8 Evaluation

My experience from working with many home storage users suggests that users are very concerned about the time and effort spent managing their devices and data at home, which has motivated both my design of Perspective and my evaluation. Therefore, I focus my study primarily on the usability of Perspective's management capabilities and secondarily on its performance overhead.

To address the usability of the system, I performed an in-lab user study to test how views impact management tasks, described in Section 8.1. I also started a long-term deployment of Perspective in two households, described in Section 8.2.

To show the feasibility of such a system, I performed several high-level benchmarks to evaluate performance overhead, described in Section 8.3. To illustrate some of the tradeoffs involved in Perspective's design decisions, I did several micro-benchmarks and qualitative comparisons of approaches, described in Section 8.4.

8.1 Usability lab study

I conducted a lab study in which non-technical users used Perspective, outfitted with appropriate user interfaces, to perform home data management tasks. I measured accuracy and completion time of each task. In order to insulate my results as much as possible from the particulars of the user interface used for each primitive, I built similar user interfaces for each primitive using the Expandable Grids UI toolkit [62].

Views-facet interface: The views-facet interface was described in Section 7.3.1. It uses customizable faceted metadata, described in Section 7.1.1, to describe data, and allows users to place any set of data described by the faceted metadata on any device in the home. Figure **??** shows the interface.

Volumes interface: This user interface is similar, but built on top of a more conventional volume-based system with directory hierarchies. Each device is classified as a client or server, and this distinction is listed in the column along with the device name. The volumes abstraction only allows permanent copies of data to be placed on servers, and it restricts server placement policies on volume boundaries. I defined each root level directory (based on user) as a volume. The abstraction allows placement of a copy of any subtree of the data on any client device, but these replicas are only temporary caches and are not guaranteed to be permanent or complete. The interface distinguishes between temporary and permanent replicas by color. The legend displays a summary of the rules for servers and permanent data and for clients and temporary data. Figure 8.2 shows a screenshot of the volume manager.

Views-directory interface: To tease apart the effects of semantic naming and using a single replica class, I evaluated an intermediate interface, which replaces the customizable faceted metadata organization with a traditional directory hierarchy. Otherwise, it is identical to the views-facet interface. In particular, it allows users to place any subtree of the hierarchy on any device. Figure 8.3 shows a screenshot of the views-directory interface.

8.1.1 Experiment design

My user pool consisted of students and staff from nearby universities in non-technical fields who stated that they did not use their computers for programming. I did a *between-group* comparison, with each participant using one of the three interfaces described above. I tested 10 users in each group, for a total of 30 users overall. The users performed a *think-aloud* study in which they spoke out loud about their current thoughts and read out loud any text they read on the screen, which provides insight into the difficulty

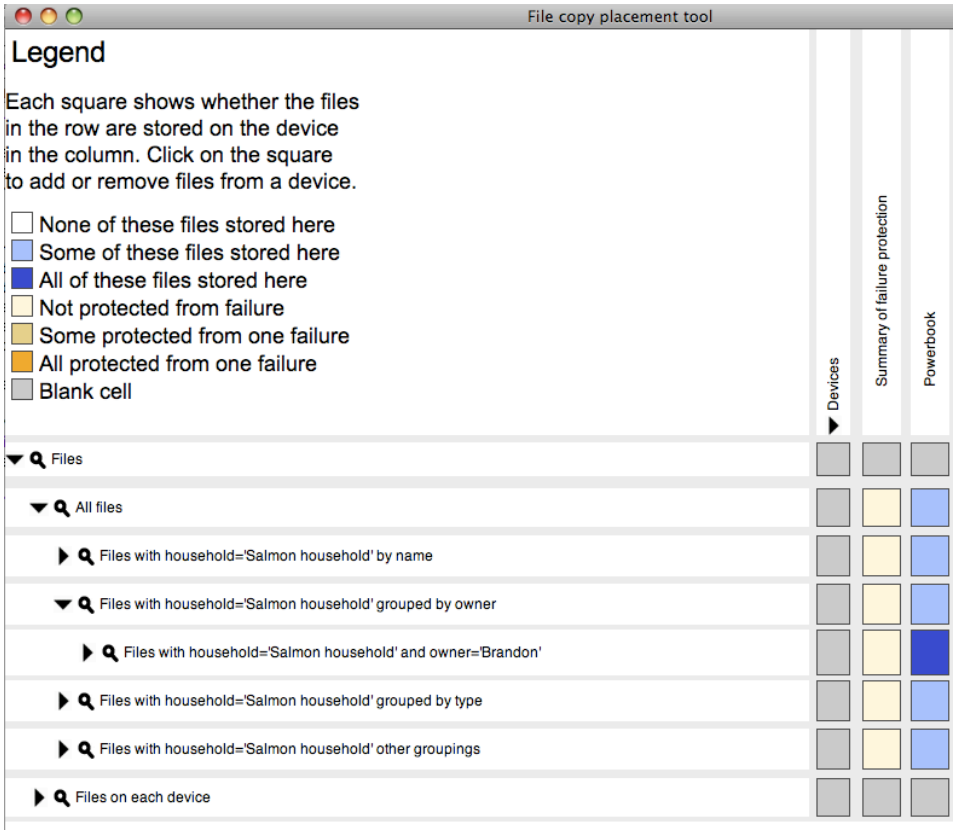

Figure 8.1. **View manager interface.** A screen shot of the view manager GUI. On the left are files, grouped using faceted metadata. Across the top are devices. Each square shows whether the files in the row are stored on the device in the column.

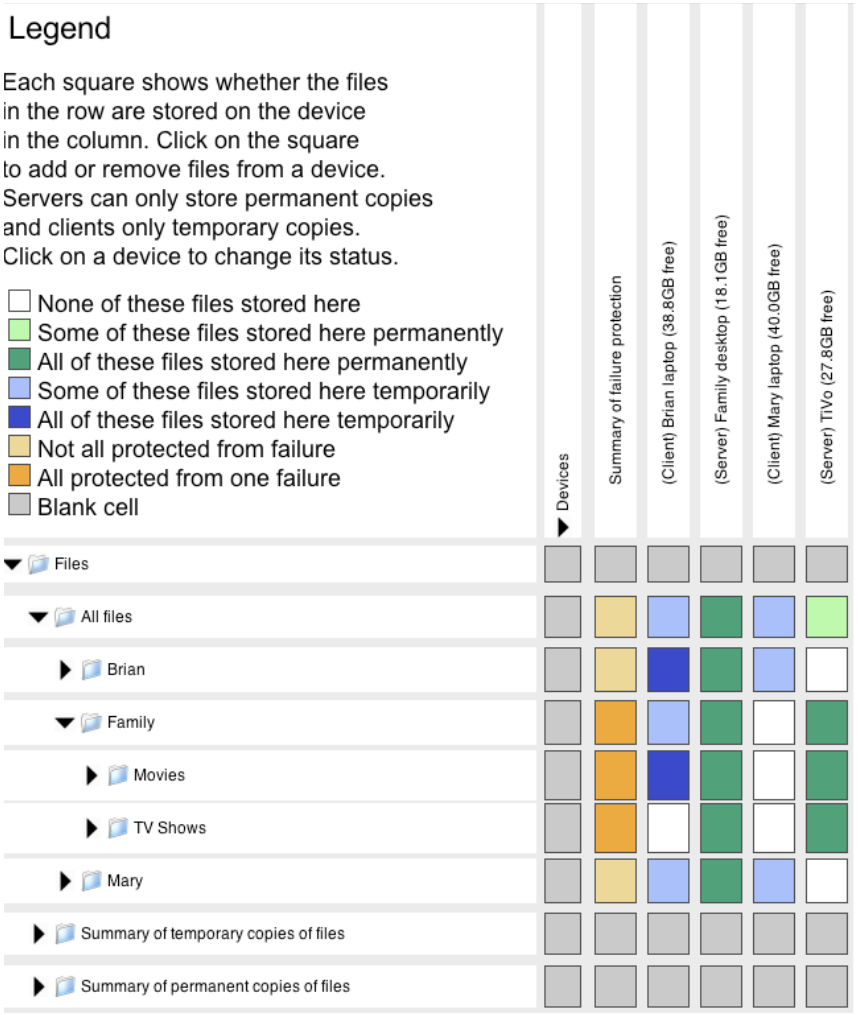

Figure 8.2. **Volumes and caching interface.** A screen shot of the volume manager GUI. On the left are files, grouped using traditional files-and-folders. Across the top are devices. Each square shows whether the files in the row are stored on the device in the column and whether the copy is temporary or permanent.

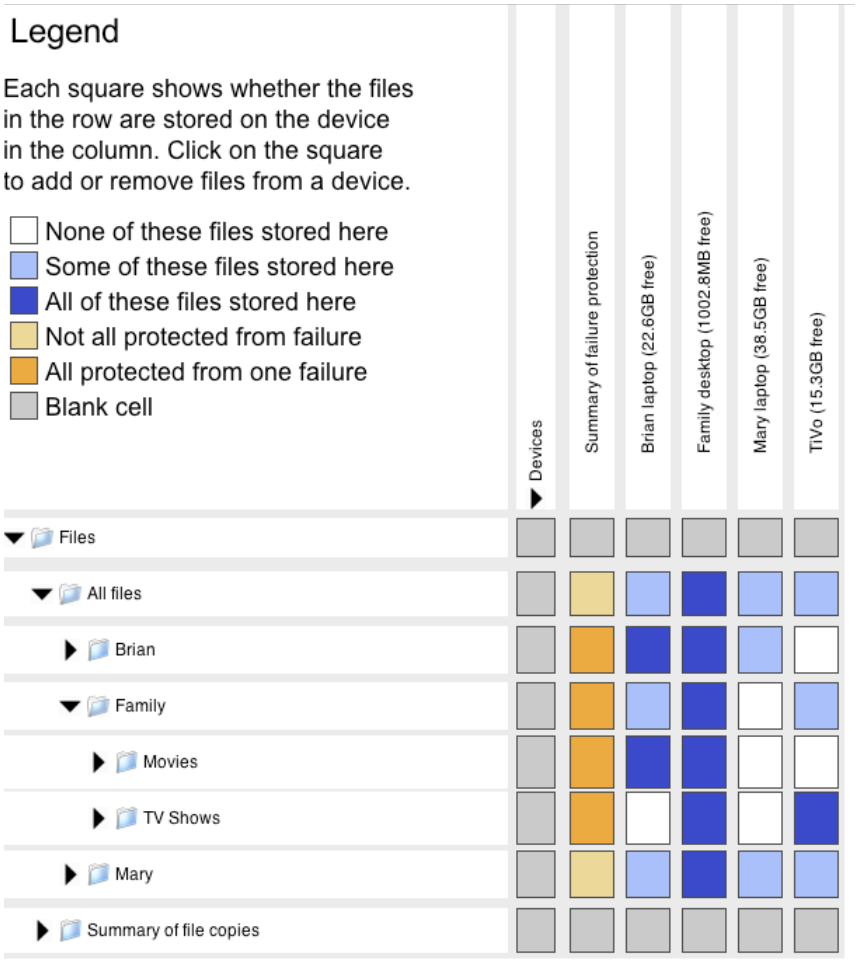

Figure 8.3. **Directory interface.** A screen shot of the directory manager GUI. On the left are files, grouped using traditional files-and-folders. Across the top are devices. Each square shows whether the files in the row are stored on the device in the column.

of tasks and users' interpretation. All tasks were performed in a *latin square* configuration, which guarantees that every task occurs in each position in the ordering and that each task is equally likely to follow any other task.

I created a filesystem with just over 3,000 files, based on observations from my contextual analysis. I created a setup with two hypothetical users, Mary and Brian, and a third "Family" user with some shared files. I modeled Brian's file layout on the Windows music and pictures tools and Mary's on Apple's iTunes and iPhoto file trees. My setup included four devices: two laptops, a desktop, and a DVR. I also provided the user with iTunes and iPhoto, with the libraries filled with all of the matching data from the filesystem. This allowed us to evaluate how users convert from structures in the applications to the underlying filesystem.

8.1.2 Tasks

Each participant performed the same set of tasks, which I designed based on my contextual analysis. I started each user with a 5 to 10 minute training task, after which my participants performed 10 data management tasks. As I discuss each class of tasks, I include the text of one example task. The full text of all the tasks can be found in Appendix B. For this study, I chose tasks to illustrate the differences between the approaches. A base-case task that was similar in all interfaces confirmed that, on such tasks, all interfaces performed similarly. The tasks were divided into two classes: single replica tasks and data organization tasks. The data organization tasks can further be divided into three classes: aggregation, comprehension and sparse collection.

Single replica tasks: Two single replica tasks (LH and CB) required the user to deal with distinctions between permanent and temporary replicas to be successful.

Example task, Mary's laptop comes home (LH): *"Mary has not taken her laptop on a trip with her for a while now, so she has decided to leave it in the house and make an extra copy of her files on it, in case the Family desktop fails. However, Brian has asked her not to make extra copies of his*

files or of the Family files. Make sure Mary's files are safely stored on her laptop."

Mary's laptop was initially a client in the volume case. This task asked the user to change it to a server before storing data there. This step was not required for the single replica class interfaces, as all devices are equivalent.

Note that because server/client systems, unlike Perspective, are designed around non-portable servers for simplicity, it is not feasible to simply make all devices servers. Indeed, my volume interface actually makes this task much simpler than current systems; in the volume interface, I allow the user to switch a device from server to client using a single menu option, where current distributed filesystems require an offline device reformat.

Data organization tasks: The data organization tasks required users to convert from structures in the iTunes and iPhoto applications into the appropriate structures in the filesystem. This allowed us to test differences between a hierarchical and semantic, faceted systems. The data organization tasks are divided into three sub-types: aggregation, comprehension, and sparse collection tasks.

