Overcoming the Network Bottleneck in Mobile Computing

Maria R. Ebling, Lily B. Mummert, David C. Steere
School of Computer Science
Carnegie Mellon University

1 Introduction

System designers have traditionally treated the network as an inexhaustible resource, focusing their efforts on optimizing CPU and storage usage. For instance, the popular NFS file system [8] supports diskless operation, thereby avoiding use of local secondary storage at the expense of increased network usage. But in mobile computing, it is the network, rather than CPU or storage, that will be the scarce resource. The time has come when we must treat the network as a first-class resource, expending the CPU and storage resources necessary to use it intelligently. In this paper we argue that *prescient caching and smart scheduling are key techniques for overcoming the network bottleneck.* We use the Coda file system [9] as a case study to substantiate our position.

2 Why the network is the bottleneck

The essence of our argument is that mobile clients will always encounter a much wider range of network performance than their stationary counterparts. Users frequently take their mobile computers to places where traditional (wired) networks do not go: to the beach, on a plane, or at home. Although global wireless connectivity is on the horizon, much of this connectivity will be intermittent, of high latency or of low bandwidth. While in the future gigabit connectivity may be available at work, home, or other discrete locations, it will be limited to a few oases in a vast desert of poor connectivity.

Moreover, the latency to access data across global distances is inherently high. With mobile computers, information that is normally nearby suddenly becomes distant! Since it takes light 16 msec to cross the United States, accessing information on the other coast is at best equivalent to a present-day local disk write, even assuming full network connectivity and infinitely fast servers, protocols, and disks. The only way to avoid this latency is to avoid communication altogether or to predict the access far enough in advance that the latency is hidden from the user.

Furthermore, just because global connectivity is on the horizon does not mean it will be free. In the future, charging for network communication services will become widespread. As anyone who has used a modem over long distance phone lines can attest, these charges accumulate rapidly. Users of mobile computers will demand that their systems use networks no more than necessary to support critical applications. Furthermore, these users (and their employers) will insist on having ultimate control over costs.

For these reasons, mobile clients must be able to cope with poor or expensive network connectivity while preserving the ability to aggressively use cheap, high-bandwidth communication when available. Our approach to this challenge exploits locality extensively. Coda anticipates network activity using prescient caching, and defers such activity until convenient using smart scheduling. Even in the absence of locality, smart scheduling can be used to hide network latency from the user.

3 Avoiding the network by prescient caching

One way to avert the high cost (either performance cost or dollar cost) of network communication is to avoid use of the network when it is expensive by predicting future accesses and fetching necessary data when the network is cheap. True prescience, of course, requires knowledge of the future. In practice, we approximate this knowledge in Coda using the following three strategies.

Two of these techniques, *LRU caching* and *hoarding*, are already present in the Coda cache manager and have been described elsewhere [5, 4]. The first of these, LRU caching, automatically keeps objects that have been recently accessed in the cache. The second of these, hoard-

This research has been supported by the National Science Foundation under Grant ECD-8907068, and the Air Force Materiel Command (AFMC) and the Advanced Research Projects Agency (ARPA) under Contract F19628-93-C-0193. Support also came from Digital Equipment Corporation, IBM Corporation, and Xerox Corporation. The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the NSF, AFMC, ARPA, DEC, IBM, Xerox, or the U. S. Government.

Authors' addresses: School of Computer Science, Carnegie Mellon University, 5000 Forbes Avenue, Pittsburgh, PA, 15213-3891. e-mail: mre@cs.cmu.edu, lily@cs.cmu.edu, dcs@cs.cmu.edu.

ing, allows the user to supply a prioritized list of files that should be present in the cache in case of a disconnection. Coda transparently combines these two techniques to anticipate file references on the short- and long-term prediction horizons.

While hoarding has certainly proven successful in anticipating references on the long-term prediction horizon, it does require user assistance. The third technique that we are developing, *Casper*, extends the LRU prediction horizon automatically. Using this technique, the cache manager maintains a concise summary of file reference history in the Casper Database. This database consists of *ghost entries*, which keep track of files that have been in the cache and allow the cache manager to assign a rank to each entry based on its reference history. These ranks are then used to augment object priorities, effectively inferring a user's future accesses based on his past activity. Evidence suggests that LRU-only mechanisms are insufficient for predicting a user's future file references, but also suggest that users' activities are reasonably limited [10, 6]. For these reasons, we believe that extended histories will be successful in anticipating future file references and thus avoiding future network use.

Coda expends CPU, storage and cheap network resources in anticipation of disconnection. Since LRU-only mechanisms are insufficient for all but the shortest failures, Coda combines a concise, extended history with user assistance in order to cope seamlessly with disconnection, from the very short-term to the very long-term.

4 Smart scheduling to reduce network cost

Although correctly anticipating future accesses can substantially reduce the need to communicate with servers, it does not completely eliminate the desire to maintain open communication channels with them. After all, the chief advantage of a distributed file system is that it allows users to easily share information. Sharing does not work well if updates are not propagated to and from mobile clients. Furthermore, since the cache manager will never be 100% successful in anticipating the user's every access, the ability to fetch critical data from the servers is a necessity. Thus, it is essential that mobile Coda clients communicate efficiently and effectively with servers.

In examining client-server traffic, we found four types of essential network activity: *demand fetching*, *prefetching*, *update propagation* and *cache maintenance*. To cope with situations in which communication with the servers is expensive, Coda minimizes and prioritizes client-server communication.

