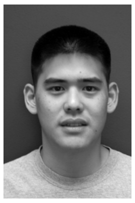
MICHAEL DEMMER, BOWEI DU, AND ERIC BREWER

TierStore: a distributed file system for challenged networks in developing regions

Michael Demmer is a PhD candidate at UC Berkeley. His research is on delay- and disruption-tolerant networking, distributed systems for unusual or challenged network environments, and application of technology in developing regions. He received his BS from Brown University.

demmer@cs.berkeley.edu



Bowei Du is a PhD candidate at UC Berkeley. His research is on distributed storage in delay-tolerant networks. He received his BS from Cornell University.

bowei@cs.berkeley.edu

Eric Brewer is a Professor of Computer Science at UC Berkeley who focuses on all aspects of Internet-based systems, including technology, strategy, and government. He leads the TIER research group on technology for developing regions, with projects in India, Ghana, and Uganda, and including communications, health, education, and e-government. He received an MS and PhD in EECS from the Massachusetts Institute of Technology and a BS in EECS from UC Berkeley and was recently elected to the National Academy of Engineering for leading the development of scalable servers.

brewer@cs.berkeley.edu

TECHNOLOGY HAS A GREAT ROLE TO play in developing regions, but we need approaches that can tolerate limited networking and power infrastructure. One promising model is to build applications around a file system interface that provides eventual consistency in these “challenged” network environments. Our resulting system, TierStore, hides much of the complexity of intermittency and simplifies the deployment of important applications such as email, Web caching, and wiki-based collaboration.

In many developing region settings throughout the world, there is an unmet need for robust information distribution applications. The limited communications infrastructure that exists in these environments means that simple information sharing systems can have a large impact. In fact, several projects have shown tangible results in the areas of health care, education, commerce, and productivity. As one example, data collection related to causes of child deaths in Tanzania led to a reallocation of resources and a 40% reduction in child mortality (from 16% to 9%) [1, 3].

However, the limited infrastructure also makes application deployment challenging. Wired networks are often either poor in quality or virtually nonexistent, cellular networks may be growing rapidly but remain a largely urban and costly phenomenon, and satellite networks provide good coverage but are prohibitively expensive. Many of these networking approaches further suffer from periodic outages owing to unreliable grid power. Thus any software system targeted toward these environments must deal with intermittent connectivity and potentially long-lasting network partitions and failures.

In response to the combination of application needs and the complexity of programming intermittency-tolerant applications, we have developed a distributed storage system called TierStore [4]. TierStore is a new approach to designing and deploying information distribution applications which aims to overcome the connectivity challenges in developing countries, while at the same time making it easy to port existing applications and develop new ones.

This work is part of the Technology and Infrastructure for Emerging Regions (TIER) [7] research effort at UC Berkeley. The aim of the TIER project is to address challenges in bringing the information technology revolution to the masses of the developing regions of the world. Unfortunately, most projects that aim to do this today rely on technology that was developed for the affluent world, yet these imported technologies fail to address key challenges in cost, deployment, power consumption, and support for semi- and illiterate users. Instead, our approach is to explore the development of novel solutions to technical challenges that explicitly take the needs of developing countries into account.

Background

In developing TierStore, we were inspired by several existing projects that deal with poor and intermittent connectivity. For example, the Wizzy Digital Courier system [9] distributes educational content among schools in South Africa by delaying dialup access until night time, when rates are cheaper. As another example, DakNet [6] provides email and Web connectivity by copying data to a USB drive or hard disk and then physically carrying the drive among locations that have no traditional network connectivity, sometimes via motorcycles. Finally, the TEK [8] disconnected Web search engine allows users to search the Web using SMTP as the underlying protocol, which can buffer communication across network outages.

These examples help underscore the value of information distribution applications in developing regions, but they all essentially started from scratch and thus use ad hoc solutions with little leverage from previous work. Instead, the goal of TierStore is to be a general-purpose framework that can abstract away most of the complications related to working with intermittent networks.