Aggregation One major difference between semantic and hierarchical systems is that, because the hierarchy forces a single tree, tasks that do not match the current tree require the user to aggregate data from multiple directories. This is a natural case as homes fill with aggregation devices and data is shared across users and devices. However, in a hierarchical system, it is difficult for users to know all folders that correspond to a given application grouping. Users often erroneously assumed that all files for a given collection were in the same folder. The semantic structure mitigates this problem, since the user is free to use a filesystem grouping suited to the current specific task.

Example task, U2 (U2): *"Mary and Brian share music at home. However, when Mary is on trips, she finds that she can't listen to all the songs by U2 on her laptop. She doesn't listen to any other music and doesn't want other songs taking up space on her laptop, but she does want to be able to listen to U2. Make sure she can listen to all music by the artist U2 on her trips."*

As may often be the case in the home, the U2 files were spread across all three user's trees in the hierarchical interfaces. The user needed to use iTunes to locate the various folders. The semantic system allowed the user to view all U2 files in a single grouping.

Aggregation is also needed when applications sort data differently from what is needed for the current task. For example, iPhoto places modified photos in a separate folder tree from originals, making it tricky for users to get all files for a particular event. The semantic structure allows applications to set and use attributes, while allowing the user to group data as desired.

Example task, Rafting (RF): *"Mary and Brian went on a rafting trip and took a number of photos, which Mary remembers they labeled as 'Rafting 2007'. She wants to show her mother these photos on Mary's laptop. However, she doesn't want to take up space on her laptop for files other than the 'Rafting 2007' files. Make sure Mary can show the photos to her mother during her visit."*

The rafting photos were initially in Brian's files, but iPhoto places modified copies of photos in a separate directory in the iPhoto tree. To find both folders, the user needed to explore the group in iPhoto. The semantic system allows iPhoto to make the distinction, while allowing the user to group all files from this roll in one grouping.

Comprehension Applications can allow users to set policies on application groupings, and then convert them into the underlying hierarchy. However, in addition to requiring multiple implementations and methods for the same system tasks, this leads to extremely messy underlying policies, which make it difficult for users to understand, especially when viewing it from another application. In contrast, semantic systems can retain a description of the policy as specified by the application, making them easier for users to understand.

Example task, Traveling Brian (TB): *"Brian is taking a trip with his laptop. What data will he be able to access while on his trip? You should summarize your answer into two classes of data."*

Brian's laptop contained all of his files and all of the music files in the household. However, because iTunes places TV shows in the Music repos-

itory, the settings included all of the music subfolders, but not the "TV Shows" subfolder, causing confusion. In contrast, the semantic system allows the user to specify both of these policies in a single view, while still allowing applications to sort the data as needed.

Note that this particular task would be simpler if iTunes chose to sort its files differently, but the current iTunes organization is critical for other administrative tasks, such as backing up a user's full iTunes library. It is impossible to find a single hierarchical grouping that will be suited to all needed operations. This task illustrates how these kinds of mismatches occur even for common tasks and well-behaved applications.

Sparse collection Two sparse collection tasks (BF and HV) required users to make policies on collections that contain single files from across the tree, such as song playlists. These structures do not lend themselves well to a hierarchical structure, so they are kept externally in application structures, forcing users to re-create these policies by hand. In contrast, semantic structures allow applications to push these groupings into the filesystem.

Example task, Brian favorites (BF): *"Brian is taking a trip with his laptop. He doesn't want to copy all music onto his laptop as he is short on space, but he wants to have all of the songs on the playlist "Brian favorites"."*

Because the playlist does not exist in the hierarchy, the user had to add the nine files in the playlist individually, after looking up the locations using iTunes. In the semantic system, the playlist is included as a tag, allowing the user to specify the policy in a single step.

8.1.3 Observations

I was surprised at how well novice users were able to perform these tasks, even with the more complex volumes interface. The overall accuracy rate for the views-facet interface was 87%. The views-directory interface trailed slightly, at 81%. The volumes interface had the lowest accuracy of 69%.

The think-aloud nature of my study allowed us to observe a variety of trends in the way users utilized these interfaces. For example, users did have trouble getting accustomed to common UI conventions, such as expanding

file groups and working with popup menus; in the pilot test, before I added some initial nuts-and-bolts interface training, they quickly became lost. Initial confusion may also be due in part to the Expandable Grid interface, which provides a large amount of information at once. However, after several tasks, most users mentioned that they felt quite comfortable with the visualization.

I also found that users often assumed that a user's files must be on their laptops, despite evidence in the interface to the contrary. I started many of my tasks in what I expected would be a fairly intuitive state: all the files stored on the desktop and none anywhere else, but users seemed puzzled at why a user's data wouldn't be on their laptop or why TV shows wouldn't be on the DVR, and they would sometimes assume this must be the case. When first using the interface, a number of users tried to drag a user's laptop around as proxy for the data they owned, even though the interface stated there was no data stored there. This suggests that users may have difficulty initially separating data categories from where they are stored.

As suggested by my contextual analysis, users found hierarchies difficult to use. Users often failed to expand file groups, even after the training task walked them through this operation, and instead puzzled for long periods of time about how to reason about the folders. Even the faceted metadata case suffered some from this challenge, as the information is presented as a set of groups into which the user can drill down.

Finally, I found that users seemed quite comfortable with faceted organization after some experience. I thought that previous experience with hierarchies would make users likely to misunderstand a faceted organization.

8.1.4 Results

All of the statistically significant comparisons are in favor of the facet interface over the alternative approaches, showing the clear advantage of semantic management for these tasks. For the single replica tasks, the facet and directory interfaces perform comparably, as expected, with an average accuracy of 95% and 100%, respectively, compared to an average of 15%

for the volume interface. For the data organization tasks, the facet interface outperforms the directory and volume interfaces with an average accuracy of 66% compared to 14% and 6%, respectively. Finally, while the accuracy of sparse tasks is not significantly different, the average time for completion for the facet interface is 73 seconds, compared to 428 seconds for the directory interface and 559 seconds for the volume interface. I discuss my statistical comparisons and the tasks in more detail in this section.

Statistical analysis: I performed a statistical analysis on my accuracy results in order to test the strength of my findings. Because my data was not well-fitted to the chi-squared test, I used a one-sided Fisher's Exact Test for accuracy and a t-test to compare times. I used Benjamini-Hochberg correction to adjust my p values to correct for my use of multiple comparisons. As is conventional in HCI studies, I used $\alpha = .05$. All comparisons mentioned in this section were statistically significant, except where explicitly mentioned.

Single replica tasks: Figure 8.4 shows results from the single replica tasks. As expected, the directory and view interfaces, which both have a single replica class, perform equivalently, while the volume interface suffers heavily due to the extra complexity of two distinct replica classes. The comparisons between the single replica interfaces and the volume interface are all statistically significant. I do not show times, because they showed no appreciable differences.

Data organization tasks: Results from the three aggregation tasks (U2, RF, and TV) and the two comprehension tasks (TB and TM) are shown in Figure 8.5. As expected, the faceted metadata approach performs significantly better than the alternative approaches, as the filesystem structure more closely matches that of the applications. The facet interface is statistically better than both other interfaces in the aggregation tasks, but I would need more data for statistical significance for the comprehension tasks.

Figure 8.6 shows the accuracy and time metrics for the sparse tasks (BF and HV). Note that none of the accuracy comparisons are statistically significant. This is because, in the sparse tasks, each file is in a unique location, making the correlation between application structure and filesystem

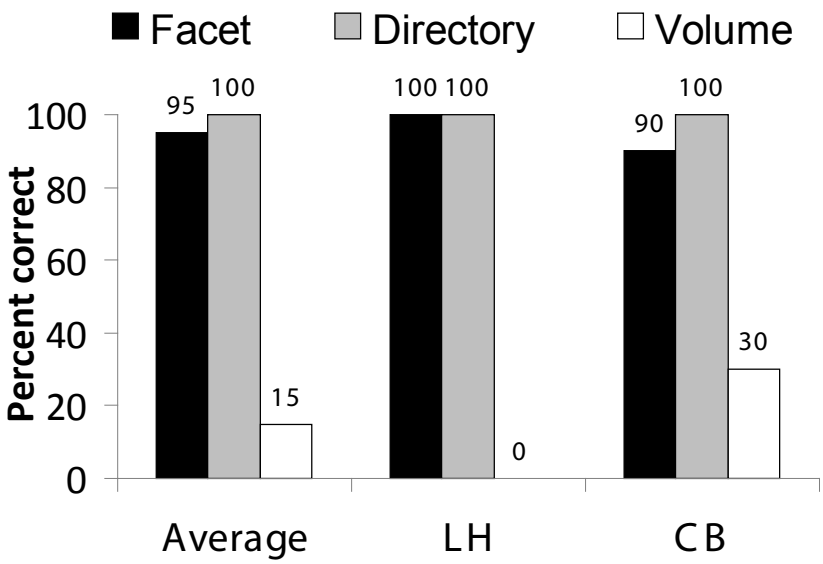

Figure 8.4. **Single replica task results.**

structure clear, but very inconvenient. In contrast, for the other aggregation tasks, the correlation between application structures and the filesystem was hazy, leading to errors. However, setting the policy on each individual file was extremely time consuming, leading to a statistically significant difference in times. The one exception is the HV task, where too few volume users correctly performed the task to allow comparison with the other interfaces. Indeed, the hierarchical interfaces took an order of magnitude longer than the facet interface for these tasks. Thus, re-creating the groups was difficult, leading to frustration and frequent grumbling that "there must be a better way to do this."

8.2 Long-term deployment

While the lab study provided some insight into the benefits of a view-based system, it does not provide insight into the challenges and advantages such a system will have in real, long-term usage. To explore the impact of a view-based system on user behavior, I deployed Perspective into the homes of two

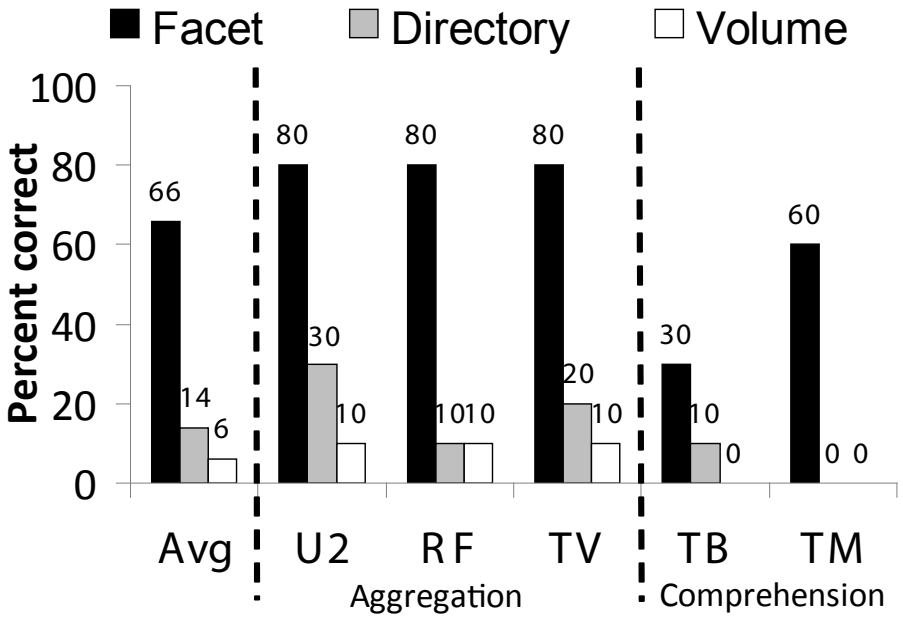

Figure 8.5. **Data organization task results.** This graph shows the results from the aggregation and comprehension tasks.

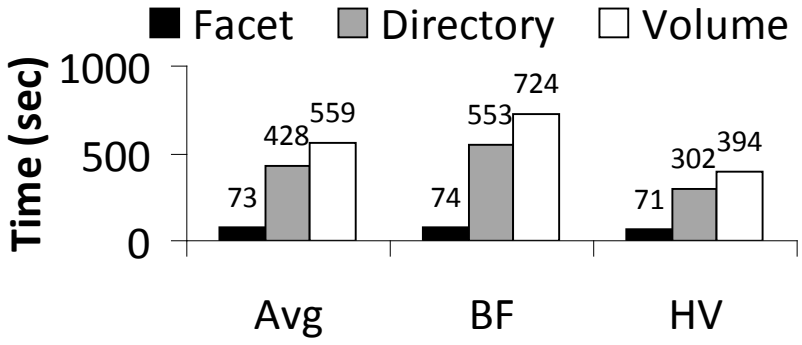

Figure 8.6. **Sparse collection task results.** This graph shows the results from sparse collection tasks.

technically savvy associates and studied the results using several interviews. The deployment has been going on for over four months now. This presents three sets of interviews done during the first month, allowing me to give initial findings. However, the study is also ongoing and should provide more detailed information over a longer period of time.

8.2.1 Initial usage

Perspective was in use, for testing purposes, long before the start of the long-term deployment. The prototype system has supported my household's DVR, which is under heavy use; it is the exclusive television for four roommates and myself and is also frequently used by sixteen other friends in the same complex. It has also stored my personal data for about two years. It has also been the backing store for the DVR in the lounge for our research group for several months.

8.2.2 Methodology

Households: While I ideally would have liked to target non-technical users in this study, as I did for the home server study, the level of maturity of Perspective prohibits us from doing such a deployment yet. For this reason, I chose two technical associates for the deployment.

