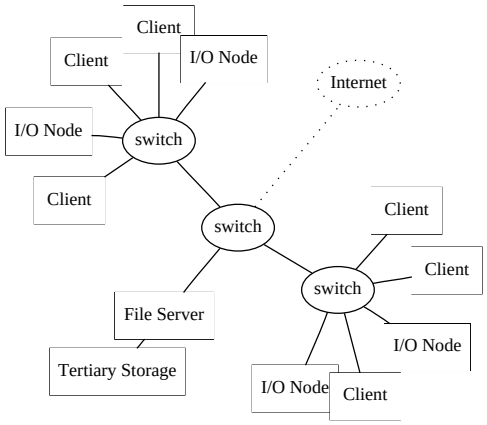
A Case for Buffer Servers

Darrell Anderson, Ken Yocum, and Jeff Chase
Department of Computer Science
Duke University
Durham, NC 27708
{*anderson, grant, chase*}*@cs.duke.edu*

Abstract

Faster networks and cheaper storage have brought us to a point where I/O caching servers have an important role in the design of scalable, high-performance file systems. These intermediary I/O servers — or buffer servers — can be deployed at strategic points in the network, interposed between clients and data sources such as standard file servers, Internet data servers, and tertiary storage. Their purpose is to provide a fast and incrementally scalable I/O service throughout the network while reducing and smoothing demands on shared data servers and the network backbone.

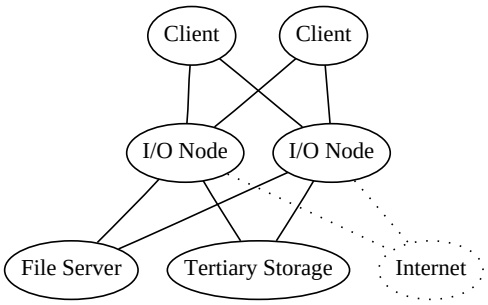
This position paper outlines a case for caching buffer servers and addresses some of the key technical challenges in the design of a buffer service. We also describe the role of buffer servers in the Trapeze project, which uses gigabit networks as a vehicle for high-speed network I/O.



1 Introduction

Storage access is a driving application for high-speed LAN interconnects. Faster I/O bus standards and the imminent arrival of Gigabit Ethernet as a commodity greatly expand the capacity of inexpensive PCs to handle large amounts of data for scalable computing, network services, multimedia, and visualization. A fundamental challenge is to construct network storage systems that combine easy-to-administer sharing of logical and physical resources, incrementally scalable capacity, and performance that matches the speed of the network.

The Trapeze project is an effort to use storage access through a gigabit-per-second interconnect (Myrinet) to "cheat" the disk I/O bottleneck for I/O-intensive applications (see Section 2). Our ultimate goal is to run external memory applications on PCs and workstations with performance approaching internal memory speeds, by using the network as the primary access path to external storage —


Client
Client
I/O Node
I/O Node
File Server
Tertiary Storage
Internet

On our current prototype, a user process can read data from a single network memory server through the file system interface at speeds approaching 100 MB/s, on 500 MHz DEC Alpha Miata (PWS500a) systems with custom Myrinet firmware optimized for network storage traffic [27, 2]. Client CPU utilizations at full bandwidth are in the 30-40% range, with server CPU utilization around 5%. An unpredicted 8KB page fault from network memory takes about 170μs. This level of performance creates a new potential for I/O-intensive computing on desktop-class systems. The Miatas and newer Pentium and Xeon PC systems have sufficient memory system bandwidth to serve the CPU and the I/O system at the same time.

Reliance on network memory has allowed us to begin experimenting with a network I/O system that can source and sink data at Myrinet network speeds, for external problem sizes up to a few gigabytes. Our interest now is to supplement the I/O servers with parallel disks to fill and drain the network memory, with storage system features to allow clients to direct the movement and placement of data on the disks. The extended servers are analogous to I/O nodes in parallel multicomputers, in that their primary purpose is to meet the short-term I/O needs of ongoing computations. In particular, they provide scratch storage for intermediate data outputs and paging: most of this data dies without ever reaching permanent file storage.