Asynchronous	Disconnection-tolerant	Synchronous
	offline WWW	
email	information portals	
voicemail	e-government services	VOIP
bulk data copy	survey/data collection	video chat
	medical records	

TABLE 1: AN APPLICATION TAXONOMY WITH RESPECT TO INTERMITTENCY

Table 1 gives a rough breakdown of example applications and their behavior with respect to intermittent networks. At the end points, "asynchronous" applications already work well in disconnected environments, whereas "synchronous" applications fundamentally require end-to-end connectivity and just cannot function during network outages. However, the large class of "disconnection-tolerant" applications can potentially work well in a disconnected fashion, yet their implementations are limited by the requirements of the underlying software platforms on which they are implemented. For example, many such services require Internet connectivity simply because they have been written as Web applications, whereas they could potentially function well even when disconnected. Our goal with the TierStore system is to make it easy to adapt these applications to work well in an intermittent environment.

One step toward this goal comes in the form of Delay-Tolerant Networking (DTN) [2]. DTN is a new approach to networking in challenged environments that seeks to address the shortcomings of traditional Internet protocols in some scenarios. Specifically, there are many cases in which it is difficult to maintain the kind of reliable, low-latency network connection needed by TCP/IP-based protocols. In these cases DTN can route network messages across a variety of different transports such as peer-to-peer wireless connections, dialup links, or physically carried flash drives or PDAs. Furthermore, DTN is based around application-defined data objects called "bundles" (not packets or circuits) and can deal with outages by storing messages in the network core to wait for connectivity to be restored. DTN also offers new approaches to routing, quality of service, and reliability based on custody transfer which help to deal with many of the problems that exist in challenged network environments.

Yet applications need more than just a messaging service. Running while disconnected implies that applications need to have local storage to respond to user requests. Distributing information between instances mandates conventions for object naming and organization to ensure that multiple sites remain in sync with each other. Operations that modify the system state need to be logically ordered, and potentially conflicting operations need to be identified and resolved. Perhaps most importantly, adapting existing applications to DTN environments would require significant rewriting to use the DTN-specific messaging APIs. TierStore is aimed squarely at addressing these application needs while leveraging the existing advantages of the DTN framework.

How TierStore Works

To address these needs, TierStore implements a replicated file system interface, and applications interact with the system using the standard POSIX APIs. This decision means that existing applications that are already written to use the file system for interprocess communication can be adapted with relatively few changes, while developers creating new applications can leverage their familiarity with the existing APIs and use a wide range of programming environments and languages to interact with the system. Figure 1 shows the TierStore system components.

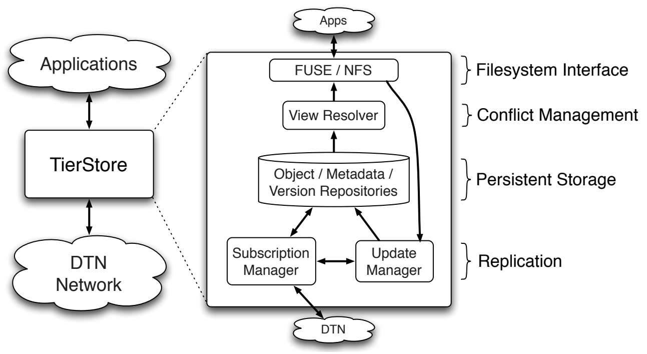


FIGURE 1: TIERSTORE SYSTEM COMPONENTS

Unlike NFS or CIFS, there is no central server that stores file data. Instead, each TierStore node keeps a copy of the data and uses a lazy distribution protocol to forward updates among nodes using DTN bundles. This means that replicated file data is available for access even when the network is down, and local filesystem interactions need not consume valuable bandwidth.

REPLICATION

File system modifications, such as writing some data to a file, are encoded as update messages. These updates are immediately applied to the local node so that applications see the effects of their operations in the file system. They are also forwarded to the subscription manager that determines how to distribute the updates to other nodes.

To enable fine-grained data sharing, the files and directories in the TierStore file system are divided into non-overlapping subsets called publications. Publications define the units of replication between TierStore installations, and they are defined in application-specific ways. For example, an individual publication might be a user's mailbox, files from a particular Web site, or a set of data collection samples from a specific region. TierStore nodes then subscribe to a set of publications, indicating that they want to receive updates to the relevant files. Subscribing is the TierStore equivalent to mounting a portion of the file system, and therefore file data in a publication is replicated only to the set of subscribed nodes.