Deployment hardware: I deployed Perspective into each household and helped each participant migrate their data into the system. I installed Perspective onto all of their existing devices and replaced any devices that could not run Perspective with a Perspective-enabled machine. I also installed a Perspective-based DVR to replace their current DVR implementation and a backup machine to use for recovery should a problem occur with Perspective.

Device and data interviews: The deployment started with two one hour interviews to understand the current data setup of each household. The first interview focused on the data they stored, the devices they used, and the way they moved data from device to device.

Weekly interviews: I also performed an interview, at the start of the deployment, that was identical to the interview held weekly with them after the deployment. This interview focused on various management activities that they might have performed. For example, we asked about recovering from a device failure, backing up data, re-organizing data, etc. For each task, we asked for examples of when they performed the task, the way they handle the problem, how they felt about it, and how much time they spend solving it. I hope to be able to use the comparison between examples given before and after the study to explore the ways that view-based management affects these tasks.

A full listing of the questions used in the study can be found in Appendix C.

8.2.3 Initial findings

While I expect more detailed information to come from the study in the future, the initial exploration has led to several interesting observations about the challenges and advantages of transitioning from files-and-folders based systems to semantic, distributed systems.

Ralph's setup: Figure 8.7 shows a picture Ralph drew of his device setup. Table 8.1 shows a summary of the devices and data Ralph stored in his home. It is interesting to note that many of the patterns we observed in our non-technical households are also true for this very technical user. He owns several old devices that have obsolete data that was never thoroughly copied forward. One of these is a laptop that is old enough he does not remember how to access it. He does use an automated backup (Time Machine) to back up his laptop.

Changes brought with Perspective: Ralph used Perspective to keep a copy of his files in sync between his laptop and desktop machine, rather than requiring a manual backup task.

He also used Perspective to sync copies of shows he recorded on the DVR to his desktop machine. This was a process he did previously by hand. He noticed that this took much less time, but that it did push him away from

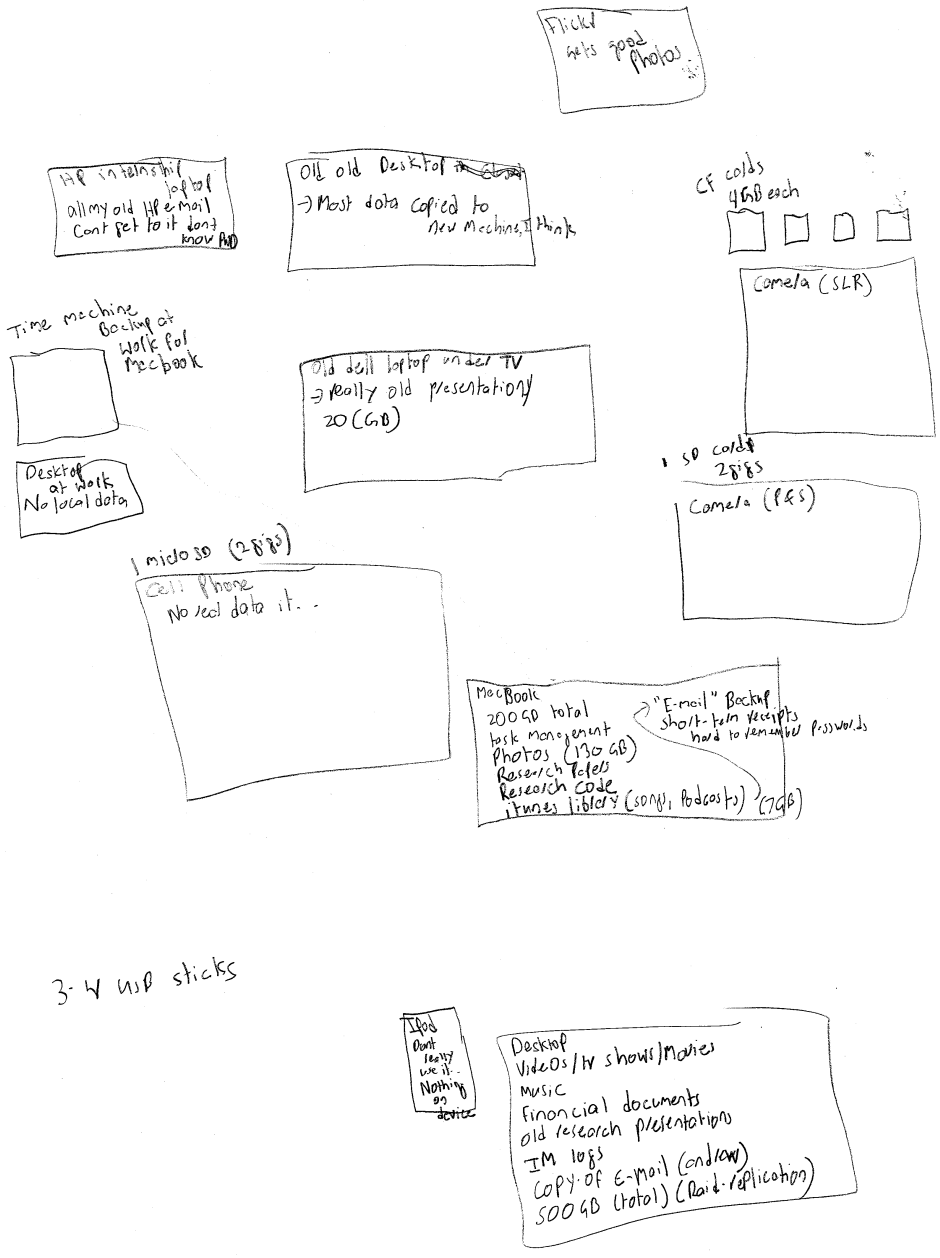

Figure 8.7. **Ralph's device setup.** This figure shows a diagram draw by Ralph as we discussed his data and device setup.

Device	Data	How it got there
MacBook laptop	Photos (130GB)	Pulled from cameras
	Research papers	
	Research code	
	iTunes library (songs, iPod backup)	
Old dell laptop	No data	This machine connected to TV
Old old desktop	No active data	Thinks most data moved forward
HP laptop	HP Email	Used previously, no longer remembers password
Time machine backup disk	Backup of MacBook	Occasional backup
iPod	None that is used	Not really used
Desktop	Videos / tv shows	
	Music	
	Financial documents	
	Old research presentations	
	Copy of email from work	Copied from MacBook

Table 8.1. **Ralph summary.** This table shows a summary of the storage devices Ralph uses, the data stored on these devices, and the way in which this data is added to the given device.

the highly structured, hand-built folder structure he had used previously towards a more course-grained, group-based style of management. Before he would spend Saturday mornings categorizing and labeling this data. But, because the tags were automatically created for him by the DVR, he was more likely to take them as they were.

Steve's setup: Figure 8.8 shows a picture Steve drew of his device setup. Table 8.2 shows a summary of the devices and data Steve stores in his home.

Steve has a large number of devices in his home. He stores much of his data on a laptop, which he then backs up to a server he stores in the closet. He also keeps a desktop machine at work, which he uses to share photographs through a web server. He also has a computer he uses for a DVR to record shows.

Steve keeps a backup of all his data on the server he keeps in his closet. He does this using the rsync tool. He also occasionally syncs photo data with the work desktop, so he can display this information online. He also keeps one set of data, the raw photographs, only on the machine in his closet, because it is large and he does not want to pay to keep it backed up twice.

Changes brought with Perspective: Perspective allowed Steve to replace the rsync paths he was manually managing with an automatically synced copy of his data. This extended to both the DVR, the desktop and

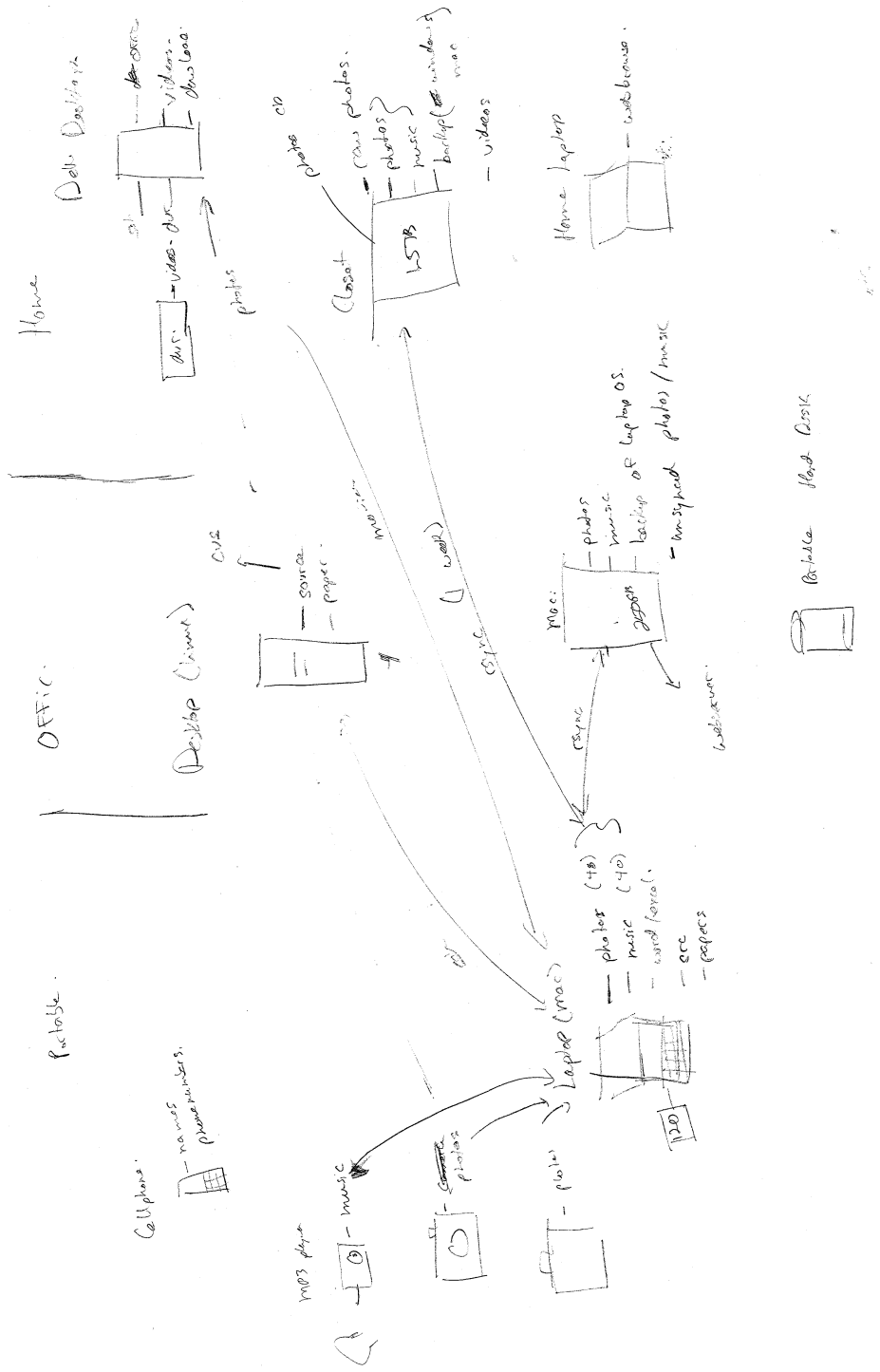

Figure 8.8. **Steve's device setup.** This figure shows a diagram draw by Steve as we discussed his data and device setup. Note the arrows showing the way that data moves from device to device in Steve's setup.

Device	Data	How it got there
Mac desktop	Photos	
	Music	
	Unsynced music	
	Unsynced photos	
Mac laptop	Photos	
	Music	
	Word documents	
	Source / papers	
Dell desktop / DVR	Videos DVRed	
	Videos download	
Server in closet	Raw photos	Copied from laptop.
	Photos backup	Uses rsync.
	Music backup	Uses rsync.
	Videos	Uses rsync from DVR.
Home laptop	No data	Old laptop used for web browsing.

Table 8.2. **Steve summary.** This table shows a summary of the storage devices Steve uses, the data stored on these devices, and the way in which this data is added to the given device.

the laptop machine. Steve also used Perspective to shuttle photos from home to his work machine, so that he could display these photos over the web.

8.3 Performance overheads

While performance is not the primary evaluation metric of Perspective, it is important to explore the overheads Perspective introduces. I have found that Perspective generally incurs tolerable overhead over the base filesystem, and its performance is sufficient for everyday use. The most telling evaluation is that Perspective has been used by myself for two years as the backing store for several multi-tuner DVRs, without performance problems. Another lab member and two test households have also been running Perspective for one month to store their actual data.

System	Write	Read
HFS+	18.1 s	17.0 s
Perspective	18.6 s	17.2 s

Table 8.3. **Simple benchmark.** This chart shows the results of a simple benchmark that writes and reads 800MB worth of 4MB files. The overhead of Perspective is under 3%. Results shown are an average of 30 runs.

8.3.1 System overhead

I used a simple benchmark to evaluate the overhead of Perspective for home workloads. This test was run on a MacBook Pro 2.5GHz Intel Core Duo with 2GB RAM running Macintosh OS X 10.5.4. My benchmark writes 200 4MB files, clears the cache by writing a large amount of data elsewhere, and then re-reads all 800MB. This sequential workload on small files simulates common media workloads. For these tasks, I compared Perspective to HFS+, the standard OS X filesystem. Figure 8.3 shows the results. Perspective introduces less than a 3% overhead in both phases of this benchmark.

