A Research Status Report on Adaptation for Mobile Data Access

Brian D. Noble and M. Satyanarayanan
School of Computer Science
Carnegie Mellon University
{bnoble,satya}@cs.cmu.edu

Abstract

Mobility demands that systems be *adaptive*. One approach is to make adaptation transparent to applications, allowing them to remain unchanged. An alternative approach views adaptation as a collaborative partnership between applications and the system. This paper is a status report on our research on both fronts. We report on our considerable experience with *application-transparent adaptation* in the Coda File System. We also describe our ongoing work on *application-aware adaptation* in Odyssey.

1. Introduction

Our research group has been investigating issues pertinent to mobile data access since 1990. This paper summarizes our experience so far and outlines our research agenda. The primary vehicle for our research has been the *Coda File System*, a descendant of AFS that provides mobile clients with shared file access. More recently, we have begun work on *Odyssey*, a platform for investigating a broader class of issues in mobile data access.

We begin by discussing certain intrinsic constraints of mobility. To successfully meet these constraints, mobile systems must be *adaptive*. We discuss two alternative approaches to incorporating adaptivity, and discuss the status of our research on each approach.

The brevity of this status report and its focus on adaptation preclude discussion of many other important aspects of Coda and Odyssey. The guide to further reading at the end of this paper provides pointers to important aspects of Coda such as server replication, optimistic replica control, conflict resolution, and isolation-only transactions, as well as dynamic sets in Odyssey.

This research was supported by the Air Force Materiel Command (AFMC) and ARPA under contract number F196828-93-C-0193. Additional support was provided by the IBM Corporation, Intel Corporation and AT&T Corporation. The views and conclusions expressed in this paper are those of the authors, and should not be interpreted as those of the funding organizations or Carnegie Mellon University.

2. Constraints of Mobility

Mobile clients face many challenges in accessing data from servers. Because the mobile client must be small and light, it is typically resource-poor in relation to a desktop client. Network connectivity, especially via wireless media over large areas, tends to vary widely in bandwidth, latency, reliability, and cost. Limited power often requires activities to be deferred, avoided, or slowed in order to prolong battery life. The relative costs of accessing fixed distributed services change as the client moves. Finally, the physical security of a mobile client is low relative to desktop machines, increasing the likelihood of loss, theft, or destruction. It should be emphasized that these constraints are intrinsic to mobility: they will not be alleviated by hardware advances.

3. Adaptation: the Key to Mobility

The lack of local resources and physical security argues for reliance on servers. However, the lack of reliable, cheap communication as well as the variable costs to access services argue for self-reliance on the part of mobile clients. The challenge for mobile data access is to strike an appropriate balance between these two competing concerns. This balance is not a static one. As the circumstances of a mobile client change, it must react and repartition duties between client and server. In other words, it must be adaptive. Such adaptation may occur anywhere along a spectrum characterized by two extremes.

At one extreme, adaptivity is entirely the responsibility of individual applications. This means that there is no single point in the system to resolve the potentially incompatible resource demands of different applications. It also means that there is no way to enforce limits on resource usage.

At the other extreme, the burden of adaptation is entirely borne by the system. We refer to this as *application-transparent adaptation*. In this model, the system provides the focal point for resource arbitration and control. The central advantage to this approach is that it is completely backward-compatible. Existing applications continue to work even when mobile.

While application-transparent adaptation is adequate in many cases, there are circumstances where application-specific knowledge is required. *Application-aware adaptation* enables use of such specialized knowledge while retaining the system as the focal point of resource control. Such an approach permits individual applications to determine how best to adapt, but allows centralized monitoring of resources, and effective enforcement of decisions regarding their usage. This represents the range of approaches between the two extremes.

4. Coda: Application-Transparent Adaptation

We have explored application-transparent adaptation extensively through our work in the Coda File System[3, 11]. Coda provides clients, particularly mobile ones, with highly available access to files. Like AFS, Coda presents a single, global namespace to clients organized in volumes[15], which are subtrees of the namespace. Applications running on Coda clients use the standard UNIX file system interface. Desktop applications can continue to run on mobile clients without modification. The client cache manager, *Venus*, is solely responsible for coping with the consequences of mobility.

