

A Case for Network-Centric Buffer Cache Organization

Gang Peng Srikant Sharma Tzi-cker Chiueh
Computer Science Department
State University of New York at Stony Brook
Stony Brook, NY, 11794
{gpeng, srikant, chiueh}@cs.sunysb.edu

Abstract

The emergence of clustered and networked storage architecture gives rise to a new type of servers which act as data conduits over the network for remotely stored data. These servers, which we call pass-through servers, are mainly responsible for passing the data through them without interpreting it in any way. In this paper, we put forward a scheme of network-centric buffer cache management. This scheme can facilitate the data transmission through pass-through servers by avoiding redundant data copying, and by caching the data in a network-ready form, while having no modifications to the existing buffer cache organization. The performance measurement on a NFS server using iSCSI storage running on Linux with this scheme shows throughput improvement of more than 50% compared to a NFS server on common Linux, while consuming about 40% less of CPU resource.

1 Introduction

Modern routers or packet forwarding devices are designed to have minimal data copying overhead. The essence of this position paper is that the same principle should be applied to the design of the buffer cache in *pass-through* servers, which are servers that simply channel data between two external parties and do not need to “interpret” most of the payloads passed-through.

An example of the pass-through servers is an NFS server that does not use local disks but relies on network storage devices such as iSCSI or Fiber-Channel storage servers. Because a pass-through NFS server does not have local disks, its main job is to pass data between NFS clients and iSCSI/Fiber-Channel storage servers. Therefore, intuitively it makes sense to organize and store all the data passed-through in a network ready format, including the cached data. By taking a network-centric view toward buffer cache organization, it is possible to eliminate redundant data copy-

ing operations, as well as associated network packet header preparation overheads. While conceptually appealing, this approach requires significant modifications to the file system underlying the NFS server, because the buffer cache is typically structured to simplify the file system’s processing.

In this paper, we make a case for network-centric buffer cache organization for pass-through servers’ page/buffer cache, by presenting the data flow of a standard iSCSI-based NFS server and its performance overhead due to memory copying, sketching a cache buffer design that embodies this idea while keeping the existing buffer cache intact, and demonstrating the potential performance gain of the proposed approach through measurements on a preliminary prototype.

2 Related Work

The idea of network-centric buffer cache organization has its roots in research aimed at reducing redundant memory copy operations. Fbuf [5] uses a per-process buffer pool that is pre-mapped in both the user and kernel address spaces, thus eliminating the user-kernel data copy. However, all the programs have to be converted to use this alternate set of APIs and programming models. Hence, this approach does not retain application compatibility. As compared with fbuf, network-centric cache organization is designed to obviate all the data copies along the data path of pass-through server, not just the user-kernel data copy. These data copies inside pass-through servers generally involve more subsystems, including applications, the file cache and the network subsystem. Also, we make great effort to keep the modifications to applications minimal. H.K. Jerry Chu [3] describes a transparent copy avoidance approach through the use of page remapping and copy-on-write. Same as fbuf, this approach intends to remove the user-kernel data copy only. Moreover, it requires Maximum Transfer Unit (MTU) size of the physical network to be larger than the system page size. Our scheme has no such restriction.

IO-Lite [7] is a unified I/O buffering and caching system for generic-purpose operating systems. It allows different subsystems, such as the file system, the file cache, the network subsystem and applications, to share a single physical copy of the data safely and concurrently, and thus eliminates redundant data copying. This data sharing is through a new data structure, named "buffer aggregate", which is an ordered list of $\langle \text{pointer}, \text{length} \rangle$ pairs, and a new set of APIs. To leverage the advantages of IO-Lite, all the subsystems need to be changed significantly. Moreover, the use of buffer aggregate complicates the cache replacement and paging substantially. In contrast, our scheme is designed to facilitate data I/O within a special type of servers, *pass-through servers*, and is much simpler and much less intrusive to existing kernel components than IO-Lite, especially with no modifications to the existing buffer cache at all. The application compatibility is also largely maintained in our scheme.

Axon [8] is a messaging system that supports IPC (inter-process communication) with high throughput and low latency across high speed networks. To achieve this goal, it designs an application-oriented lightweight transport protocol (ALTP) and has the critical path of ALTP implemented in the network interface. As a result, Axon requires significant changes to existing operating systems, communication protocols and network interfaces. In comparison, network-centric cache organization focuses on a simpler scenario of communication over high speed network among different participants, which is the data transmission through pass-through servers, and can be readily applied to existing operating systems.