8.3.2 Transfer overhead

Similarly, Figure 8.9 shows the overhead of Perspective in copying data from one device to another. Perspective only introduces 4% of overhead in the remote case where we copy data from one device to another. An interesting point of this experiment is that the inherent overhead of Perspective, the cost of doing sync, is a very small fraction of the overhead, meaning that most of the overhead is simply in the prototype's less efficient data transfer mechanism.

I used two MacBook Pros, with 1.87 GHz processors and 1 and 2 GB RAM for this experiment. I connected these devices to a 10Mbps half duplex wired hub to approximate home wireless bandwidth conditions.

8.3.3 View overhead

In order to route updates to the appropriate devices, Perspective must evaluate file metadata against the views currently in the system. Figure 8.10

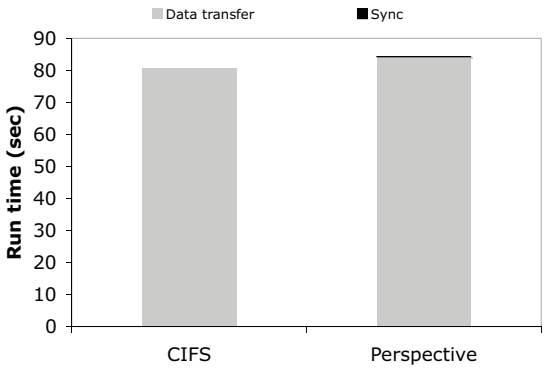

Figure 8.9. **Remote performance.** This test transfers 10 mp3s (64MB total) between devices. Perspective introduces a 4% overhead. However, the inherent protocol overhead is only .14 sec. This graph shows the averages of 5 runs.

shows the cost of evaluating views against foreground updates. In these experiments, I ran tests consisting entirely of reads, writes, and creates through Perspective, but varied the number of views in the system from 0 to 200. I constructed the views so that no objects matched the views, but the system would have to check the views. Each view contained two OR clauses. Even 200 views, considerably higher than my expected number of views in a home deployment, impose negligible overhead on these tests. This is consistent with the results from the Ensemblue experiences [52]. I used a MacBook Pro with a 1.87 GHz processors and 2 GB RAM for this experiment.

8.4 Design choices and performance

This section outlines various novel design points of Perspective and Insight, highlighting how each algorithm or method improves upon existing approaches for the specific focus of Perspective and the relative strengths and weaknesses of the approach compared to alternatives.

While some of these comparisons lend themselves to actual benchmarks, a number of the comparisons are difficult to do quantitatively as they compare methodologies from wildly different systems. Thus, comparing the base

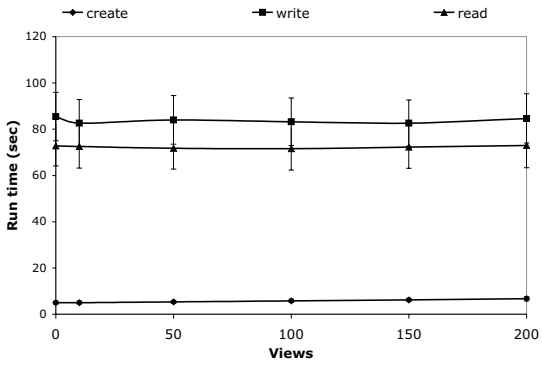

Figure 8.10. **View overhead.** We expect 12-100 views in the system. At even 200 views, the system has no noticeable overhead whem no data items match any view. Each data point is the average of 10 runs, and the standard deviation is also shown.

systems would be inappropriate. While I could simulate many of the approaches using Perspective, it is unclear how valuable this comparison would be. Instead, for these comparisons, I provide a qualitative comparison of the costs involved for each operation, given in big O notation, and a description of the trade-offs involved.

8.4.1 Overlap trees

Overlap trees allow Perspective to efficiently compute how many copies of a given file set are stored in the system, despite the more flexible storage policies that views provide. It is important to make this operation efficient because, while it is only used in administration tasks, these tasks require calculation of a large number of these overlaps in real time as the user browses and manipulates data placement policies.

Table 8.4 summarizes the benefits of overlap trees. We compared overlap trees to a simple method that enumerates all matching files and compares them against the views in the system. We break out the cost for tree creation and then the cost to compute an overlap. The "probe" case uses a query and view set that requires the overlap tree to probe the filesystem to compute the overlap, while the "no probe" case can be determined solely through

Num files	Create OT	OT no probe	OT w/ probe	Simple
100	9.6ms	0.3ms	3.5ms	961ms (.9sec)
1000	29ms	0.6ms	3.8ms	12759ms (12sec)
10000	249ms	0.6ms	3.4ms	95049ms (95sec)

Table 8.4. **Overlap tree benchmark.** This table shows the results from the overlap tree benchmark. It compares the time to create a tree and perform an overlap comparison, with or without probes, and compares to the simple enumerate approach. Each results is the average of 10 runs.

query comparisons. Overlap trees take a task that would require seconds or minutes and turns it into a task requiring milliseconds. This test was run on a MacBook Pro 2.5GHz Intel Core Duo with 2GB RAM running Macintosh OS X 10.5.4.

8.4.2 View-based data synchronization

In order to provide partial replication with disconnection, a system must provide a way for devices to exchange the needed updates after they have been disconnected from one another. Views provide an efficient way to provide this synchronization. However, this area has had a large amount of related work, and no one approach is purely better than the others. Table 8.5 shows a number of alternative approaches currently proposed and an evaluation across several important metrics.

Comparison points: It is difficult to compare synchronization methods, because each system has its own limitations and advantages, often including side-cases. I attempt to capture the various axes of comparison here. One important metric is the cost of synchronization itself. In many systems, this is divided into two separate cases. If the data on one device can be shown to be a superset of the other, many systems provide a more optimized approach. However, it is also possible for two devices without such a superset to sync, usually at a higher cost. Another important area of comparison is the requirements the system makes on where data is stored. Most systems provide some restrictions on replication levels in order to improve sync performance. Another interesting comparison is the requirements on syn-

Approach	**Sync cost superset**	**Sync cost non-superset**	**Storage requirements**	**Sync connectedness**
Server with hoarding	O(number of updates)	Cannot sync	Server must store all data	All devices must sync with server
Pseudo-server	(Server case) O(number of updates)	(Pseudo-server case) O(number of files on device) + O(number of updates)	Server must store all data	All devices must sync with server
Return all data	N/A	O(number of files on both devices)	None	Per-file connectedness, system does not construct
Update log	N/A	O(number of updates * number of replicas)	None	Per-file connectedness, system does not construct
Cimbiosis	O(number of updates)	O(number of updates) + O(number of files on smaller device)	Some device must store all data	*sync tree*, system constructs
PRACTI	N/A	O(number of updates * number of replicas) + O(invalidations)	None	Per-file connectedness, system does not construct
Views	O(number of updates)	O(number of updates * devices with matching views)	None	Per-file connectedness, freshness timestamps show when not held

Table 8.5. **Synchronization methods.** This table shows alternative approaches to synchronization for partial replication with disconnection and costs and limitations of each approach.

chronization patterns in order to assure that data on all devices is kept up to date. A system must make sure that an update generated on one device will eventually reach all of the other replicas of this file.

Alternative approaches and comparison: One alternative is the *server with hoarding* approach, as proposed by Coda [67]. This system provides good synchronization performance with the server, only requiring the updates that are actually relevant to each device to be sent to that device. However, it does not allow the client devices to sync with any device other than the server, a limitation not shared by the other approaches listed. It requires that all data is stored on the server and requires all client devices to sync with the server.

The *pseudo-server* case extends upon this case by allowing client devices to sync with one another. It does so by constructing a pseudo-server, a device elected as a leader, by copying all of the metadata on the devices in the ensemble onto the pseudo-server. In this approach, sync when the server is present is as efficient as the server with hoarding case. However, sync costs in an ensemble are much higher. The device must upload all of its metadata to the pseudo-server and then receive all needed updates back from the pseudo-server. In Ensemblue [52], because the server still determines the authoritative ordering of updates, all devices must sync with the server to keep data in sync. All data must still be stored on the central server. However, a pseudo-server approach could be layered on top of a consistency method that did not have these restrictions.

The *return all data* approach is a simple alternative to a server-based approach. In this approach, on a sync request, a device returns the metadata for all files that it stores. This approach is simple to implement, but sync costs are high: each sync cost is order of the number of files on the device. This approach does not require any limitations on where data is stored. In order to assure that all replicas of a file are kept in sync, there must be a sync path from any file replica to all other replicas. I call this *per-file connectedness*. This is described in detail in Section 6.2.3. However, a return-all-data approach does not provide a way to guarantee this kind of connectedness; the user or some other tool must provide feedback if the

property is not held.

The *update log* approach improves upon the performance of return-all-data by keeping a log of recent updates on each device, and tracking the times at which pairs of devices have synchronized. In this approach, the cost of a sync operation is the number of updates. However, because each pair of syncs is independent, each device will see each update once for each other replica of the file in the system, assuming it syncs eventually with all other devices. This kind of system puts no requirements on where data must be stored. However, it still requires some tool to determine when file replicas may be drifting out of sync.

The *Cimbiosis* [59] system provides an approach based on *sync trees*. In Cimbiosis, some device must store all the data in the system. Each device then finds a parent device with a superset of the data it stores. This allows Cimbiosis to limit the versioning information in the best case to a single version number. For Cimbiosis, the cost of a sync within the sync tree is only this constant sized version and then the needed updates. Device pairs where one device is not a descendant of the other can still sync, at a higher cost. In this case, the devices must exchange the updates and then a list of the IDs of all files stored on the device. This is order of the number of files on the device, although it only requires the IDs, rather than the full metadata. Cimbiosis requires that some device store all files in the system, to serve as a root for the tree. However, the *sync tree* assures that all replicas of each file are kept in sync, so long as devices syncs occasionally with their parent, an advantage over other approaches.

The *PRACTI* system provides a similar approach to the update log, but allows devices to substitute groups of updates that are not relevant to a given device with a constant sized *imprecise invalidation*. This keeps the cost of a synchronization down to the number of updates plus a small number of invalidations. However, it will not filter updates from other devices, so it will see each update once for each replica of the file in the system. It does not distinguish between a superset case, providing the same performance in all cases. It also does not require data to be stored in any particular location. Like most approaches, it does not provide a way to guarantee

connectedness. PRACTI also provides more flexible consistency possibilities, whereas the remaining approaches are all per-file consistency. PRACTI is also build around a files-and-folders system; while I expect it can be ported to a semantic system it is not clear how this could be done.

In contrast, views provide many of the advantages of Cimbiosis without requiring any device to store all the data found in the filesystem. In cases where the data on one device is a superset of the other, determined using view logic, views provide the ability to pass only the needed updates. Because the child device can pull all of the version numbers from the parent device, it will not see updates from other devices storing file replicas. If the two devices do not have a superset relationship, views still limit the transfer to the number of updates needed. However, it loses the ability to filter these updates from subsequent syncs, so it will see each update once for each file replica. The workload will determine whether this overhead, or the Cimbiosis overhead for transferring the IDs of all files on one device, will be more costly. Views do not require any device to store any particular set of data. Views require *per-file connectivity* to ensure all replicas are updated, a different property than the sync tree property. It is not clear which of these invariants is more difficult to maintain or more valuable. Freshness timestamps, as described in Section 6.2.3, provide a way to ensure that all replicas are connected, and provide suggestions if it is not the case.

8.4.3 View-based distributed search

Views provide a way to efficiently perform distributed search in a device ensemble. Views provide advantages over existing methods, but also some trade-offs. I compare views with four other approaches for three metrics. Table 8.6 shows the alternative methods and a comparison between them.

Comparison points: I compare alternative approaches based on three metrics. First, I compare the network overhead of ensemble creation. Each approach requires the exchange of some amount of information in order to initially establish a searchable ensemble. Second, I compare the network overhead of a search. It is critical to make search efficient in a distributed

semantic system, as this is the basis of all naming for the system. For systems with rich metadata, these network costs can be sizeable. For example, the metadata on my laptop is around 330MB, meaning that a five device ensemble might have to exchange up to 1.5GB of data simply to establish an ensemble or do a search.

Third, I compare the number of devices that must respond to each query. While this is not a particularly important metric when devices are of a similar size and are not power limited, in a heterogeneous environment like the home, it is critical that small devices, like cell phones, participating in the network do not have to handle all searches that much more powerful devices, like desktop computers, issue. A power and CPU limited device like a cell phone would have little chance of keeping up with the other devices.

Alternative approaches and comparison: The most efficient implementation of search is the *pre-built server* case. This is not possible when one has an ad-hoc collection of devices, but serves as a best-case comparison point for other approaches. In this approach, there is no cost to create the ensemble, since all the data is already stored on the server. The cost to search is limited to the number of files in the result, and only one device, the server, must answer any query.

The second approach is *brute force*. This is the approach taken by filesystems without semantic support (e.g. grep). For each search, they run through all of the files in the filesystem looking for results. While an ensemble is easy to create (no information must be exchanged), the cost for each search is proportional to the number of files on all devices in the filesystem, making it infeasible for frequent use. In addition, all devices must respond to all queries.

The third approach is the *Diamond* approach [32]. This approach extends each individual device with search functionality and then forwards all searches to all devices. The cost for ensemble creation is still negligible. The cost for a search is reduced substantially; because each device searches internally, the network cost is only the number of results found on all devices. However, this approach introduces extra costs that are infeasible in the extremely heterogeneous home environment. In this approach, a cell phone

participating in the network would have to handle all searches that much more powerful devices, like desktop computers, issued, even if the searches could not contain any of the data on the cell phone. A power and CPU limited device like a cell phone would have little chance of keeping up with the other devices.