Coda has been in active use for five years, and has proved to be a valuable testbed for exploring application-transparent adaptation. Coda clients are in regular use over a wide range of networks such as 10 Mb/s ethernet, 2 Mb/s radio, and 9600 baud modems. In following sections, we describe how Coda deals with the best and worst possible network conditions, and then discuss how it adapts to conditions between these end points.

4.1. Strongly Connected Operation

As a starting point in understanding how Venus adapts to varying network conditions, we first explore the best case; high-quality, fast LANs. In such a situation, Venus is said to be *strongly connected*. When strongly connected, a Coda client behaves much like an AFS client. When an application opens a file in Coda, Venus checks to see if the file is already cached. If it isn't, Venus fetches the file from a server to its local disk cache. It then services reads and writes on behalf of applications.

When a client caches a file from the servers, it also obtains a *callback* – a promise to be told if the file is updated by another client. When a dirtied file is closed, a copy of the new file contents is sent back to the servers. The servers notify any clients who had callbacks for that file that it has changed. This is known as a *callback break*. Experience shows that this approach to maintaining file-cache coherence offers excellent scalability and performance.

4.2. Disconnected Operation

In the worst case, Venus cannot contact any server; we say that such a client is *disconnected*. Venus continues to service reads and writes to files that are in-cache; reads and writes on files that are not cached cannot be serviced, and Venus returns an error for them. To reduce the likelihood of cache misses, Venus uses a mechanism called *hoarding*[3] to augment LRU cache management. Through hoarding, a user can provide advice on what files are likely to be important in the future. Periodic *hoard walks* ensure that files predicted to be useful through a combination of LRU and hoard priority remain in the cache in anticipation of unexpected disconnections.

During disconnected operation, updates are persistently logged by Venus, using a variety of optimizations to reduce resource consumption. For example, if a file is created, renamed, and later deleted, none of those records need be saved; it is as if none of the operations ever happened. When connectivity is reestablished, Venus replays these updates at the servers through *reintegration*.

4.3. Weakly Connected Operation

There are a broad range of conditions between strongly connected and disconnected operation. Coda users operate clients over 2 Mb/s radio links, and over modems as slow as 9600 baud. As network bandwidth decreases, the importance of reordering or delaying network traffic to preserve the illusion of strong connectivity increases[1].

To preserve the strongly-connected illusion, Venus endeavors to satisfy most demand cache misses as soon as possible, and delays other traffic as necessary. Faithful to the end-to-end argument, these decisions are made at as high a level in the system as possible. The question of how to reschedule network traffic is revisited as available network quality changes. Adaptive decisions are made in three key areas; cache coherence, reintegration, and demand cache fetches. Each of these is discussed briefly in the following sections.

4.4. Cache Coherence

While a client is disconnected, it may miss callback breaks for cached files. Thus, upon reconnection, each file is suspect and must be revalidated. On a LAN, this overhead is small; it can be expensive on a slower network. The intermittent connectivity typical of wireless networks exacerbates this problem.

To alleviate the costs of revalidation in low bandwidth situations, Coda maintains two kinds of callbacks; individual files, and full volumes. Volume callbacks are acquired only when the cached state of a volume is known to be coherent with that on the servers. Along with the volume callback, a client acquires a *version stamp* for the volume. The version stamp is incremented at the servers with each change to any file or directory in that volume. A client with a volume callback is promised that it will be notified of any change to the volume.

On slow networks, the use of volume stamps dramatically reduces the time for validation after reconnection. Instead of validating each object individually, Venus first compares volume stamps. In the common case, when nothing has changed, this single RPC implicitly validates all objects from a volume. If bandwidth is plentiful, Venus can later obtain individual object callbacks to minimize false invalidations,

4.5. Reintegration

When weakly connected, Venus uses a mechanism called *trickle reintegration* to strike a balance between prompt propagation and indefinite delay, thus preserving scarce network bandwidth for demand traffic. In trickle reintegration, updates are logged as if Venus were disconnected and propagated asynchronously. Venus ensures that no demand fetch is impacted for more than a fixed time bound, t, by trickle reintegration. It does this by estimating the available bandwidth, and propagating only an appropriate prefix of the log. Trickle reintegration uses an age-based technique to strike a balance between prompt propagation and preserving the effectiveness of log optimizations.