3 Data Flow within an iSCSI-Based NFS Server

iSCSI (Internet SCSI) [1] is an IP-based storage networking standard for accessing remote storage devices. By encapsulating SCSI commands within IP packets, iSCSI makes it possible for a server and its storage to be separated over a large geographical distance. iSCSI is based on a client-server architecture that attempts to emulate the original SCSI (Small Computer System Interface) standard. The clients, called iSCSI initiators, are units that access data stored on the remote servers through the iSCSI protocol, and the servers, called iSCSI targets, provide data access service through the iSCSI protocol to the initiators. The iSCSI protocol itself runs over TCP. An iSCSI NFS server, whose software structure is shown in Figure 1, stores its data on the remote iSCSI targets, not on the local disks. As to the buffer/page cache, an iSCSI initiator looks just like a local storage unit since the iSCSI initiator hides the detail of iSCSI protocol by encapsulating SCSI commands and interpreting iSCSI responses.

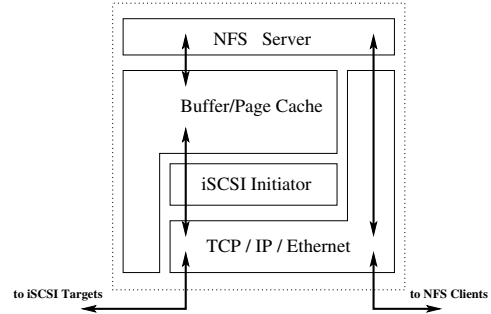


Figure 1. Software modules and data path within a NFS server based on iSCSI storage. The solid lines with arrows indicate the data copying across different modules.

Each module in Figure 1 needs to explicitly copy data to its neighboring modules. The solid lines with arrows depict these data copying operations across different modules within an iSCSI-based NFS server. These data copying operations are necessary because different modules require the data to be represented in different formats. In the Linux kernel, for example, data is stored in the format of `sk_buff` in the network stack, and as contiguous buffer chunks in the page/buffer cache, which are accessed by the NFS server. When a read-like NFS request, including `ROOT`, `GETATTR`, `LOOKUP`, `READLINK`, `READ`, `REaddir` and `STATFS` [6], arrives at the NFS server, it services the request by reading the requested data from the page/buffer cache in the local file system. If the content happens to be cached in the buffer/page cache, the local file system copies the data directly to the NFS server. Otherwise, a disk access request is sent to the iSCSI initiator module to retrieve the data from the remote iSCSI targets. When the iSCSI initiator receives the requested data over the network, it copies the data from the network stack first into the buffer/page cache, and then into the NFS server. Once the NFS server module obtains the requested data, it composes an NFS reply and sends it out using Remote Procedure Call (RPC) protocol. Sending the replies out incurs one more data copying from the NFS server module to the network stack.

For a write-like NFS request, such as `SETATTR`, `WRITECACHE`, `WRITE` and `MKDIR` [6], the same number of data copying operations is required within the NFS server. When an NFS write-like request arrives, the content stored in the request is first copied from the network stack into the NFS server module, which then writes the data into the page cache of the local file system. In the case that some dirty pages need to be flushed, the NFS server writes them back to the remote iSCSI targets by issuing disk requests

to the iSCSI initiator, which copies the data to be flushed to the network stack. Some other types of NFS requests, including CREATE, REMOVE, RENAME, LINK, SYMLINK, MKDIR and RMDIR [6], require both read and write operations.

In summary, a read-like NFS request takes two (in the case of page cache hit) or three (in the case of page cache miss) memory copying operations; a write-like NFS request requires two memory copying operations, if the content of this request is overwritten by the succeeding requests before being flushed to the iSCSI storage, or three memory copying operations when the data of the request is purged to the iSCSI storage eventually.

Most modern computer architectures, such as Intel's Pentium series, only support primitive load/store instructions for memory to memory copying, which, as a result, consume a significant amount of CPU resource because of the required CPU intervention [3]. The main task of an NFS server is to move data from disk to the requesting clients, and thus involves heavy memory copying in this process. The drastic performance increase in CPU significantly improves the protocol processing part of the NFS server, but not as much in memory copying, which needs to touch memory byte-by-byte and thus does not scale well with the CPU clock rate. Table 1 shows the memory copying performance of a representative PC as of December 2002. The amount of data copied in this measurement is much larger than L1 and L2 caches, and the performance effect of caching is completely eliminated.