The fourth approach is the *pseudo-server* approach [52]. In this approach, the devices in the ensemble elect a powerful device to be a pseudo-server for the ensemble. All devices send their metadata to this device, which then acts as a server for the duration of the ensemble. This provides efficient searches, and only requires a single device to answer a query. However, ensemble creation is very expensive, making it difficult for ensembles with dynamism. This approach also requires at least one device in the ensemble to be a "powerful" device, capable of serving the metadata of all other devices in the ensemble.

In contrast, views provide efficient search without the cost of building a central metadata store and without the extra cost of sending queries to unneeded power-limited devices. If there is a complete view currently accessible that covers the given query (as shown by the *Views (complete view)* row), views will act just like the central server case by noticing that the query only needs to go to that device. It thus obtains the same benefits without requiring a centralized server and without requiring any device to have a copy of all metadata in the system.

If an appropriate complete view is not available (as depicted by the *Views* row), views allow the system to only query devices that could contain matching objects. This allows a power-limited device to participate in the ensemble without having to see most of the queries in the system. Of course, it is possible that the views in the system do not match the query, and we must search all devices. In this worst case scenario, views match the Diamond case, as all queries are forwarded to all devices.

One scenario where views might be less efficient is in the case where ensembles are very stable and views do not well match the queries in the system. In this case, views will behave like the Diamond approach, which

Approach	**Create ensemble network cost**	**Search network cost**	**Devices queried**
Pre-built server	None	O(number of results files)	1
Brute force	None	**O(number of files on all devices)**	**All devices**
Diamond	None	O(number of copies of result files)	**All devices**
Pseudo-server	**O(number of files on all devices)**	O(number of results files)	1
Views	O(number of views)	O(number of copies of results files)	Devices with matching views
Views (complete view)	O(number of views)	O(number of results files)	1

Table 8.6. **Search methods.** This table shows the costs of various operations in each of the alternative approaches to search. The limitation that caused me to use views is highlighted for each approach in bold.

will require a little extra network work for replicated files over the it pseudo-server approach.

8.4.4 View-based event routing

Views also provide a way to route events to relevant devices. The most common use of such a mechanism in a filesystem is to forward updates made to one file replica to all other replicas of that file in the current ensemble. However, the same methods can also be used to provide application-level event notification. This section compares view-based event routing with other approaches. I compare views with three alternative event routing approaches on three metrics. Table 8.7 outlines the alternatives and their trade-offs.

Comparison points: I compare alternative approaches based on two metrics. First, I compare the network overhead of ensemble creation. Second, I compare the number of devices sent a message. This determines both the network overhead and the number of devices that must spend power on the operation.

Alternative approaches and comparison: As with search, the *pre-built server* case is ideal. In this case, we do not need to transfer any data on ensemble creation. We only need to route updates to devices that we are sure hold the file replica since the server has a full index, and we only need to wake the devices with replicas to which the update belongs. We cannot use this approach if we do not have a central server, but it is useful as a best-case scenario.

Another approach is to use a *pseudo-server* [52]. As in the search case, on ensemble creation, devices elect a leader and send all metadata to this device. All devices then route any update through the pseudo-server. This allows the system to only send updates to devices that actually have a file replica, although the pseudo-server also must be included, even if it does not store a replica of the file. However, ensemble creation again requires transfer of all metadata on all devices to the pseudo-server, an expensive operation.

Another approach is to use a *send-to-all* approach. In this approach, the system does no coordination on ensemble creation, and passes all updates to

Approach	**Create ensemble network cost**	**Devices sent messages**
Pre-built server	None	O(number of file replicas)
Pseudo-server	**O(number of files in the system)**	O(number of file replicas)+1
Send-to-all	None	**All devices**
Views	O(number of views)	O(number of matching views)

Table 8.7. **Event routing methods.** This table shows the costs of various operations in each of the alternative approaches to event routing. The limitation that caused me to use views is highlighted for each approach in bold.

all devices, allowing them to determine what updates are relevant to them. While this is simple and requires little overhead in ensemble creation it is expensive for each update. This is especially problematic in a heterogeneous system, where some devices may be resource-constrained.

The view-based approach provides very little ensemble set up time (only the exchange of views) and then routes updates to devices with matching views. In the normal case, where all views are precise (i.e. all complete views), this will provide the same performance as the central-server case without the extra cost of ensemble setup. If the views are imprecise (partial views), then updates may be sent to devices that have matching views but do not actually store all of the matching replicas. As partial views are not currently used heavily in Perspective, this is not a large concern.

9 Conclusion

This dissertation presents the view abstraction to correct the disconnect between semantic data access and folder-based replica management. A *view* is a compact description of a set of files, expressed much like a search query, and a device on which that data should be stored. For example, one view might be *"all files with type=music and artist=Beatles stored on Liz's iPod"* and another *"all files with owner=Liz stored on Liz's laptop"*. Each device participating in a view-based filesystem maintains and publishes one or more views to describe the files that it stores. A view-based filesystem ensures that any file that matches a view will eventually be stored on the device named in the view.

In this dissertation, I present the view architecture, the design of a view-based filesystem, the design of view-based management tools, and a set of user studies exploring the advantages of such a management system.

9.1 Contributions

This dissertation contains a number of contributions to the literature. First, I present two exploratory studies into home storage. While the literature contains a rich history of home studies, my studies each have unique focus. The deployment study is unique in focusing on the way non-technical users react to a distributed filesystem. The contextual analysis of home users is unique in its focus on device upgrade and reliability of home data.

Second, I present the first semantic abstraction for replica management, the view, and a user study evaluating the usability impact of semantic management in contrast with more conventional approaches.

Third, I present a group of algorithms needed to provide the view abstraction in a distributed filesystem. These algorithms include the first consistency protocol that allows for semantically-defined partial replication in an eventually consistent, heterogeneous, decentralized filesystem environment. These algorithms also include methods to provide efficient search in a view-based system.

Fourth, I present Perspective, the first filesystem to provide view-based replica management. Perspective is fully functional and is in use in several household in the Pittsburgh area. In addition to showing the feasibility of a view-based filesystem, the Perspective prototype provides a platform for continuing research into distributed semantic storage, and distributed home storage.

Fifth, I present a group of tools which allow users to manipulate file replicas using the view abstraction. I introduce *overlap trees* as a mechanism for efficiently reasoning about how many replicas exist of a particular dataset, and where these files are stored, even when no view exactly matches the attributes of the dataset. I present the *view manager interface* as an interface for semantic replica management. I also present *customizable faceted metadata* as a way to browse semantic filesystem data.

Sixth, I present results from an initial deployment of the Perspective filesystem. These results explore some of the initial advantages and challenges of view-based management in practice.

9.2 Future and ongoing work

There is a lot of ongoing and possible future work in the area of semantic home storage. In this section, I outline areas that would be interesting to explore in more depth and areas where other students are building on the Perspective system.

9.2.1 Security

This is a critical area in the home. My user studies have shown that privacy and security are important in the home environment and that conventional

access control methods do not appear to meet this need. A number of students and faculty are working on both mechanisms to provide the appropriate security abstractions and user study work to understand the security needs of home users.

9.2.2 Semantic system deployment

The long term deployment is continuing in several households. In addition to providing a test ground for various home storage research, this deployment should provide greater insight into how users manage semantic naming over a long period of time and how they handle less frequent management tasks, such as device failure and device upgrade.

9.2.3 Visualizing semantic data

Perspective uses a variant of faceted metadata to visualize the data in the filesystem. However, a large number of approaches are possible, such as keyword search, hierarchical tag namespaces, etc. Exploration into the effects of these various techniques on data management, especially over long periods of time, would be valuable.

9.2.4 Efficient faceted data storage

While the current implementation of Perspective is efficient enough to support day-to-day usage, faceted metadata support would still be an interesting area of exploration. As the queries required by users and applications grow more complex, it is unlikely that the current Perspective search implementation, built on top of a standard database, will be sufficient. However, I expect an implementation tuned to this kind of workload could do much better.

9.2.5 Update connectivity

Perspective uses freshness timestamps to help users understand when replicas are out of date. However, this algorithm is still somewhat crude. Explo-

ration into both the user interfaces and algorithms required to help home users understand the freshness of their data in a Perspective-style system would also be valuable to the field. One interesting area could be the application of techniques similar to those in Ebling et al. [15] to allow users to understand what versions of various files have already be transferred to various devices.

9.2.6 Replica removal and deletion

One challenge for novice and expert users alike is distinguishing between the removal of a single replica and the removal of all copies of a file. Exploration into how best to provide and explain this distinction to non-technical users would be valuable research.

9.2.7 Conflicts

Disconnected systems must deal with data conflicts occasionally, as users modify two replicas at overlapping time periods. Now that we have a long term deployment, exploration into how frequent these conflicts are, how they happen, and how users deal with them would be valuable research.

A Home data contextual analysis questions

A.1 Personal questions

What do each of you do? Are you students, working?

How do you know each other?

How long have you been living together?

A.2 Data and Devices

Please list the digital storage devices in your home. This might include, computers, music players, TiVo, external hard drives. Feel free to go to your devices to answer these questions.

Device	Owner	Usage	Capacity (in GB)	% full	Leaves house?

Please describe the data on each device, including the size of the data. This could be a division based on data type, usage, etc. Also, tell us how much you would care if you lost this set of data. Feel free to browse the data on your devices to answer these questions. Please explain why you feel the way you do about each data set.

How much would you care if you lost this data?

1	Don't care
2	Annoyance
3	Considerable loss of time or money
4	Big loss, couldnt or wouldnt want to replace
5	Disastrous

Device	Data type	Size (in GB)	Would you care if lost?

How much of your data was created in the last year?

User	Percent created last year

Do you back up any of your data? How? (Could be to a disk drive, to CDs, etc.) How frequently? (daily, weekly, monthly, yearly)

Data type	Backup type	Frequency

A.3 Scenarios

Describe the last time you backed up your data. (Include software, hardware, etc.) Can you step through a backup right now?

Have you had digital storage devices fail before? If so, describe what happened.

Have you ever restored your data from a backup? If not, why? If so, describe it.

Have you run short on disk space on a device before? What happened? What did you do?

Describe the process of setting up your data on the last new device you purchased.

Have you ever needed data and not been able to access it? Describe the situation, and what you did to solve it.

Describe the last time you copied or moved data from one device to another. What was the data, why did you move it, and how did you move it?

Talk about the last time you shared data between household members. What data? When? Why?

B Usability lab study tasks

B.1 Welcome

You are testing a new system to control the storage of computer files in the home. This system allows you to access files stored on any device (laptop, TiVo, etc) from any device it can talk to. This means that if all their devices are in their house you can access any data stored on any device. However, if you were to take a device on a trip, for example, you may not be able to access the data on devices left at home. The system also provides a global name for files; so a file's name is independent of where it is stored, and the same no matter which device you use. So a file might be owned by Brian and called "MyFile", but stored on any number of devices.

This system also allows you to place copies of the same file on multiple devices, which the system will assure are kept up to date with updates made to any of these copies. The system you will be testing allows you to control where copies of files are stored in this system. This system is still in the testing phase, so some operations may be slow; please be patient with it.

You will be working with Brian and Mary's household today. Brian and Mary are a young couple. Both Brian and Mary have laptops that they take with them to work. They also have a family desktop machine and a Digital Video Recorder (TiVo) that they have in their apartment. Today, we will ask you to perform several tasks for them. They share many of their files between their devices at home using this new system. They listen to each others music on their laptops and they watch movies on their TiVo, for example.

NEWPAGE

We would like you to "think aloud". As you do each of the tasks, please read any instructions out loud, and talk out loud about what you are thinking, what surprises you, and what you expect. We will record this, which will help us analyze our system. As you do the task, please remember that we are evaluating our system, not you. Feel free to ask any questions you may have before starting. However, to avoid biasing our test, we cannot answer questions once you have started the tasks.

When you are ready to start, please inform the researcher.

B.2 Training Views task

The "File copy placement tool" window on the left lets you control where data is stored in your house.

On the left of the window are files. Each row represents either an individual file, or a group of files. Groups of files have a triangle to the left of their name. Clicking on the triangle shows the things inside of that group.

Task 1: Click on the triangle to the left of "All Files" to see all the contents of this file group.

Task 2: Now click on the triangle next to "All files grouped by owner" to see the contents of this group. When you are finished, click on the triangle next to "All files grouped by owner" again to hide the contents of this group.

Task 3: Next, click on the triangle next to "All files grouped by type" to see the contents of this group.

Task 4: Next, click on the triangle next to "Documents" to see the contents of this group. When you are finished, click on the triangle next to "Documents" again to hide the contents of this group. Then click on the triangle next to "All files grouped by type" again to hide the contents of this group.

Across the top of the window are devices. Each device in the house is represented by one column. Each square in the grid represents whether the files in the row are stored on the device in the column. For example, a white square shows that none of the files in the row are stored on the device in the column, while a dark blue square shows that all of the files in the row are

stored on the device in the column, and a light blue square shows that some, but not all of the files in the row are stored on the device in the column. To change the setting, you can click on the square. A list of possible actions will show up on the screen. Clicking on one of the items in this list will perform the listed action.

Task 5: Click on the square in the row of "All files" and the column of "Family desktop". The words "Store here" will appear. Click on "Store here". Now all the files in the household are stored on the family desktop.

NEWPAGE

Alternately, if you drag a group of files onto anywhere in the column for a device, it will also store the files in that group on the device in the column.

Task 6: Now, store the files owned by Mary on Mary's laptop.

The "Summary of failure protection" column shows whether the files listed in the row are protected if a single device in the house fails. The dark orange squares show that this group of files will be safe even if any one home device crashes. The other colors represent partially or unprotected file groups.