4.6. Demand Cache Fetches

Some cache misses are more important than others. For example, an author may be willing to use a different font file rather than waiting many minutes for a missing one to be fetched over a slow network. But for a critical file, he may be willing to wait a long time. To capture this notion, Venus incorporates a model of *user patience*, with the hoard priority of a file indicating its importance. Tools exist to help users calibrate hoard file priorities, as well as to view recent cache misses.

When a cache miss occurs, Venus estimates the time it would take to fetch the file. If the estimate exceeds the patience threshold for that file, Venus returns an error rather than servicing the miss. The user patience model is the source of adaptivity in cache miss handling. It maintains usability at all bandwidths by balancing the two factors that intrude upon transparent handling of misses: long fetch delays and the need for user advice.

4.7. Deployment Status

Coda was first released for general use in 1990, relying only on server replication. Support for disconnected operation was added in 1991. Support for weakly connected operation was added in stages between 1993 and 1995. Coda currently serves a community of about 40 users, 25 of whom use Coda for their day-to-day file storage needs. This user community stores about 4GB of data in Coda, and actively uses disconnected and weakly-connected operation.

5. Odyssey: Application-Aware Adaptation

Our work in Coda has shown application-transparent adaptation to be an effective technique. It has several advantages, especially that many applications written for desktop environments need not be changed at all; they will continue to work largely as intended in a mobile environment. However, there are situations where the system cannot be solely responsible for coping with mobility. To address this shortcoming, we are implementing a platform for mobile computing called Odyssey. In the following sections we describe our early work with this project.

5.1. Fidelity: the Basis for Adaptation

We define *fidelity* as the degree to which a copy of data presented for use matches the reference or 'true' copy. In general, fidelity has many dimensions. One well-known, universal dimension is consistency. Other dimensions depend on the type of data in question. For example, video data has at least two additional dimensions: frame rate and image quality of individual frames. Spatial data, such as topographical maps, have dimensions of minimum feature size or resolution. For telemetry data, appropriate dimensions include sampling rate and currency.

The dimensions of fidelity are natural axes of adaptation for mobility. But adaptation cannot be solely determined by the type of data; it also depends on the application. As we show in the next section, different applications using the same data may make different tradeoffs among dimensions of fidelity.

5.2. Why Involve Applications?

Consider a movie stored on a server, and two applications accessing that video stream from a mobile client. The first application is a video playback application, *player*, and the second, *editor*, is a video scene editor. These two applications must make different fidelity tradeoffs in accessing the same video stream. No single policy can satisfy them both.

The player's primary goal is to preserve correspondence between movie time and real time. A secondary goal is to play the movie at the original frame rate, resolution, and image quality. In times of plentiful resources, the player can indeed meet both goals. However, when network bandwidth becomes scarce, the player may have to sacrifice its secondary goal in order to meet its primary goal. Thus, it may choose to switch to a black-and-white stream at full frame rate, to drop frames, or otherwise reduce the bandwidth requirements of the stream. To guard against total disconnection, the player may even hoard a very low quality version of the movie.

The editor's main goal is very different from that of the player; it must ensure that the user sees every frame of the video stream to allow precise editing. To do this, the editor is willing to relax the correspondence between movie time and real time. Thus, when network bandwidth decreases, the editor will access the movie at a rate slower than real time to avoid dropping frames.

It is hard to see how any single operating system policy can adequately service both of these applications' needs, even though they are accessing exactly the same data. Regardless of the system's decisions, either the player or the editor – and quite possibly both – will not be satisfied. No system can be clever enough to anticipate and satisfy every application's needs. On mobile machines, where the environment is unpredictable, such unsatisfactory service will be even more evident. Only with the active participation of applications can scenarios such as the above be satisfactorily handled. Hence the need for application-aware adaptation.

5.3. How to Involve Applications

How best can we involve applications in adaptation? Since applications react to changes around them, we must define the *environment* to which applications adapt. We must then provide mechanism for applications to become aware of how their environment changes. Finally, we need a mechanism to allow applications to inform the system of their desired adaptations; the underlying system must then resolve the potentially conflicting demands of various applications.

We define the environment of an application by the *resources* available to the client host on which the application runs. The availability of these resources are each measured by the underlying system in some unit appropriate to the resource itself. For example, one resource might be the network bandwidth on a particular connection, measured in bytes per second. Another might be battery power remaining to the client, measured in minutes. All of the resources together constitute an application's environment.