TierStore uses DTN for all internode communication, meaning that it can leverage its range of network transports and in-network message queuing features. Thus TierStore need not be concerned with the details of how updates are communicated, but instead it can queue an update message for transmission whenever the network becomes available, using whatever transmission mechanism is most appropriate for the environment.

To help distribute data efficiently over low-bandwidth links, each TierStore node is configured as part of a multicast-like distribution tree in the DTN network. Each publication can be thought of as a multicast group, so updates need only be transmitted once across a network link in the tree and are reforwarded down the tree, eventually reaching all subscribed nodes. In our early deployments, this distribution tree was manually configured, as the number of nodes was fairly small, but we are currently working on a new DTN multicast implementation to automate this process.

HANDLING CONFLICTS

Alice	Bob
Network is unavailable ...	
$ echo "red" > /tierstore/foo	$ echo "blue" > /tierstore/foo
$ ls /tierstore	$ ls /tierstore
foo	foo
Network is available ...	
$ ls /tierstore	$ ls /tierstore
foo foo.#bob	foo foo.#alice
$ cat foo	$ cat foo
red	blue
$ cat foo.#bob	$ cat foo.#alice
blue	red

FIGURE 2: DEFAULT CONFLICT HANDLER. DISCONNECTED USERS ALICE AND BOB MAKE EDITS TO THE SAME FILE. WHEN THEY RECONNECT, THE OTHER'S EDITS WILL BE VISIBLE AS A CONFLICT FILE IN THE FILE SYSTEM.

Since TierStore nodes might be disconnected for long periods of time, they must be able to modify the filesystem state while disconnected, so applications need some way of handling concurrent updates. Yet long outages mean that traditional approaches such as file locking will not work well. Instead, we allow arbitrary operations to occur and then detect (and possibly resolve) conflicts when nodes return into connectivity.

The first (and best) way to handle conflicts is to avoid them in the first place. TierStore only considers conflicts on a per-file basis, so updates to different files or directories are independent and do not conflict. Thus applications that partition their data into separate directories or use uniquely named files that are not updated at different parts of the network are thereby conflict-free. Many of the applications we have ported to TierStore naturally fall into this category.

When conflicts are unavoidable, applications can register custom handlers to resolve the situation. These handlers are able to look at conflicting versions of a file and arbitrarily rename, merge, or modify them to deal with the conflict. If there is no custom resolver, a default policy appends each conflicted filename with .#X, where X is the identity of the node that generated the conflict. This approach allows applications to see both versions of the conflicted file, similarly to how CVS allows multiple versions of a file to simultaneously exist on different branches. Applications can then resolve the conflict later at any point in the network.

However, one subtle aspect of this default policy is that file operations that occur at a particular node are presented unmodified to applications that are running at that node (i.e., without the .#X extension). This does mean that the displayed filesystem structure can vary at different locations in the network, but it also has the important side effect that nodes always "see" the files they have generated and modified locally, regardless of any conflicting updates that may have occurred at other locations (see Figure 2). This is an important decision that helps when porting unmodified applications, since their local file modifications do not suddenly disappear if another node makes a conflicting update to a file. It also means that application state remains self-consistent even in the face of conflicts and, most importantly, is sufficient to handle many applications without needing to write a custom conflict resolver.

Using TierStore

We have adapted several commonly used applications for use with TierStore to validate our system: an IMAP email service, a shared offline Web cache, and a shared wiki collaboration system. These three applications represent a range of requirements. Email in IMAP folders is accessed by a single user, but the folders may be replicated to many different computers. A Web cache is shared by many users but is read-only. A wiki system is shared by many users and has the potential for many conflicting writes from users editing overlapping parts of the wiki.

To support a shared email service with the TierStore system, we used a stock IMAP server configured to store mail content in maildir format. The maildir storage format is ideal for use with TierStore because each mail message is stored as a separate file and the metadata associated with the message is encoded in the message file name. Mail folders are thus mapped to directories in the TierStore file system, and each user mailbox is placed in a separate publication. This allows each computer in the network to elect to replicate only some of the users' mailboxes. To handle benign conflicts in state (e.g., flagging a message on one computer and marking it read on another), we wrote a conflict resolver that takes conflicting state flags and resolves them to be the union of the flags.