The "Summary of files on each device" row shows a summary of what files are stored on what devices in the home. This allows you to see exactly what is stored on each device at a glance, without having to look through all of the file categories.

Task 7: Click on the triangle next to the "Summary of files on devices" group to see what files are stored on what devices.

Every time you tell the system to put a group of files onto a device, a new row is created in this section. By looking at the dark blue squares in the column for a device, you can tell exactly what you have told the system to store on that device. For example, all files are stored on the Family desktop. You can use this column to determine all the files on a given device, or what files are protected from failure.

Task 8: Use the "Summary of files on devices" group to answer the following question:

TEXTBOX BrianTravelSeeAnswer What files are stored on Mary's laptop? Be as precise as possible, but give your answer in terms of file groups (no need to list individual files).

B.3 Training Directories task

The "File copy placement tool" window on the left lets you control where data is stored in your house.

On the left of the window are files. Each row represents either an individual file, or a group of files. Groups of files have a triangle to the left of their name. Clicking on the triangle shows the things inside of that group.

Task 1: Click on the triangle to the left of "All Files" to see all the contents of this file group.

Task 2: Now click on the triangle next to "Brian" to see the contents of this group. When you are finished, click on the triangle next to "Brian" again to hide the contents of this group.

Task 3: Next, click on the triangle next to "Mary" to see the contents of this group.

Task 4: Next, click on the triangle next to "Documents" to see the contents of this group. When you are finished, click on the triangle next to "Documents" again to hide the contents of this group. Then click on the triangle next to "Mary" again to hide the contents of this group.

Across the top of the window are devices. Each device in the house is represented by one column. Each square in the grid represents whether the files in the row are stored on the device in the column. For example, a white square shows that none of the files in the row are stored on the device in the column, while a dark blue square shows that all of the files in the row are stored on the device in the column, and a light blue square shows that some, but not all of the files in the row are stored on the device in the column. To change the setting, you can click on the square. A list of possible actions will show up on the screen. Clicking on one of the items in this list will perform the listed action.

Task 5: Click on the square in the row of "All files" and the column of "Family desktop". The words "Store here" will appear. Click on "Store here". Now all the files in the household are stored on the family desktop.

NEWPAGE

Alternately, if you drag a group of files onto anywhere in the column for a device, it will also store the files in that group on the device in the column.

Task 6: Now, store the files owned by Mary on Mary's laptop.

The "Summary of failure protection" column shows whether the files listed in the row are protected if a single device in the house fails. The dark orange squares show that this group of files will be safe even if any one home device crashes. The other colors represent partially or unprotected file groups.

The "Summary of files on each device" row shows a summary of what files are stored on what devices in the home. This allows you to see exactly what is stored on each device at a glance, without having to look through all of the file categories.

Task 7: Click on the triangle next to the "Summary of files on devices" group to see what files are stored on what devices.

Every time you tell the system to put a group of files onto a device, a new row is created in this section. By looking at the dark blue squares in the column for a device, you can tell exactly what you have told the system to store on that device. For example, all files are stored on the Family desktop. You can use this column to determine all the files on a given device, or what files are protected from failure.

Task 8: Use the "Summary of files on devices" group to answer the following question:

TEXTBOX BrianTravelSeeAnswer What files are stored on Mary's laptop? Be as precise as possible, but give your answer in terms of file groups (no need to list individual files).

B.4 Training Volumes task

The "File copy placement tool" window on the left lets you control where data is stored in your house.

On the left of the window are files. Each row represents either an individual file, or a group of files. Groups of files have a triangle to the left of their name. Clicking on the triangle shows the things inside of that group.

Task 1: Click on the triangle to the left of "All Files" to see all the contents of this file group.

Task 2: Now click on the triangle next to "Brian" to see the contents of this group. When you are finished, click on the triangle next to "Brian" again to hide the contents of this group.

Task 3: Next, click on the triangle next to "Mary" to see the contents of this group.

Task 4: Next, click on the triangle next to "Documents" to see the contents of this group. When you are finished, click on the triangle next to "Documents" again to hide the contents of this group. Then click on the triangle next to "Mary" again to hide the contents of this group.

Across the top of the window are devices. Each device in the house is represented by one column. Each square in the grid represents whether the files in the row are stored on the device in the column. For example, a white square shows that none of the files in the row are stored on the device in the column, while a dark blue square shows that all of the files in the row are stored on the device in the column, and a light blue square shows that some, but not all of the files in the row are stored on the device in the column. To change the setting, you can click on the square. A list of possible actions will show up on the screen. Clicking on one of the items in this list will perform the listed action.

Task 5: Click on the square in the row of "All files" and the column of "Family desktop". The words "Store here" will appear. Click on "Store here". Now all the files in the household are stored on the family desktop.

NEWPAGE

Alternately, if you drag a group of files onto anywhere in the column for a device, it will also store the files in that group on the device in the column.

Task 6: Now, store the files owned by Mary on Mary's laptop.

The "Summary of failure protection" column shows whether the files listed in the row are protected if a single device in the house fails. The dark orange squares show that this group of files will be safe even if any one home device crashes. The other colors represent partially or unprotected file groups.

Each device is either a server, which can only store permanent copies of top level directories, or a client, which can only store temporary copies of any group of files. Clicking on the device allows you to switch it between a server or client device.

The "Summary of files on client devices" row shows a summary of what files are stored on what client devices in the home. This allows you to see exactly what is stored on each device at a glance, without having to look through all of the file categories.

Task 7: Click on the triangle next to the "Summary of files on client devices" group to see what files are stored on what devices.

Every time you tell the system to put a group of files onto a device, a new row is created in this section. By looking at the dark blue squares in the column for a device, you can tell exactly what you have told the system to store on that device. For example, all files are stored on the Family desktop. You can use this column to determine all the files on a given device, or what files are protected from failure.

Task 8: Use the "Summary of files on devices" group to answer the following question:

TEXTBOX BrianTravelSeeAnswer What files are stored on Mary's laptop? Be as precise as possible, but give your answer in terms of file groups (no need to list individual files).

B.5 Training

Mary uses iTunes and iPhoto to manage her music, movies, tv shows and photos. You may also need to use these interfaces while completing these tasks. You will only need the operations we show you in this introduction, and you will not need to change anything from these interfaces.

To open iTunes, double click on the icon labelled iTunes on the desktop.

Task 1: Click on the iTunes icon to open the iTunes interface.

Across the left side of the iTunes interface are various groups of music files, including user playlists. When you click on one of these groups the files in that group appear in the window on the right.

Task 2: Click on "Music" in the left pane to see all the music.

To view information on a song, right click on the song name in the right hand window, then click on "Get info". This will show various properties of the file, including location.

Task 3: View the properties of the song "Different".

Now close iTunes by clicking on the red circle in the upper left corner of the window. To open iPhoto, double click on the icon labelled iPhoto on the desktop.

Task 4: Click on the iPhoto icon to open the iPhoto interface.

Across the left side of the iPhoto interface are various groups of picture files, including albums set up by Mary and Brian. When you click on one of these groups the files in that group appear in the window on the right.

Task 5: Click on "Library" in the left pane.

To view information for a picture, right click on the picture, then click on "Show file". This will open a window at the location of the file for this picture. Use the scroll bar at the bottom of the window to move right and left. Do not move anything in this window, just use it to see where things are stored.

Task 6: Look at the location of one of the pictures. Then close iPhoto by clicking on the red circle in the upper left hand corner of the window.

Note that because this is a test, all of the photos will look similar, despite the groupings. Please ignore the contents of the photos themselves, and instead use the labels to guide you on these tasks.

NEWPAGE RADIO TrainingDifficulty Please rate, on a scale from 1 to 7, how difficult you found this task. 1 Not at all difficult 2 3 4 Moderately difficult 5 6 7 Very difficult

RADIO TrainingConfidence Please rate, on a scale from 1 to 7, how confident you are that you correctly completed this task. 1 Not at all confident 2 3 4 Moderately concerned 5 6 7 Very confident

Please wait for the researcher

B.6 Mary's travels

Mary takes her laptop on trips with her. While on these trips she likes to have access to the files she owns, but she doesn't need access to Brian's files or the Family files. Make sure Mary can access all her files when she takes her laptop on these trips.

Use the "File copy placement tool" to the left to change where files are stored in the house, and the iTunes and iPhoto applications to locate data if needed.

When you are finished with the task, push continue.

NEWPAGE RADIO MaryMobilityDifficulty Please rate, on a scale from 1 to 7, how difficult you found this task. 1 Not at all difficult 2 3 4 Moderately difficult 5 6 7 Very difficult

RADIO MaryMobilityConfidence Please rate, on a scale from 1 to 7, how confident you are that you correctly completed this task. 1 Not at all confident 2 3 4 Moderately concerned 5 6 7 Very confident

Please wait for the researcher

B.7 Concerned Brian

Brian is concerned that he might lose the data he owns if the family desktop breaks. However, Mary has asked Brian not to make extra copies of her files

or the Family files. Make sure Brian's files are safe even if the family desktop fails.

Use the "File copy placement tool" to the left to change where files are stored in the house, and the iTunes and iPhoto applications to locate data if needed.

When you are finished with the task, push continue.

NEWPAGE RADIO BrianReliabilityDifficulty Please rate, on a scale from 1 to 7, how difficult you found this task. 1 Not at all difficult 2 3 4 Moderately difficult 5 6 7 Very difficult

RADIO BrianReliabilityConfidence Please rate, on a scale from 1 to 7, how confident you are that you correctly completed this task. 1 Not at all confident 2 3 4 Moderately concerned 5 6 7 Very confident

Please wait for the researcher.

B.8 Mary's laptop comes home

Mary has not taken her laptop on a trip with her for a while now, so she has decided to leave it in the house and make an extra copy of her files on it in case the Family desktop fails. However, Brian has asked her not to make extra copies of his files, or of the Family files. Make sure Mary's files are safely stored on her laptop.

Use the "File copy placement tool" to the left to change where files are stored in the house, and the iTunes and iPhoto applications to locate data if needed.

When you are finished with the task, push continue.

NEWPAGE RADIO BrianReliabilityDifficulty Please rate, on a scale from 1 to 7, how difficult you found this task. 1 Not at all difficult 2 3 4 Moderately difficult 5 6 7 Very difficult

RADIO BrianReliabilityConfidence Please rate, on a scale from 1 to 7, how confident you are that you correctly completed this task. 1 Not at all confident 2 3 4 Moderately concerned 5 6 7 Very confident

Please wait for the researcher.

B.9 U2

Mary and Brian share music at home. However, when Mary is on trips, she finds that she can't listen to all the songs by U2 on her laptop. She doesn't listen to any other music, and doesn't want other songs taking up space on her laptop, but she does want to be able to listen to U2. Make sure she can listen to all music by the artist U2 on her trips.

Use the "File copy placement tool" to the left to change where files are stored in the house, and the iTunes and iPhoto applications to locate data if needed.

When you are finished with the task, push continue.

NEWPAGE RADIO AndrewBirdDifficulty Please rate, on a scale from 1 to 7, how difficult you found this task. 1 Not at all difficult 2 3 4 Moderately difficult 5 6 7 Very difficult

RADIO AndrewBirdConfidence Please rate, on a scale from 1 to 7, how confident you are that you correctly completed this task. 1 Not at all confident 2 3 4 Moderately concerned 5 6 7 Very confident

Please wait for the researcher.

B.10 TV

Brian likes to watch recorded and downloaded TV shows using the TiVo. However, sometimes when Brian sits down to watch the TiVo, he can't see some of the TV shows. Make sure Brian can watch any TV show using the TiVo, even if the other devices in the house may be turned off or outside of the house.

Use the "File copy placement tool" to the left to change where files are stored in the house, and the iTunes and iPhoto applications to locate data if needed.

When you are finished with the task, push continue.

NEWPAGE RADIO BrianMobilityDifficulty Please rate, on a scale from 1 to 7, how difficult you found this task. 1 Not at all difficult 2 3 4 Moderately difficult 5 6 7 Very difficult

RADIO BrianMobilityConfidence Please rate, on a scale from 1 to 7, how confident you are that you correctly completed this task. 1 Not at all confident 2 3 4 Moderately concerned 5 6 7 Very confident

Please wait for the researcher

B.11 Brian favorites

Brian is taking a trip with his laptop. He doesn't want to copy all music onto his laptop because he is short on space, but he wants to have all of the songs on the playlist "Brian favorites".

Use the "File copy placement tool" to the left to change where files are stored in the house, and the iTunes and iPhoto applications to locate data if needed.

When you are finished with the task, push continue.

NEWPAGE RADIO BatmanUnderstandDifficulty Please rate, on a scale from 1 to 7, how difficult you found this task. 1 Not at all difficult 2 3 4 Moderately difficult 5 6 7 Very difficult

RADIO BatmanUnderstandConfidence Please rate, on a scale from 1 to 7, how confident you are that you correctly completed this task. 1 Not at all confident 2 3 4 Moderately concerned 5 6 7 Very confident

Please wait for the researcher

B.12 Home videos

Brian has been editing family videos for himself and Mary for the last few years. He and Mary are visiting his mother next week, and taking Mary's laptop with them. Mary doesn't want to take up space on her laptop for all of the home video files, just the final versions. Make sure they can show the final versions of the home videos on their trip.

Use the "File copy placement tool" to the left to change where files are stored in the house, and the iTunes and iPhoto applications to locate data if needed.

When you are finished with the task, push continue.