3 The Case for a Buffer Service

While a buffer service could take several forms, our preferred scheme manages the memory and disks as a cache over a common file system name space incorporating primary file servers and tertiary storage. The cache is served by a collection of storage nodes (buffer servers) distributed through the network, interacting through a distributed protocol to locate cached data and keep it consistent.

The buffer service is motivated by familiar assumptions about I/O accesses. First, read traffic from each client is amenable to caching, and most references to a given data file originate from a particular region of the network over some period of time. Most shared read accesses are to read-only files that can be replicated easily in the network, such as executables, input datasets, or media files. Second, a significant share of write traffic is short-lived. Most updates are to non-shared files, including temporary files, private long-term storage, and paging traffic. Similar assumptions have justified a full range of file caching schemes, and have been validated by trace studies in conventional LAN environments [4, 21].

We add three new assumptions, based on recent technology trends.

- In addition to allowing sharing of physical storage resources, the network is the fastest access path to I/O. Even with disk bandwidths growing at 40% per year, leading edge networks historically deliver an order of magnitude higher bandwidth than a single disk. Where performance is important, it is more attractive to remove disk storage from the clients and plug it into the network as a shared resource, so that clients may use multiple disks in tandem to source and sink data at network speeds. The higher speed network adds very little latency to data fetched indirectly through a buffer server.

- Storage accesses from clients tend to be bursty, and clients and storage nodes can burst at close to network bandwidth. This means that a demanding client can consume a large share of network bandwidth, but can be served by a relatively small number of storage nodes. This creates both a need and an opportunity to localize I/O traffic in the network.

- The cost of additional storage is now low enough to allow increased use of caching and replication. Today, an 85 GB software-managed IDE RAID array can be purchased for under $2000, and can serve data at 60 MB/s from a $1200 PC equipped with a high-speed network interface. A buffer service could be built at modest cost from a collection of similar machines dispersed through the network.

Given these assumptions, buffer servers offer the following benefits:

- *Network memory.* Buffer servers can hold prefetched data, recent writes, or "hot" data in memory; a client can fetch pages or blocks across the network from buffer server memory 50 to 100 times faster than an average disk seek. Network memory can act as a fast "L2 cache" for file blocks and virtual memory pages. Hit ratios in network memory can be improved by speculatively prefetching data from buffer server disk [25].

- *Read caching, prefetching, and hoarding*. In addition to the caching benefits, anticipated data needs can be met by prefetching from remote servers to nearby buffer servers, perhaps during idle periods [14]. Once data is stored nearby, clients can access it at full speed (e.g., video data or input data sets for a computation). Storage in the buffer service can also be used as an expansion pool for other caching needs, such as local Web caches.

- *Write buffering.* Buffer servers can absorb write bursts at network speeds, buffering them into reliable memory (e.g., [7]) and eventually to disk. This can reduce write traffic to file servers for repeated writes (e.g., during an edit–build–test development cycle) and files that

die in the cache (e.g., temporary data files for intermediate stages in computations). Given sufficient storage, only long-lived writes will filter through to the permanent end-servers, and these can be scheduled when bandwidth is available.

- *Hierarchical network storage.* Buffer servers may act as a high-speed staging area over tertiary storage. Tertiary datasets or other items can be brought into the buffer server in preparation for use or further processing. Here "tertiary storage" can include Internet information servers, which now provide a wide variety of useful but uncomfortably large datasets such as access traces, satellite images, census data, economic data, etc. These items can be bound into the local name space, e.g., using WebFS [24] running as an NFS proxy similar to Alex [6].

- *Active storage hierarchies.* Since they emphasize high-speed data access to a relatively small set of clients, buffer servers are a natural point for extended interfaces that advanced applications use to control their I/O. Examples include the user-directed layout of data across disks and movement of data between remote disk and remote memory, as well as other extensions including support for basic filtering and merging steps executed close to where the data resides [20, 1]. If Active Disks are available, they can be incorporated as another, lower level in the CPU hierarchy. Buffer servers also provide a context for bringing parallel file system concepts into workstation networks.

In summary, buffer servers present an opportunity to integrate high-speed storage into workstation/PC networks in a cost-effective way. From an administrative standpoint, buffer servers are easy to manage since they are autonomous and can be organized in the most convenient way without reconfiguring the file system name space, which is typically a prerequisite to incrementally expanding file service capacity to an organization.