Given that memory copying is becoming more and more expensive in terms of number of CPU cycles required, it is imposing a greater and greater burden on iSCSI-based NFS servers. For example, based on the measurement shown in Table 1, the data copying part of a TCP-based data transfer session consumes between 45.4% to 68% CPU resource, assuming the session's throughput is 50 MBytes/sec. This is in addition to other CPU-consuming tasks, such as network interrupt handling, TCP processing [4], NFS protocol handling, iSCSI processing and other related system tasks. Adding these tasks together will easily saturate the CPU and thereby degrade the overall throughput of NFS servers that are based on iSCSI storage.

An effective solution to the performance problem of iSCSI-based NFS servers, like many low-latency messaging work, is to reduce the number of memory copying operations.

4 Network-Centric Cache Organization for Pass-Through Servers

Conceptually, an iSCSI-based NFS server should behave like a router, in that it simply forwards packets from iSCSI targets to NFS clients. We call this type of servers *pass-*

CPU	Memory	Throughput
Pentium-III 1 GHz 256 KB cache	512 MB PCI-133 ECC SDRAM	220.47 MB/sec

Table 1. Memory copying performance of a representative PC as of December 2002

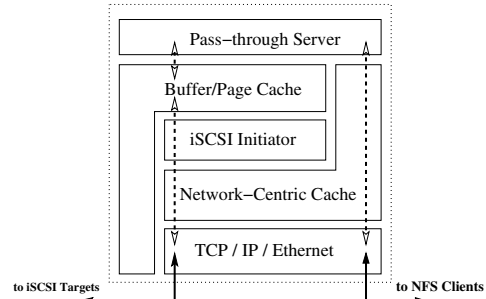


Figure 2. Software modules and data path within pass-through server based on iSCSI storage with network-centric cache.

through servers because their job is to channel data between two parties. So it should be relatively straightforward to emulate the single data copying behavior of modern routers. However, there are complications. First, although an iSCSI-based NFS server does not need to “touch” most packets’ contents, it does need to interpret those packets that correspond to NFS meta-data, examples of which include inodes, directory files and the superblocks of the file system. Fortunately the majority of the bytes passing through an iSCSI-based NFS server do not require any additional processing beyond forwarding. These bytes only need to be copied into and out of the memory once. Second, an iSCSI-based NFS server, unlike other pass-through devices, caches data. Moreover, the organization of this cache is typically dictated by the file system. To achieve single data copying, we need to eliminate any data copying between the network stack and the page/buffer cache.

Recognizing that *all* data cached on a pass-through server will eventually be sent out or replaced, we propose that the buffer/page cache in a pass-through server be organized in a network friendly format, or the concept of *network-centric cache organization*. By organizing the cached data in a network-ready form, when a data packet arrives, it is read into the network stack and cached in the network-centric cache without any modification. For example, such a packet is stored as a `sk_buff` list in the Linux kernel. When a cached data item is to be sent out, it is retrieved directly from the network-centric cache and trans-

mitted through the network stack.

There are several important performance benefits from organizing the page cache of a pass-through server in a network-centric form. First and foremost, it eliminates unnecessary data copying within the pass-through servers. The dashed lines with arrows in Figure 2 correspond to data copying operations that would have taken place in a pass-through server without network-centric cache. As a side benefit of avoiding unnecessary data copying, the total memory usage is reduced. Second, as cached data is stored in a network ready format, the amount of work required to send out a cached data item is also reduced substantially. For example, the protocol headers do not need to be repeatedly allocated and deallocated, as they are also stored in the cache. Moreover, the checksum of the cached payload can be either pre-computed or inherited from the payload's originator, and does not need to be calculated again and again every time it is sent out.

There is one important question to answer on the way to implement the network-centric cache organization. It is about whether the network-centric cache should be unified with the current buffer/page cache or not. Ideally, these two caches should be combined together as it would give a consistent view to pass-through servers. However, as the buffer/page cache stores data in fix sized pages whereas network-centric cache is organized as network buffer lists, whose sizes vary at large, this approach would incur significant changes to the existing file cache and the cache replacement procedure. For this reason, we choose to put the network-centric as a distinct part, which behaves just like a secondary cache to the buffer cache.