NEWPAGE RADIO BatmanUnderstandDifficulty Please rate, on a scale from 1 to 7, how difficult you found this task. 1 Not at all difficult 2 3 4 Moderately difficult 5 6 7 Very difficult

RADIO BatmanUnderstandConfidence Please rate, on a scale from 1 to 7, how confident you are that you correctly completed this task. 1 Not at all confident 2 3 4 Moderately concerned 5 6 7 Very confident

Please wait for the researcher

B.13 Rafting

Mary and Brian went on a rafting trip and took a group of photos, which Mary remembers they labeled as Rafting 2007. She wants to show her mother these photos on Mary's laptop. However, she doesn't want to take up space on her laptop for files other than the Rafting 2007 files. Make sure Mary can show the photos to her mother during her visit.

Use the "File copy placement tool" to the left to change where files are stored in the house, and the iTunes and iPhoto applications to locate data if needed.

When you are finished with the task, push continue.

NEWPAGE RADIO BatmanUnderstandDifficulty Please rate, on a scale from 1 to 7, how difficult you found this task. 1 Not at all difficult 2 3 4 Moderately difficult 5 6 7 Very difficult

RADIO BatmanUnderstandConfidence Please rate, on a scale from 1 to 7, how confident you are that you correctly completed this task. 1 Not at all confident 2 3 4 Moderately concerned 5 6 7 Very confident

Please wait for the researcher

B.14 Travelling Brian

Please wait for the researcher.

This task will not require you to make modifications to the system setting, only to answer the question.

Brian is taking a trip with his laptop. What data will he be able to access while on his trip? You should summarize your answer into two classes of data.

Use the "File copy placement tool" to the left to see where files are stored in the house, and the iTunes and iPhoto applications to locate data if needed.

Hint: Use the "Summary of files on devices" row.

TEXTBOX BrianTravelSeeAnswer Please enter the data Brian will be able to access on his trip. Be as precise as possible, but give the answer in terms of groups of files (no need to list individual files.) This answer should be summarized into two classes of files.

When you are finished with the task, push continue.

NEWPAGE RADIO BrianReliabilityDifficulty Please rate, on a scale from 1 to 7, how difficult you found this task. 1 Not at all difficult 2 3 4 Moderately difficult 5 6 7 Very difficult

RADIO BrianReliabilityConfidence Please rate, on a scale from 1 to 7, how confident you are that you correctly completed this task. 1 Not at all confident 2 3 4 Moderately concerned 5 6 7 Very confident

Please wait for the researcher.

B.15 Travelling Mary

Please wait for the researcher.

This task will not require you to make modifications to the system setting, only to answer the question.

Mary is taking a trip with her laptop. What data will she be able to access while on her trip? You should summarize your answer into two classes of data.

Use the "File copy placement tool" to the left to see where files are stored in the house, and the iTunes and iPhoto applications to locate data if needed.

Hint: Use the "Summary of files on devices" row.

When you are finished with the task, push continue.

TEXTBOX BrianTravelSeeAnswer Please enter the data Mary will be able to access on her trip. Be as precise as possible, but give the answer in terms of groups of files (no need to list individual files.) This answer should be summarized into two classes of files.

NEWPAGE RADIO BrianReliabilityDifficulty Please rate, on a scale from 1 to 7, how difficult you found this task. 1 Not at all difficult 2 3 4 Moderately difficult 5 6 7 Very difficult

RADIO BrianReliabilityConfidence Please rate, on a scale from 1 to 7, how confident you are that you correctly completed this task. 1 Not at all confident 2 3 4 Moderately concerned 5 6 7 Very confident

Please wait for the researcher.

C Deployment questions

C.1 Pre-install interview

Overview: Today we will ask you some general questions about the devices that you have and what data you store on them. We try to understand how you organize and manage your data, as well as how you backup and synchronize your data.

BACKGROUND: For each of the people living in the house: What are your Name/Age/Profession

DEVICES: Please list the digital storage devices in your home. This might include:

- USB sticks
- Computers
- Cell phone
- DVR
- Music players
- Gaming systems
- Hard drives
- Cameras
- Memory cards
- Online storage (Flickr, Facebook, Gmail)

– Work machines (computers at work that store home data)

For each device, please specify:

– What kind of device is it?

– What storage capacity does it have?

– When did you acquire it?

– What was the purpose of acquiring this device?

– How often do you use it?

– Who else uses it? How often do they use it?

– Where is it used? In which locations? How often in each place?

– What is it used for?

– Upsetness likert to parallel the data and data-device question

DATA GROUPS: Identify data groups that the user has.
Example: pictures, videos, work documents, email.
For each data group:

– How would you describe this data? What is the relationship with other data groups?

– What is the size of the data group?

DEVICE/DATA pair: For each device/data pair, ask:

– What is the relationship to data from the same data group stored on another devices? (E.g. identical, a subset of, an older version of, etc)

– Why is the data placed on this device? The purpose could be: as backup, so I can access it from work, so I have it with me on the cell phone, as primary storage, etc.

- How did the data get on the devices? Did you copy it manually? Did you use an automatic synchronization tool like FolderShare? How often does data get copied/written on this device? (Frequency and method).
- Why the method?
- What operations are done on the data? (read/ read-write/ backup; high-level operations as well)
- How upset would you be if this data were lost from this particular device? (Scale from 1 to 5) [Upsetness Likert]
- Think about some times when you need to access this data from this device. How annoying would it be if it were temporarily unavailable when you were trying to access it? (5 minutes, one hour, one day, for the weekend)

REVIEW of WORKFLOW: (From the previous questions, make sure we have a good understanding of the data flow.) Additional summary questions to ask as needed. The goal is to be able to go back in the weekly interviews and see how this changed.

- How long did it take to get your data recovery/management working? Was it easy or hard to set up?
- What would you like to change in your current work flow? What are you not satisfied with?
- How much time do you spend per day/week to synchronize/manage your data?
- What features/system would you like to have to help you manage your data?

UPSETNESS SCALE:
How upset would you be (were you) in this situation?

1	2 3 4	5
Dont care	Upset/Annoyed	Devastated

INCONVENIENCE SCALE:
How inconvenient is (was) this process?

1	2 3 4	5
Easy/convenient	Annoying but manageable	Visiting the DMV

SATISFACTION SCALE:
How satisfied are you (were you) with this system or process?

1	2 3 4	5
Unsatisfied	Meets my needs; acceptable	Highly satisfied

C.2 Weekly interview

During the study, we will conduct weekly interviews. The goal is to assess the effectiveness, its ease of use and the users satisfactions with Perspective. We will discuss events that have happened in the last week, and how the users usual data management workflow has changed through the use of Perspective.

Data management operations = any data operations that have to do with backing up, copying, moving data across devices.

MANAGEMENT TASKS: For the initial interview ask for examples in the past. Once deployed, use comment box entries or ask the question if we dont get comment box answers. All of these questions are inside or outside of Perspective.

- Were you able to access any files from more than one device?
- What data did you access from multiple devices?
  - From what devices did you use it?
  - Why did you need the data, and why on the devices?
  - How did you get access to the data?
  - How long did it take you to work out?
  - (Initial interview) How often does this happen?
- Did you not have access to a file when you needed it?

  - What data? Why did you need it? Why didnt you have it?
  - How did you solve it? How much did you care? (upsetness likert)
  - How long did it take to solve?
  - How long did you go without the data?
  - How confident are you that in the future (e.g, next week) this kind of problem will not happen again?
  - (Initial interview) How often does this happen?
- Did you have a device fail?
  - What device? How did it fail?
  - What data was on it?
  - How did you recover from the failure? How long did it take?
  - Did you lose data? How much did you care about the loss? (upsetness likert)
  - How much did you care about the failure? (upsetness likert)
  - (Initial interview) How often does this happen?
- Did you lose data?
  - What data? How was it lost?
  - How much did you care? (upsetness likert)
  - Did it change the way you are managing this data?
  - (Initial interview) How often does this happen?
- Did you recover data from a backup?
  - Were you successful?
  - How long did it take?
  - How much did you care? (upsetness likert)
- What if a device failed tomorrow? (Choose device)

    - How would you recover from the failure? How long would it take?
    - Would you lose data? How much would you care? (upsetness likert)
    - How much would you care about the failure? (upsetness likert)
  - Did you add a device to your home setup?
    - What device?
    - Why did you add it? What did you use it for?
    - What data did you move onto it and why?
    - What data did you create on it?
    - (Initial interview) How often does this happen?
  - Did you remove a device from your home setup?
    - What device?
    - Why did you remove it?
    - What data did you move off of it and why?
    - What data never made it off the device?
    - (Initial interview) How often does this happen?
  - Did you change the tags on your data? This includes directories, ID3 tags, Perspective metadata, etc.
    - What data did you change it on?
    - How did you change the tags?
    - Why did you change the tags?
    - How long did it take you?
    - (Initial interview) How often does this happen?
  - Did you change the way your data was backed-up or protected from failure?
    - In what way did you change the configuration?

  - Why did you change the configuration?
  - How long did it take you?
- Did you try to do a management task on your data and give up?
  - What was it?
  - What was difficult about it?
  - What did you do instead?
- Did you spend time on other data management tasks?
  - What was it?
  - How long did it take?

Choose instances of their workflow (taking photos, music, etc.) and ask whether they have done this task this week and how they did it.

FEEDBACK/CONFIDENCE:

- How confident are you that your data is adequately protected from failure? (confidence likert)
- How confident are you that your data will be accessible when you need it? (confidence likert)
- Do you check to make sure your files are correctly placed and up to date? How?

PERSPECTIVE - EVALUATION:

- How satisfied are you with Perspective? (sat likert)
- Overall, how intuitive was it to create new views/setup Perspective to do what you wanted?
- What functionality do you think Perspective is lacking? Which features do would you like to see added? (? E.g. devices must come in physical proximity to get synchronized)

DATA ORGANIZATION:

- What are you unhappy with about your data organization?
- What are you happy with about your data organization?
- How happy are you with your organization. (sat likert)

GENERAL QUESTIONS (some not applicable for initial interview):

- first week: How much time did you spend on data management this week? How much time in Perspective vs. outside Perspective?
- Which of your data groups are being managed by Perspective?
- Which data groups are not? Which data do you still manage manually or through previous/other methods? Why? (e.g. Perspective does not support the device, I use online storage, etc)
- first week: Overall management process satisfaction? (sat likert)
- How satisfied are you with managing your data through Perspective? (sat likert)

Bibliography

[1] Flickr Web Page. http://www.flickr.com. 16

[2] Marcos K. Aguilera, Robert E. Strom, Daniel C. Sturman, Mark Astley, and Tushar D. Chandra. Matching events in a content-based subscription system. *PODC.* (Atlanta, GA, 04–06 May. 1999), pages 53–61. ACM, 1999. 15

[3] R. Aipperspach, T. Rattenbury, A. Woodruff, and J. Canny. A Quantitative Method for Revealing and Comparing Places in the Home. *UBICOMP* (Orange County, CA, Sep. 2006), 2006. 9, 26

[4] Aperture Web Page. http://www.apple.com/aperture/. 16

[5] Avahi Web Page. http://avahi.org. 75

[6] Beagle web page, http://beagle-project.org, 2007. 1, 20

[7] Genevieve Bell and Joseph Kaye. Designing technology for domestic spaces: A Kitchen Manifesto. *Gastranomica*, **2**(2):46–62, 2002. 10

[8] Hugh Beyer and Karen Holtzblatt. *Contextual Design: Defining Customer-centered Systems.* Morgan Kaufmann Publishers, 1998. 8

[9] Sumeer Bhola, Yuanyuan Zhao, and Joshua Auerbach. Scalably supporting durable subscriptions in a publish/subscribe system. *DSN.* (San Francisco, CA, 22–25 Jun. 2003), pages 57–66. IEEE, 2003. 15

[10] Bonjour Web Page. http://www.apple.com/macosx/technology/-bonjour.html. 75

[11] Barry Brown and Louise Barkhuus. The Television Will Be Revolutionized: Effects of PVRs and Filesharing on Television Watching. *CHI* (Montreal, Canada, 2006), 2006. 10

[12] A. J. Bernheim Brush and Kori M. Inkpen. Yours, Mine and Ours? Sharing and Use of Technology in Domestic Environments. *UBICOMP 07*, 2007. 9, 35, 39, 48

[13] Antonio Carzaniga, David S. Rosenblum, and Alexander L. Wolf. Achieving Expressiveness and Scalability in an Internet-Scale Event Notification Service. *Nineteenth ACM Symposium on Principles of Distributed Computing (PODC2000)* (Portland, OR, Jul. 2000), pages 219–227, 2000. 15

[14] Mike Dahlin, Lei Gao, Amol Nayate, Arun Venkataramana, Praveen Yalagandula, and Jiandan Zheng. PRACTI Replication. *NSDI.* (May. 2006), 2006. 12, 80

[15] Maria R Ebling, Bonnie E John, and Mahadev Satyanarayanan. The importance of translucence in mobile computing systems. *CHI 2002*, 2002. 8, 152

[16] W Keith Edwards and Rebecca E Grinter. At home with ubiquitous computing: seven challenges. *Ubiquitous Computing International Conference (UBICOMP)* (Atlanta, GA, 2001), 2001. 9

[17] Facebook Web Page. http://www.facebook.com. 16

[18] David Frohlich and Robert Kraut. The Social Context of Home Computing. In . Springer, 2003. 9

[19] David M. Frohlich, Susan Dray, and Amy Silverman. Breaking up is hard to do: family perspectives on the future of the home PC. *International Journal of Human-Computer Studies*, **54**(5):701–724, May. 2001. 8, 9

[20] Filesystem in User Space Web Page. http://fuse.sourceforge.net/. 4, 88

[21] Roxana Geambasu, Magdalena Balazinska, Steven D. Gribble, and Henry M. Levy. HomeViews: Peer-to-Peer Middleware for Personal Data Sharing Applications. *SIGMOD.*, 2007. 14