To become aware of changes to the environment, an application first registers its interest in particular resources. It does so by specifying, for each resource in which it is interested, the acceptable lower and upper bounds on the availability of that resource. It also registers an upcall procedure. If a specified resource leaves the stated tolerance bounds, the upcall procedure is invoked with the resource's new availability. In this way, applications can track only those facets of the environment in which they are interested, while amortizing the cost of monitoring the environment over all interested applications.

When an application is told of a change in resource availability, it must adapt its access. Adaptation consists of varying the fidelity of data presented to the user. Since fidelity is dependent on the kind of data being presented, the methods of adaptation must also be type-specific. To support these type-specific methods, we provide a general mechanism to communicate with a portion of the system that understands that type.

5.4. Architectural Support

We have designed, and are implementing, an architecture to support this notion of application-aware adaptation.

There are two main pieces of this architecture. First, we must add some notion of type to the distributed store. Second, we must have the mechanism at the mobile client to provide both type-specific support for each type in the store as well as a single point of resource arbitration.

Odyssey, like Coda, provides its clients with a single, global namespace comprised of volumes. Unlike Coda, these volumes carry with them a notion of type. Each object within one of these typed volumes, or *tomes*, is of the same type. While there may be many tomes of a given type, each tome has a single type associated with it, and thus provides a type to each item in the store.

At clients there is a cache manager, much like Coda's Venus, which services operations in the Odyssey store on behalf of applications. The Odyssey cache manager is structured into two logical pieces: the *viceroy*, providing generic support for all objects in the store, and a collection of *wardens*, one per type, each providing support for objects of that particular type.

The viceroy's most important task is to act as the single point of resource control in the system; each warden is subordinate to it. In this role, the viceroy is responsible for monitoring the availability of resources, and notifying applications of changes to them as appropriate.

The wardens are responsible for implementing the access methods on objects of their type – both the standard UNIX operations as well as any type-specific ones. The wardens are also responsible for implementing any type-specific operations to change the fidelity at which data is cached and presented to the user.

To date, we have implemented a simple prototype of the Odyssey architecture, and the wardens, servers, and appilcations for video and geographical data. This prototype is rudimentary in many respects, but suggests the approach is valid, both in terms of the overhead in resource monitoring as well as the additional complexity brought on by involving applications in adaptation. However, these questions remain open; we plan on exploring them more fully through a refined implementation.

6. Conclusion

Mobile clients face many challenges. These challenges render adaptation the key to mobile data access. There are two approaches to adaptation: application-transparent, which we have explored in the Coda File System; and application-aware, which we are exploring in Odyssey

Our experience with Coda confirms that application-transparent adaptation is indeed effective in many cases. Many desktop applications are able to run unmodified in Coda over a wide range of network quality. The system has proven effective in actual use by a user community.

In circumstances where the Coda approach is no longer adequate, we believe that application-aware adaptation is the correct strategy. Odyssey is our vehicle for exploring this hypothesis. An early prototype of Odyssey shows this approach to be promising, and we are currently implementing a more complete system.

7. Acknowledgements

This work builds upon the contributions of many past and present Coda and Odyssey project members, including Jay Kistler, Puneet Kumar, David Steere, Lily Mummert, Maria Ebling, Hank Mashburn, Josh Raiff, Qi Lu, Morgan Price and Bob Baron. Perhaps the most important contribution of all has been made by the Coda user community, through its bold willingness to use and help improve an experimental system.

8. Further Reading

This paper provides only the briefest overview of our work. A detailed annotated guide may be found on the World Wide Web at this URL:

```
http://www.cs.cmu.edu/afs/cs.cmu.edu/
        project/coda/Web/coda.html
```

Given below are the highlights from this list:

- Server replication [11]
- Conflict resolution [4, 5]
- Disconnected operation [3, 12]
- Weakly connected operation [8, 7]
- Isolation-only transactions [6]
- Tools and building blocks [9, 2, 13]
- Odyssey design [14, 10].
- Dynamic sets [17, 16]

References

[1] EBLING, M., MUMMERT, L., AND STEERE, D. Overcoming the Network Bottleneck in Mobile Computing. In *Proceedings of the IEEE Workshop on Mobile Computing Systems and Applications* (Santa Cruz, CA, December 1994).