Basically, as a separate component, the network-centric cache interacts with the buffer cache, the networked storage driver, i.e., iSCSI initiator, and the pass-through servers. When there are network packets coming in, either in response to the access to the networked storage from the storage server, or as write-like requests destined to the pass-through server, in the form of network buffer list, the network-centric cache intercepts them. Then, the network-centric cache organizes these packets by indexing them using what we call *logical block unit (LBU) keys*. A LBU key can be a logical block number (LBN), if the packets carry data from the iSCSI storage, or an application-specific identifier when the packets hold data written to the pass-through server. An example of keys for the latter case is a key of a file handler plus an offset value when the packets are associated with a NFS WRITE request. Instead of copying the real data stored in the packets across different modules, which we call *physical copying*, only the associated LBU keys are copied among the modules. We call this copying the *logical copying*. As a LBU key only needs several bytes of space and is much smaller than a LBU (normally 4 KBytes), logical copying requires very little CPU resources. When a

page cache entry is sent out, either as a disk write to the networked storage or as response to a read-like request from the clients of the pass-through server, the network-centric cache module uses its LBU key to retrieve the corresponding entry from its own cache, and substitutes it for the copy from the file cache or the pass-through server.

In many cases, the size of the network packet that fits in one network buffer does not match the size of the data block used in disk access, which also corresponds to the granularity of remote storage access. For example, the MTU of an Ethernet packet generally is 1500 bytes, whereas a typical disk access size is 4 KBytes. Because of this mismatch, it is possible that a network buffer may overlap with multiple logical disk blocks. As a result, to ensure that the network buffers associated with a logical disk block can be handled as a single unit, the network buffers overlapping with different logical disk blocks need to be splitted into multiple ones, each only containing the data of a single logical disk block. Obviously, this network buffer break incurs overhead, including data copying, checksum calculation as well as network buffer allocation. To eliminate these unnecessary operations due to the size mismatch between the network buffer and the logical disk block, we developed a software jumbo-frame (SJF) scheme. which allows the network layer, e.g., IP, to send and receive packets whose sizes exceed the MTU of the physical network *without requiring additional data copying*. SJF is implemented as a software layer sitting between the IP layer and the network interface card (NIC) driver and does IP packet fragmentation and defragmentation. With SJF, a network buffer structure stores a chain of network buffers whose aggregation payload size matches the logical block size exactly.

Apparently, our scheme is not without problems. Having the network-centric cache separated from the buffer cache, the real data is cached in the network-centric cache while the LBU keys are stored in the buffer cache. As we choose to keep the buffer cache intact, an entire buffer page could just store the LBU key of the corresponding logical disk block. Hence, the memory usage might be doubled. However, this memory overuse can be easily resolved by bounding the buffer cache size (or even disabling the buffer cache), which can be done in most modern operating systems, such as Linux, FreeBSD and Windows NT. Though a small buffer cache could lead to more cache misses, these misses eventually will be taken care of by a much larger network-centric cache effectively. In the same time, modularization of the network-centric cache makes it easy to port to various platforms, even though they may have quite different page/buffer cache organizations.

5 Potential Performance Gain

5.1 Prototype

We have implemented the network-centric cache buffering scheme in Linux with NFS server as the pass-through server using iSCSI storage. One issue specific to the NFS server is that the NFS server needs to interpret a small portion of data passed-through. These data generally are meta-data, which correspond to the inodes information, the content of directory file or superblocks. For this reason, we need to distinguish the packets that the network-centric cache intercepts depending on the type of data the packets are carrying. If the packets happen to convey the meta-data, the network-centric cache simply copies the meta-data to the buffer cache and drops the packets, exactly the same way as in a system without the network-centric cache. If the packets carry the regular data, the network-centric cache will store them and do logical copying, as we described before. This differentiation of meta-data from regular data is done by checking the backing store type of the page in disk requests passed from the buffer cache, or the type of NFS requests.

As the actual data content are stored in the network-centric cache while the corresponding LBU keys in the system buffer cache, possibly with different cache policies, the inconsistency may happen between these two caches. In our prototype, we take care of this issue as follows: when it misses in the system buffer cache but hits in the network-centric cache, the local file system will issue disk requests, which eventually hit in the network-centric cache; if it hits in the system cache but misses in the network-centric cache, the network-centric cache will retrieve the real data from the remote storage server on behalf of the requesting process.

5.2 Measurements and Analysis

Our testbed setup for the preliminary prototype comprised of a storage server, an NFS server, and four NFS clients. Both the servers and the clients were Pentium-III 1GHz machines. The storage server had a 1GB of memory and all other machines had 512 MB of memory. The disk on the storage server was a RAMDISK of size 750 MB. We used RAMDISK to avoid real disk performance bottlenecks. All machines were equipped with Intel PRO/1000 MT Gigabit server adapters and were connected to a Gigabit switch. The operating system kernel used was Linux 2.4.19. The NFS server kernel was modified to support network-centric caching mechanism. The NFS server was based on NFS version 3 and the data was accessed over UDP. The iSCSI implementation used was based on the reference implementation [2] by the InterOperability Lab.