[22] David K. Gifford, Pierre Jouvelot, Mark A. Sheldon, and James W. O'Toole Jr. Semantic file systems. *SOSP.* (Asilomar, Pacific Grove, CA). Published as *Operating Systems Review*, **25**(5):16–25, 13–16 Oct. 1991. 1, 20, 93

[23] Google desktop web page, http://desktop.google.com, Aug. 2007. 1, 20

[24] Google Documents Web Page. http://docs.google.com. 16

[25] Burra Gopal and Udi Manber. Integrating Content-based Access Mechanisms with Heirarchical File Systems. *OSDI.* (New Orleans, LA, Feb. 1999), 1999. 20

[26] Rebecca E Grinter, W Keith Edwards, Mark W Newman, and Nicolas Ducheneaut. The work to make a home network work. *European Conference on Computer Supported Cooperative Work (ESCW)* (Paris, France, 18–22 Sep. 2005), 2005. 9, 33, 35

[27] Richard G. Guy. *Ficus: A Very Large Scale Reliable Distributed File System.* PhD thesis, published as Ph.D. Thesis CSD-910018. University of California, Los Angeles, 1991. 13, 80, 81

[28] Richard G. Guy, John S. Heidemann, Wai Mak, Thomas W. Page Jr, Gerald J. Popek, and Dieter Rothmeier. Implementation of the Ficus replicated file system. *USENIXSummer.* (Anaheim, California), pages 63–71, 11–15 Jun. 1990. 98

[29] A Consumer's Eye View of Whole Home Storage, 2009. http://www.intelconsumerelectronics.com/Consumer-Electronics-3.0/Whole-Home-Storage.aspx. 10

[30] Kenton O Hara, April Slayden Mitchell, and Alex Vorbau. Consuming Video on Mobile Devices. *CRYPTO* (San Jose, CA, 2007), 2007. 9

[31] Debby Hindus. The importance of homes in technology research. *Cooperative Buildings (CoBuild)* (Pittsburgh, PA, 1999), 1999. 8

[32] Larry Huston, Rahul Sukthankar, Rajiv Wickremesinghe, M. Satyanarayanan, Gregory R. Ganger, Erik Riedel, and Anastassia Ailamaki. Diamond: A storage architecture for early discard in interactive search. *FAST.* (San Francisco, CA, 31 Mar.–02 Apr. 2004), pages 73–86. USENIX Association, 2004. 14, 144

[33] iPhoto Web Page. http://www.apple.com/ilife/iphoto. 16

[34] iTunes/iPod Web Page. http://www.apple.com/itunes/. 16

[35] Bonnie E. John, Len Bass, Maria-Isabel Sanchez-Segura, and Rob J. Adams. Bringing Usability Concerns to the Design of Software Architecture. *Engineering for Human-Computer Interaction* (Hamburg, Germany, 11–13 Jul. 2004), 2004. 8

[36] Alexandros Karypidis and Spyros Lalis. OmniStore: A system for ubiquitous personal storage management. *IEEE International Conference on Pervasive Computing and Communications.* IEEE, 2006. 12

[37] Robert J Logan, Sheila Augaitis, Robert B Miller, and Keith Wehmeyer. Living Room Culture – An Anthropological Study of Television Usage Behaviors. *Human Factors and Ergonomics Society 39th Annual Meeting*, pages 326–330, 1995. 10

[38] Dahlia Malkhi and Doug Terry. Concise Version Vectors in WinFS. *DISC.* (Cracow, Poland, Sep. 2005), 2005. 1, 20

[39] Catherine C Marshall. How People Manage Personal Information over a Lifetime. In . University of Washington Press, 2007. 9

[40] Catherine C Marshall. No Bull, No Spin: A comparison of tags with other forms of user metadata. *JDCL 2009* (Austin, TX, 15–19 Jun. 2009), 2009. 9

[41] Catherine C Marshall. Rethinking Personal Digital Archiving, Part 1: Four Challenges from the Field. *DLib Magazine*, **14**(3/4), Mar. 2008. 9, 52

[42] Catherine C Marshall. Rethinking Personal Digital Archiving, Part 2: Implications for Services, Applications, and Institutions. *DLib Magazine*, **14**(3/4), Mar. 2008. 9

[43] Catherine C. Marshall, Sara Bly, and Francoise Brun-Cottan. The Long Term Fate of Our Personal Digital Belongings: Toward a Service Model for Personal Archive. *Archiving 2006* (Springfield, VA, 2006), 2006. 9, 44, 50, 61, 64

[44] Michael Mateas, Tony Salvador, Jean Scholtz, and Doug Sorenzen. Engineering Ethnography in the Home. *CHI* (Vancouver, Canada, 1996), 1996. 8

[45] Windows Media Player Web Page. http://www.microsoft.com/windows/windowsmedia/player/10/default.aspx. 16

[46] C. Nass, B. Reeves, and G. Leshner. Technology and roles: a tale of two TVs. *Journal of Communication*, **46**(2):121–128, 1996. 34

[47] Edmund B. Nightingale and Jason Flinn. Energy-efficiency and storage flexibility in the Blue file system. *OSDI.* (San Francisco, CA, 06–08 Dec. 2004), pages 363–378. USENIX Association, 2004. 12

[48] Jon O'Brien, Tom Rodden, Mark Rouncefield, and John Hughes. At home with the technology: an ethnographic study of a set-top-box trial. *CHI*, 1999. 10, 48

[49] Antti Oulasvirta and Lauri Sumari. Mobile kits and laptop trays: managing multiple devices in mobile information work. *CHI* (San Jose, CA, 2007), 2007. 9

[50] Justin Mazzola Paluska, David Saff, Tom Yeh, and Kathryn Chen. Footloose: A Case for Physical Eventual Consistency and Selective Conflict Resolution. *IEEE Workshop on Mobile Computing Systems and Applications* (Monterey, CA, 09–10 Oct. 2003), 2003. 12, 84

[51] D. Stott Parker, Gerald J. Popek, Gerald Rudisin, Allen Stoughton, Bruce J. Walker, Evelyn Walton, Johanna M. Chow, David Edwards, Stephen Kiser, and Charles Kline. Detection of mutual inconsistency in distributed systems. *IEEE Trans. on Software Engineering*, **9**(3):240–247, May. 1983. 81

[52] Daniel Peek and Jason Flinn. EnsemBlue: Integrating distributed storage and consumer electronics. *OSDI* (Seattle, WA, 06–08 Nov. 2006), 2006. 9, 13, 14, 66, 79, 137, 141, 145, 147

[53] Picasa Web Page. http://picasa.google.com. 16

[54] Benjamin C. Pierce and Jerome Vouillon. *What's in Unison? A Formal Specification and Reference Implementation of a File Synchronizer*. Technical report MS-CIS-03-36. Dept. of Computer and Information Science, University of Pennsylvania, 2004. 13

[55] Peter R. Pietzuch and Jean M. Bacon. Hermes: A Distributed Event-Based Middleware Architecture. *International Workshop on Distributed Event-Based Systems* (Vienna, Austria), 2002. 15

[56] Rob Pike, Dave Presotto, Ken Thompson, and Howard Trickey. Plan 9 from Bell Labs. *United Kingdom UNIX systems User Group* (London, UK, 9–13 Jul. 1990), pages 1–9. United Kingdom UNIX systems User Group, Buntingford, Herts, 1990. 74

[57] Ansley Post, Petr Kuznetsov, and Peter Druschel. PodBase: Transparent storage management for personal devices. *International Workshop on Peer-to-peer Systems 2008* (Tampa, FL, 2008), 2008. 14

[58] Nuno Preguica, Carlos Baquero, J. Legatheaux Martins, Marc Shapiro, Paulo Sergio Almeida, Henrique Domingos, Victor Fonte, and Sergio

Duarte. FEW: File Management for Portable Devices. *Proceedings of The International Workshop on Software Support for Portable Storage*, 2005. 13

[59] Venugopalan Ramasubramanian, Thomas L. Rodeheffer, Douglas B. Terry, Meg Walraed-Sullivan, Ted Wobbler, Catherine C. Marshall, and Amin Vahdat. Cimbiosys: A Platform for content-based partial replication. *NSDI.* (Boston, MA, Apr. 2009), 2009. 13, 80, 84, 142

[60] Dave Randall. Living Inside a Smart Home: A Case Study. In . Springer London, 2003. 10

[61] David Ratner, Peter Reiher, and Gerald J. Popek. *Dynamic Version Vector Maintenance.* Tech report CSD-970022. Department of Computer Science, University of California, Los Angeles, 1997. 81, 87

[62] Robert W. Reeder, Lujo Bauer, Lorrie Faith Cranor, Michael K. Reiter, Kelli Bacon, Keisha How, and Heather Strong. Expandable grids for visualizing and authoring computer security policies. *CHI* (Florence, Italy, 2007), 2007. 16, 113, 117

[63] Eric Riedel, Christos Faloutsos, Garth Gibson, and Dave Nagle. Active disks for large-scale data processing. *IEEE Computer*, pages 68–74, Jun. 2001. 14

[64] Yasushi Saito and Christos Karamanolis. *Name space consistency in the Pangaea wide-area file system.* HP Laboratories SSP Technical Report HPL–SSP–2002–12. HP Labs, Dec. 2002. 13

[65] Brandon Salmon, Frank Hady, and Jay Melican. *Learning to Share: A Study of Sharing Among Home Storage Devices.* Technical Report CMU-PDL-07-107. Carnegie Mellon University, Oct. 2007. 23

[66] Brandon Salmon, Steven W. Schlosser, Lily B. Mummert, and Gregory R. Ganger. *Putting home storage management into perspective.* Technical Report CMU-PDL-06-110. Sep. 2006. 9

[67] M. Satyanarayanan. The evolution of Coda. *ACM Transactions on Computer Systems*, **20**(2):85–124. ACM Press, May. 2002. 12, 14, 66, 80, 81, 92, 141

[68] Bill Schilit and Uttam Sengupta. Device Ensembles. *IEEE Computer*, **37**(12):56–64. IEEE, Dec. 2004. 13, 71, 75

[69] Bob Sidebotham. VOLUMES – the Andrew file system data structuring primitive. *EUUGAutumn.* (Manchester, England, 22−24 Sep. 1986), pages 473–480. EUUG Secretariat, Owles Hall, Buntingford, Herts SG9 9PL, Sep. 1986. 12, 14, 81

[70] Alex Siegel, Kenneth Birman, and Keith Marzullo. Deceit: a flexible distributed file system. *USENIXSummer.* (Anaheim, California), pages 51–61, 11–15 Jun. 1990. 12

[71] Sumeet Sobti, Nitin Garg, Fengzhou Zheng, Junwen Lai, Yilei Shao, Chi Zhang, Elisha Ziskind, Arvind Krishnamurthy, and Randolph Y. Wang. Segank: a distributed mobile storage system. *FAST.* (San Francisco, CA, 31 Mar.–02 Apr. 2004), pages 239–252. USENIX Association, 2004. 12

[72] Craig A. N. Soules and Gregory R. Ganger. Connections: Using Context to Enhance File Search. *SOSP.* (Brighton, United Kingdom, 23–26 Oct. 2005), pages 119–132. ACM, 2005. 1

[73] Spotlight web page, http://www.apple.com/macosx/features/spotlight, Aug. 2007. 1, 20

[74] Peter Sutton, Rhys Arkins, and Bill Segall. Supporting Disconnectedness - Transparent Information Delivery for Mobile and Invisible Computing. *International Symposium on Cluster Computing and the Grid (CCGrid)*, 2001. 15

[75] Alex S. Taylor, Richard Harper, Laurel Swan, Shahram Izadi, Abigail Sellen, and Mark Perry. Homes that make us smart. *Personal and Ubiquitous Computing.* Springer London, 2006. 10

[76] Douglas B. Terry, Marvin M. Theimer, Karin Petersen, Alan J. Demers, Mike J. Spreitzer, and Carl H. Hauser. Managing update conflicts in Bayou, a weakly connected replicated storage system. *SOSP.* (Copper Mountain Resort, CO, 3–6 Dec. 1995). Published as *Operating Systems Review*, **29**(5), 1995. 13, 80, 81, 82, 92

[77] The UPnP Forum. http://www.upnp.org/. 75

[78] Amy Voida, Rebecca E Grinter, Nicolas Ducheneaut, W Keith Edwards, and Mark W Newman. Listening in: practices surrounding iTunes music sharing. *CHI* (Porland, OR, 2005), 2005. 9

[79] Bruce Walker, Gerald Popek, Robert English, Charles Kline, and Greg Theil. The LOCUS distributed operating system. *SOSP.* (Bretton Woods, New Hampshire). Published as *Operating Systems Review*, **17**(5):49–70, Oct. 1983. 12

[80] Markus Weiland and Raimund Dachselt. Facet Folders: Flexible Filter Hierarchies with Faceted Metadata. *CHI 2008* (Florence, Italy, 05–10 Apr. 2008), 2008. 20

[81] Windows 7 News: Windows 7 Libraries. http://windows7news.com/2009/04/07/windows-7-libraries/. 20

[82] WinFS 101: Introducing the New Windows File System, March 2007. http://msdn.microsoft.com/en-us/library/aa480687.aspx. 1, 20

[83] A. Woodruff, S. Augustin, and B. E. Foucault. Sabbath Day Home Automation: It's Like Mixing Technology and Religion. *CHI* (San Jose, CA, 2007), 2007. 10

[84] Ping Yee, Kirsten Swearingen, Kevin Li, and Marti Hearst. Faceted Metadata for Image Search and Browsing. *CHI*, 2003. 20, 74, 102