4 Design Issues

A complete buffer service presents several interesting design challenges. Ideally, it would be possible to inject buffer servers incrementally into an existing client/server file system environment without reimplementing existing clients and servers (e.g., NFS). As more buffer servers are added, other nodes should detect them and incorporate their resources into the service. Finally, we need to handle writes, which introduces consistency and reliability issues. For example, the design should be able to guarantee "horizontal consistency" for clients accessing the same file through different buffer servers.

We are implementing the buffer service as a distributed service built into a general-purpose operating system kernel (FreeBSD 4.0). The buffer server I/O nodes could be network storage appliances (e.g., NASD), but we use generic PCs because they are cheap, fast, and programmable. While the I/O bus is the limiting component in our buffer server configurations, it merely constrains the number of disks that can be used effectively on each server (and its network interface), rather than fundamentally limiting performance: I/O bus latencies are insignificant where disk accesses are involved, and bus latencies are significantly reduced by adaptive message pipelining [27]. Moreover, connecting disks to a standard PC chassis offers a processor close to the disks at low cost; our experience is that a 300MHz Pentium-II can serve data at close to network speed (100 MB/s) with sufficient CPU and memory bandwidth remaining to drive global I/O prefetching [25] or other server-side extensions. While the CMU NASD group has noted that this approach can add up to 80% to the cost of disk capacity [10], the extra money paid for the CPU, memory, and shared network interface is roughly equivalent to the price differential between IDE and SCSI storage today.

Our buffer service prototype is implemented as a pair of loadable kernel modules using the stackable VFS file system interface [17]. Both modules are stacked above NQNFS [16], an extended NFS file system implementation using leases [11]. The server side module handles requests from clients, manages buffer server disk and memory, and issues read-through and writeback requests to the backing servers. The client-side module caches block location maps and redirects read and write requests to the buffer service using a lightweight RPC protocol. Client systems that cannot use the client-side module could access files through a buffer server acting as an NFS proxy (this approach is also used in xFS and Frangipani). This adds an extra network hop to some requests, and also requires use of the heavier-weight NFS and IP protocols.

Here are a few of the key issues and our current approaches to resolving them.

- *Symbolic naming.* Our buffer service passes all name space operations through to the backing servers, decoupling name space management (and access control) from block management. This is similar to a number of systems that use central file managers, including Swift [5], Zebra [12], and Cheops/NASD [10]. Since only the block I/O operations are offloaded, the file manager could become a bottleneck, e.g., for workloads with many small files. In contrast, xFS [3] distributes file manager responsibilities. However, the central file manager leads to a simple and compatible implementation, and client-side name and attribute caches are effective for working sets up to a few thousand files.

- *Security.* On file lookups (*open*), the client module hashes the file server's returned NFS file handle to a token that uniquely identifies the file within the buffer service. This token is used for subsequent operations on the file; the file handle itself is held by the buffer service for fetches and writebacks to the backing server. Security is no worse or better than NFS; authentication at *open* time prevents unauthorized clients from obtaining file handles. Client I/O requests pass abbreviated credentials to buffer servers, which in turn present them to file servers for fetches and writebacks. Thus client kernels are trusted to correctly represent the identity of their users to the buffer service.

- *Location.* Buffer servers and clients must be able to locate blocks cached anywhere in the buffer service. The blocks of a given file may be cached on different servers; the directory protocol must allow the buffer service to function as a unified level of the caching hierarchy. This can be done using file block maps obtained and cached by clients (or NFS proxies) at open time, in a manner similar to Zebra, xFS, or Cheops/NASD. The system can support concurrent write sharing by synchronizing on the block map.

- *Reliability and failure semantics.* In principle, buffer servers should fail no more often than file servers. In our design, buffer servers do not guarantee to write synchronously to disk, so permanent losses may occur unless a reliable memory is used [7]. Any losses must be reported cleanly to clients (rather than exposing corrupt data), and clients must be able to flush writes from the buffer server or bypass it entirely for critical files. In principle it is possible to improve reliability by directing the buffer service to replicate on a per-file basis, but we have not considered such extensions.