For push-based shared Web cache functionality, we took an existing offline-enabled Web cache, WWWOFFLE, and configured its cache directory to point to a TierStore shared directory. For a selected set of Web sites, we populate the cache directory with a Website crawl of commonly accessed reference Web sites. This model suits the needs of organizations that create local mirrors of online references such as Wikipedia. Using the TierStore system allows administrators to integrate Web content mirrors from many alternative sources into the same system. A bulk load of content via media such as DVDs can be supplemented by small incremental deltas pushed over conventional Internet access.

Finally, we have ported an existing piece of wiki software, PmWiki, to TierStore as well as our created own wiki software [5], which leverages TierStore as its storage back end. PmWiki stores its pages as individual files on the local file system. This matches the semantics of TierStore quite well, because conflicts in the file system map to edit conflicts at the page level; page-level conflicts are well understood by users of wikis, since they already occur because of delays between when a page is loaded in the browser and when the edit is saved to the server. Using the default conflict resolver, users of PmWiki will see edit conflicts as specially named pages on the wiki site. In our own wiki software, we implemented a resolver that performs a text-based merge of the conflicting pages.

Next Steps

On the research front, we are continuing to push TierStore in two directions. The first is focused on the networking layer to develop a robust publish/subscribe-based distribution network that functions well in DTN environments. The next is focused on developing an easy-to-use SQL interface to support disconnection-tolerant Web applications that interface with a database instead of the file system.

We are also continuing to work on several TierStore deployments in developing countries. One example is supporting community radio stations in Guinea Bissau, a small West African country characterized by a large number of islands and poor infrastructure. For many of the residents, their main information source comes from small radio stations that produce and broadcast local content. TierStore is being used to help bridge the communication barriers between the different islands and distribute content from these stations throughout the country. We are also currently doing a pilot deployment in Senegal, where TierStore will be used to link student medical records between two schools and a local hospital.

In general, our initial results from working on TierStore are encouraging, and we hope to gain additional insights through more deployment experience.

REFERENCES

[1] E. Brewer, M. Demmer, B. Du, M. Ho, M. Kam, S. Nedevschi, J. Pal, R. Patra, S. Surana , and K. Fall, “The Case for Technology in Developing Regions,” *IEEE Computer* 38(6), 25–38 (June 2005).

[2] V. Cerf, S. Burleigh, A. Hooke, L. Torgerson, R. Durst, K. Scott, K. Fall, and H. Weiss, “Delay-Tolerant Networking Architecture,” RFC 4838, April 2007.

[3] D. de Savigny, H. Kasale, C. Mbuya, and G. Reid, in *Fixing Health Systems* (International Development Research Centre Books, Ottawa, 2004).

[4] M. Demmer, B. Du, and E. Brewer, “TierStore: A Distributed File System for Challenged Networks in Developing Regions,” in *FAST ’08: 6th USENIX Conference on File and Storage Technologies* , pp. 35–48 (Feburary 2008).

[5] B. Du and E. Brewer, “DTWiki: A Disconnection and Intermittency Tolerant Wiki,” in *17th Annual International WWW Conference* (April 2008).

[6] A.S. Pentland, R. Fletcher, and A. Hasson, “DakNet: Rethinking Connectivity in Developing Nations,” *IEEE Computer* (January 2004).

[7] Technology and Infrastructure for Emerging Regions (TIER) Research Group: http://tier.cs.berkeley.edu/.

[8] W. Thies, J. Prevost, T. Mahtab, G. Cuevas, S. Shakhshir, A. Artola, B. Vo, Y. Litvak, S. Chan, S. Henderson, M. Halsey, L. Levison, and S. Amarasinghe, “ Searching the World Wide Web in Low-connectivity Communities,” in *Proceedings of the 11th International World Wide Web Conference, Global Community Track* (May 2002).

[9] Wizzy Digital Courier: http://www.wizzy.org.za/.