A first step toward minimizing network traffic is to reduce the amount of cache validation traffic. We have succeeded in substantially reducing this traffic by raising the granularity of callbacks from a single file to an entire volume [7]. This technique allows large portions of the cache to be validated with a single message, resulting in substantial savings in practice, particularly in environments where communication is intermittent. Note that other techniques such as differential write back [1] and data compression [3] also attempt to minimize network traffic; these techniques are orthogonal to this discussion and could also be incorporated into our system to further reduce network traffic.

Coda also prioritizes data transmission activity. In the default prioritization, demand requests are given the highest priority since the user is presumably waiting for the data. The remaining types of requests, namely prefetches and updates, are relegated to the background. However, a user may alter the prioritization explicitly if he so chooses. Because background prefetching and updating are performed asynchronously, we refer to them as *trickle charging* and *trickle discharging* the cache. We prioritize trickle charge over trickle discharge because a prefetch request is likely to become a demand request if postponed.

Trickle discharge, a form of write-back caching, logs updates on disk and propagates them to the servers as network bandwidth and cost allow. Although the desire to make new data available at the server suggests that updates be propagated eagerly, doing so would render ineffective the log optimizations used to coalesce updates. There is substantial evidence from our experience with disconnected operation indicating that these optimizations are highly effective [4]. To keep network communication to a minimum, we monitor the compressibility of the replay log over time, and defer propagation of updates until the log is quiescent.

Each of these techniques requires us to expend CPU and disk resources in order to reduce network demands. Minimizing data transmission is useful but not sufficient to address the network bottleneck. Examining the sources of communication, prioritizing them, and decoupling them from computation to the extent possible is necessary to properly utilize this precious resource.

5 Smart scheduling in the absence of locality

All of the above techniques implicitly rely on locality of reference. Unfortunately, there are important applications, such as search, that do not exhibit locality of reference. Given the increasing size and scale of today's distributed systems such as WWW[2] and AFS[11], search will be an essential operation to tomorrow's mobile users. Fortunately, we can exploit the semantics of search to schedule network resources intelligently, allowing us to

reorder requests and overlap fetches to hide latency.

Coda introduces a simple abstraction, called *dynamic sets*, to better support search. Using this abstraction, a user creates a dynamic set by specifying a query which Coda expands into a set of file names. Coda then fetches these named objects for the user, who iterates on them looking for the desired object.

Dynamic sets offer several direct benefits to mobile users. First, the strong hint of future access implicit in set membership allows the system to exploit available bandwidth for prefetching. Second, dynamic sets allow the system to reorder requests, overlapping user think time or computation time on quickly obtained elements with the latency of fetching others. Finally, the system could ship the query to the data, allowing it to reduce network utilization by choosing a much smaller subset of objects to transfer back to the mobile host.

Dynamic sets achieve these benefits by allowing higher levels to cleanly provide hints to lower levels in order to help them more intelligently schedule network usage. Early experiments with this approach confirm that substantial savings can be achieved without sacrificing performance [12].

6 Conclusion

In order to effectively support mobile computers, system designers must view the network as a first-class resource, expending CPU and disk resources to reduce the use of network resources during periods of poor network connectivity. Even today, in the time it takes to transfer 4 KB over a 64 Kb line, one can execute 12.5 million instructions, or write those same 4 KB to disk 12 times. As processor technology advances, we expect these performance differences to be even more pronounced. Thus, it is our position that squandering a few processor cycles and a bit of disk space in order to minimize network usage is, and will remain, a sensible tradeoff. Our approach systematically substitutes local resources to minimize the effect of the network bottleneck. At the same time, we aggressively exploit cheap network resources when available. We posit that such an *adaptive* approach to network usage is essential for effective operation in a mobile environment.

References

[1] AirSoft AirAccess 2.0 Mobile Networking Software: Product Overview and Reviewer's Guide, June 1994.

[2] BERNERS-LEE, T., CAILLIAU, R., GROFF, J. F., AND POLLERMAN, B. World Wide Web: The Information Universe. *Electronic Networking: Research, Applications, and Policy 1*, 2 (Spring 1992).

[3] DOUGLIS, F. On the Role of Compression in Distributed Systems. In *Proceedings of the 5th ACM SIGOPS Workshop on Models and Paradigms for Distributed Systems Structuring* (September 1992).

[4] KISTLER, J. J. *Disconnected Operation in a Distributed File System*. PhD thesis, Carnegie Mellon University, School of Computer Science, 1993.

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

[6] KUENNING, G. H., POPEK, G. J., AND REIHER, P. L. An Analysis of Trace Data for Predictive File Caching in Mobile Computing. In *Proceedings of the Summer 1994 USENIX Conference* (Boston, MA, 1994).

[7] MUMMERT, L., AND SATYANARAYANAN, M. Large Granularity Cache Coherence for Intermittent Connectivity. In *USENIX Summer Conference Proceedings* (June 1994), USENIX Association, pp. 279 – 289.

[8] SANDBERG, R., GOLDBERG, D., KLEIMAN, S., WALSH, D., AND LYON, B. Design and Implementation of the Sun Network File System. In *Summer Usenix Conference Proceedings, Portland* (1985).

[9] 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).

[10] 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* (August 1993).

[11] SPASOJEVIC, M., AND SATYANARAYANAN, M. A Usage Profile and Evaluation of a Wide-Area Distributed File System. In *Winter Usenix Conference Proceedings* (San Francisco, CA, 1994).

[12] STEERE, D., AND SATYANARAYANAN, M. A Case for Dynamic Sets in Operating Systems. Tech. Rep. CMU-CS-94-216, School of Computer Science, Carnegie Mellon University, November 1994.