5 Related Work

Buffer servers combine the benefits of several previous approaches. Caching has long been the basis for scalability in distributed file systems, and AFS [13] and others have expanded the cache to include local disks. Today, faster LANs create an opportunity to move cache disks and memory one or two hops away from endstations, simplifying sharing of storage resources and aggregating them for better performance (e.g., by disk striping) in a cost-effective way.

The iAFS project [19, 18] reported simulation results from caching nodes interposed between AFS clients and servers. One difference between iAFS and buffer servers is that an iAFS client speaks with a single intermediate iAFS server acting on behalf of a given end-server. In contrast, any number of buffer server I/O nodes may store data for a single end-server, offering clients sustained aggregate disk bandwidth for large I/O requests.

The idea of a unified shared cache as an intermediate level of the storage hierarchy is common to cooperative caching [8] and global memory management [9], which exploit the low latency of network memory access on high-speed networks by allowing clients to share their I/O cache buffers for the benefit of the global workload. While our proposal is compatible with cooperative caching — among the endstations, buffer servers, or both — it also brings to bear dedicated storage resources that can be redeployed and moved through the network as needed. In this respect buffer servers are similar to memory servers, but extended to incorporate parallel disks for high-bandwidth access to larger data sets. The buffer service would also use distributed load information to select storage nodes in a manner similar to these systems.

Many file systems including Swift [5], Zebra [12], Swarm [22], and others [15, 23, 3] stripe data across network disks; our proposal uses similar techniques in the context of caches, which naturally localize traffic within the network.

The basic buffer server storage model is similar to Cheops/NASD [10] and Frangipani/Petal [15, 23]. Large storage systems are built incrementally by adding storage nodes to a network. The nodes function as block servers, carrying the bulk of I/O traffic while leaving higher-level naming and access control functions to file mangers and/or clients. Because buffer servers are based on commodity PCs, they offer significant memory for caching and prefetching, incorporate a current-generation processor, and ride the leading edge of the cost/performance curve.

6 Conclusion

This position paper presents the case for caching I/O servers that incorporate memory and parallel disk caching through a unified *buffer service.* Technology advances — faster networks (e.g., Gigabit Ethernet) and cheaper storage — both increase the need for scalable high-speed file access and reduce the cost of caching servers as a way to provide it.

References

[1] A. Acharya, M. Uysal, and J. Saltz. Active disks: Programming model, algorithms and evaluation. In *Proceedings of the Eighth Symposium on Architectural Support for Programming Languages and Operating Systems*, pages 81–91, Oct. 1998.

[2] D. Anderson, J. S. Chase, S. Gadde, A. J. Gallatin, K. G. Yocum, and M. J. Feeley. Cheating the I/O bottleneck: Network storage with Trapeze/Myrinet. In *Proceedings of the*

1998 Usenix Technical Conference, June 1998. Duke University CS Technical Report CS-1997-21.

[3] T. Anderson, M. Dahlin, J. Neefe, D. Patterson, D. Roselli, and R. Wang. Serverless network file systems. In *Proceedings of the ACM Symposium on Operating Systems Principles*, pages 109–126, Dec. 1995.

[4] M. G. Baker, J. H. Hartman, M. D. Kupfer, K. W. Shiffiff, and J. K. Ousterhout. Measurements of a distributed file system. In *Proceedings of the 13th ACM Symposium on Operating Systems Principles*, pages 193–211, Oct. 1991.

[5] L.-F. Cabrera and D. D. E. Long. Swift: Using distributed disk striping to provide high I/O data rates. *Computing Systems*, 4(4):405–436, Fall 1991.

[6] V. Cate. Alex - a global filesystem. In *Proceedings of the 1992 USENIX File System Workshop*, pages 1–12, May 1992.

[7] P. M. Chen, W. T. Ng, S. Chandra, C. Aycock, G. Rajamani, and D. Lowell. The Rio file cache: Surviving operating system crashes. In *Proceedings of the 1996 International Conference on Architectural Support for Programming Languages and Operating Systems*, Oct. 1996.

[8] M. D. Dahlin, R. Y. Wang, and T. E. Anderson. Cooperative caching: Using remote client memory to improve file system performance. In *Proceedings of the First Symposium on Operating System Design and Implementation*, pages 267–280, Nov. 1994.