[2] EBLING, M., AND SATYANARAYANAN, M. SynRGen: An Extensible File Reference Generator. In *Proceedings of the 1994 ACM SIGMETRICS Conference* (Nashville, TN, May 1994).

[3] KISTLER, J. J., AND SATYANARAYANAN, M. Disconnected Operation in the Coda File System. *ACM Transactions on Computer Systems 10*, 1 (February 1992).

[4] KUMAR, P., AND SATYANARAYANAN, M. Log-Based Directory Resolution in the Coda File System. In *Proceedings of the Second International Conference on Parallel and Distributed Information Systems* (San Diego, CA, January 1993).

[5] KUMAR, P., AND SATYANARAYANAN, M. Flexible and Safe Resolution of File Conflicts. In *Proceedings of the USENIX Winter 1995 Technical Conference* (New Orleans, LA, January 1995).

[6] LU, Q., AND SATYANARAYANAN, M. Improving Data Consistency in Mobile Computing Using Isolation-Only Transactions. In *Proceedings of the Fifth IEEE HotOS Topics Workshop* (Orcas Island, WA, May 1995).

[7] MUMMERT, L. B., EBLING, M. R., AND SATYANARAYANAN, M. Exploiting Weak Connectivity for Mobile File Access. In *Proceedings of the 15th Symposium on Operating System Prinicples* (Copper Mountain, CO, December 1995).

[8] MUMMERT, L. B., AND SATYANARAYANAN, M. Large Granularity Cache Coherence for Intermittent Connectivity. In *Proceedings of the 1994 Summer USENIX Conference* (Boston, MA, June 1994).

[9] MUMMERT, L. B., AND SATYANARAYANAN, M. Long Term Distributed File Reference Tracing: Implementation and Experience. Tech. Rep. CMU-CS-94-213, Carnegie Mellon University, November 1994.

[10] NOBLE, B. D., PRICE, M., AND SATYANARAYANAN, M. A Programming Interface for Application-Aware Adaptation in Mobile Computing. In *Proceedings for the Second USENIX Symposium on Mobile and Location-Independent Computing* (Ann Arbor, Michigan, April 1995). Also as technical report CMU-CS-95-119, School of Computer Science, Carnegie Mellon University.

[11] SATYANARAYANAN, M., KISTLER, J. J., KUMAR, P., OKASAKI, M. E., SIEGEL, E. H., AND STEERE, D. C. Coda: A Highly Available File System for a Distributed Workstation Environment. *IEEE Transactions on Computers 39*, 4 (April 1990).

[12] SATYANARAYANAN, M., KISTLER, J. J., MUMMERT, L. B., EBLING, M. R., KUMAR, P., AND LU, Q. Experience with Disconnected Operation in a Mobile Computing Environment. In *Proceedings of the 1993 USENIX Symposium on Mobile and Location-Independent Computing* (Cambridge, MA, August 1993).

[13] SATYANARAYANAN, M., MASHBURN, H. H., KUMAR, P., STEERE, D. C., AND KISTLER, J. J. Lightweight Recoverable Virtual Memory. *ACM Transactions on Computer Systems 12*, 1 (Februrary 1994), 33–57. Corrigendum: May 1994, Vol. 12, No. 2, pp. 165-172.

[14] SATYANARAYANAN, M., NOBLE, B., KUMAR, P., AND PRICE, M. Application-aware Adaptation for Mobile Computing. *Operating Systems Review 29* (January 1995). Also as technical report CMU-CS-95-183, School of Computer Science, Carnegie Mellon University.

[15] SIDEBOTHAM, R. Volumes: The Andrew File System Data Structuring Primitive. In *European Unix User Group Conference Proceedings* (August 1986). Also available as Tech. Rep. CMU-ITC-053, Carnegie Mellon University, Information Technology Center.

[16] STEERE, D. C., AND SATYANARAYANAN, M. Using Dynamic Sets to Overcome High I/O Latencies During Search. In *Proceedings of the Fifth IEEE HotOS Conference* (Orcas Island, WA, May 1995).

[17] STEERE, D. C., AND WING, J. Specifying Weak Sets. In *Proceedings of the 15th International Conference on Distributed Computing Systems* (Vancouver, B.C., Canada, June 1995).