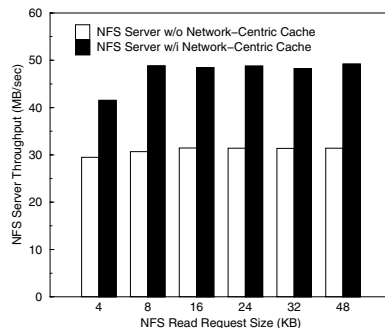


Figure 3. Performance throughput for different NFS request sizes with and without network-centric cache organization.

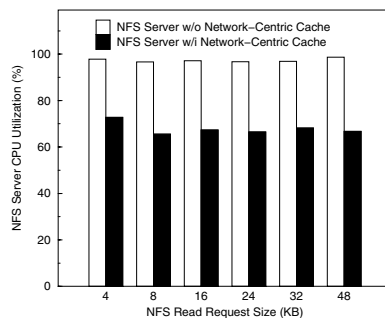


Figure 4. CPU utilization for different NFS request sizes with and without network-centric cache organization.

We measured the performance of our setup for the maximum NFS server throughput and the CPU utilization for varying NFS read request sizes. The request sizes ranged from 4 KB to 48 KB. The performance comparison was done for NFS server with and without network-centric cache implementation. As shown in Figure 3, the throughput for normal NFS server ranged from 30 MB/s for 4 KB request to around 32 MB/s for 48 KB request. Under the same setup, the NFS server with network-centric cache organization had a sustained throughput as high as around 50 MB/sec for 48 KB request. In terms of CPU utilization, which result is shown in Figure 4, the plain NFS server saturated the CPU, while it ranged from 70% at 4 KB request size to 60% for 48 KB request size for NFS server with network-centric cache organization. Thus, network-centric cache organization yields a throughput gain of over 50%, while consuming about 40% less of CPU resource.

6 Conclusions

This paper first illustrates the data flow of a standard iSCSI-based NFS server and its performance overhead due to redundant data copying. Based on the analysis, we propose the design of a network-centric buffer cache organization for pass-through servers, e.g., NFS servers. This network-centric cache organization improves the performance of data transmission through pass-through servers by avoiding unnecessary data copying, and by storing the data in a network-ready form. The structure of this cache organization allows it to be readily applied to most modern operating systems. To resolve the size mismatch of network buffer with disk logical block unit, a software jumbo-frame protocol is developed, which can be implemented as a loadable module sitting between the network layer and the network interface driver. Experimental results from the NFS server on Linux with a preliminary prototype of network-centric cache exhibits more than 50% gain of throughput, while with 40% less of CPU consumption, in comparison of a normal Linux case.

We are working on building a fully operational network-centric cache organization scheme tailored to NFS server running on Linux, and evaluating its impact on NFS server performance as well as cache effectiveness under realistic experiment setups. Also, due to the encouraging results from the NFS server, we are planning to explore the impact of the network-centric cache organization on other pass-through servers, such as media server or FTP server using networked storage.

Acknowledgment

We would like to thank the anonymous referees for their helpful comments. This research is supported by NSF awards EIA-9818342, ANI-9814934, and ACI-9907485, and USENIX student research grants.

References

- [1] iscsi. <http://www.ietf.org/internet-drafts/draft-ietf-ips-iscsi-19.pdf>.
- [2] iscsi reference implementation. <http://www.iol.unh.edu/>.
- [3] H. K. J. Chu. Zero-copy TCP in solaris. In *USENIX Annual Technical Conference*, pages 253–264, January 1996.
- [4] D. Clarke, V. Jacobson, J. Romkey, and H. Salwan. An analysis of tcp processing overhead. *IEEE Communications Magazine*, 27(6):23–29, June 1989.
- [5] P. Druschel and L. L. Peterson. Fbufs: A high-bandwidth cross-domain transfer facility. In *Symposium on Operating Systems Principles*, pages 189–202, 1993.
- [6] S. Microsystems. Nfs: Network file system protocol specification. IETF RFC 1094, March 1989.
- [7] V. S. Pai, P. Druschel, and W. Zwaenepoel. IO-Lite: a unified I/O buffering and caching system. *ACM Transactions on Computer Systems*, 18(1):37–66, 2000.
- [8] J. P. G. Sterbenz and G. M. Parulkar. Axon: A high speed communication architecture for distributed applications. In *Proc. of IEEE INFOCOM 1990, Wash., D.C.*, Jun 1990.