[9] M. J. Feeley, W. E. Morgan, F. H. Pighin, A. R. Karlin, and H. M. Levy. Implementing global memory management in a workstation cluster. In *Proceedings of the Fifteenth ACM Symposium on Operating Systems Principles*, 1995.

[10] G. A. Gibson, D. F. Nagle, K. Amiri, F. W. Chang, H. Gobioff, C. Hardin, E. Riedel, D. Rochberg, and J. Zelenka. A cost-effective, high-bandwidth storage architecture. In *Proceedings of the Eight Conference on Architechtural Support for Programming Languages and Operating Systems*, Oct. 1998.

[11] C. Gray and D. Cheriton. Leases: An efficient fault-tolerant mechanism for distributed file cache consistency. In *Proc. of the Twelfth Symposium on Operating Systems Principles*, pages 202–210, Dec. 1989.

[12] J. H. Hartman and J. K. Ousterhout. The Zebra striped network file system. In *Proceedings of the Fourteenth ACM Symposium on Operating Systems Principles*, pages 29–43, Ashville, NC, 1993. ACM Press.

[13] J. H. Howard, M. L. Kazar, S. G. Menees, D. A. Nichols, M. Satyanarayanan, R. M. Sidebotham, and M. J. West. Scale and performance in a distributed file system. *ACM Transactions on Computer Systems*, 6(1):51–81, Feb. 1988.

[14] J. Kistler and M. Satyanarayanan. Disconnected operation in the Coda file system. In *ACM Transactions on Computer Systems*, Feb. 1992.

[15] E. Lee and C. Thekkath. Petal: Distributed virtual disks. In *Eighth International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 84–93, Oct. 1996.

[16] R. Macklem. Not quite NFS, soft cache consistency for NFS. In *USENIX Association Conference Proceedings*, pages 261–278, Jan. 1994.

[17] M. K. McKusick. The virtual filesystem interface in 4.4BSD. *Computing Systems Journal of the USENIX Association*, 8(1):3–25, 1995.

[18] D. Muntz and P. Honeyman. Multi-level caching in distributed file systems -or- your cache ain't nuthin' but trash. In *Proc. of the Winter 1992 USENIX Conference*, pages 305–314, 1991.

[19] D. A. Muntz, P. Honeyman, and C. J. Antonelli. Evaluating delayed write in a multilevel caching file system. In *Proc. 2nd Intl. Conf. on Dist. Platforms*, pages 415–429, Feb. 1996.

[20] E. Riedel, G. Gibson, and C. Faloutsos. Active storage for large-scale data mining and multimedia. In *Proceedings of the 24th VLDB Conference*, 1998.

[21] D. Roselli and T. Anderson. Characteristics of file system workloads. Technical Report CSD-98-1029, UC Berkeley, Department of Computer Science, Oct. 1998.

[22] R. Sundaram. Design and implementation of the Swarm storage server. Technical Report TR98-02, University of Arizona, Department of Computer Science, Feb. 1998.

[23] C. Thekkath, T. Mann, and E. Lee. Frangipani: A scalable distributed file system. In *Ninth International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 224–237, Oct. 1997.

[24] A. Vahdat, P. Eastham, and T. Anderson. WebFS: A global cache coherent filesystem. *Technical Draft*, Dec. 1996.

[25] G. M. Voelker, E. J. Anderson, T. Kimbrel, M. J. Feeley, J. S. Chase, A. R. Karlin, and H. M. Levy. Implementing cooperative prefetching and caching in a globally-managed memory system. In *Proceedings of ACM International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS '98)*, June 1998.

[26] R. Wang and T. Anderson. xFS: A wide area mass storage file system. In *Proc. Fourth Workshop on Workstation Operating Systems*, pages 71–78, 1993.

[27] K. G. Yocum, D. C. Anderson, J. S. Chase, S. Gadde, A. J. Gallatin, and A. R. Lebeck. Adaptive message pipelining for network memory and network storage. Technical Report CS-1998-10, Duke University Department of Computer Science, Apr. 1998.

[28] K. G. Yocum, J. S. Chase, A. J. Gallatin, and A. R. Lebeck. Cut-through delivery in Trapeze: An exercise in low-latency messaging. In *Proceedings of the Sixth IEEE International Symposium on High Performance Distributed Computing (HPDC-6)*, pages 243–252, Aug. 1997.