

Copy Avoidance in Networked File Systems

Jose' Carlos Brustoloni
Bell Labs
Lucent Technologies

GBN'98 - IEEE ComSoc TCGN - March 1998 - San Francisco, CA

1

Emulated copy for file I/O?

Emulated copy: Copy avoidance scheme for network I/O [OSDI'96].
Uses TCOW, input alignment, and page swapping
to preserve copy semantics.

Network I/O: server buffers usually are *ephemeral*.

File I/O: server buffers often are *cached*.

→ Mismatches:

- TCOW: server loses reference and can't cache client pages after request processing completion.
- Input alignment and page swapping: deplete or corrupt server buffers with previous contents of client buffers.

2

The question of whether emulated copy can be used for file I/O was proposed to me when I was finishing my Ph.D. at Carnegie Mellon. I had demonstrated emulated copy primarily as a copy avoidance scheme for network I/O.

Emulated copy outputs data directly from client buffers. To preserve copy semantics, emulated copy uses TCOW, a reference counting mechanism that guarantees that client pages cannot be overwritten during output processing.

On the other hand, to prevent corruption of client input buffers, emulated copy inputs data into distinct server buffers with the same page offset and length as the corresponding client buffers. After successful input, emulated copy then swaps pages between server and client buffers.

Although not specific to network I/O, emulated copy does assume that server buffers are ephemeral, that is, are each deallocated when processing of the request that uses it completes. Ephemeral buffers are typical of character I/O.

On the contrary, file I/O usually involves cached server buffers, each of which may remain allocated long after processing of the first request that uses it. Cached buffers are typical of block I/O.

If used on file output, emulated copy would not allow the file system to cache the data without copying, because TCOW's references to client pages are lost when output completes. On the other hand, if used on file input, emulated copy would deplete or corrupt the file system's cache by swapping its pages with old client pages.

Mapped file I/O

Widely available scheme for copy avoidance in file I/O.

Map file = allocate new client region and map server cache pages to client region on exception basis.

No copying if page pools for VM and buffer cache are the same.

3

Although emulated copy may not provide adequate copy avoidance for file I/O, a well-known solution is widely available: mapped file I/O.

Mapping a file requires only allocating a new region in the client's address space. Pages from the file system's cache are then mapped to the client region on an exception basis, i.e., when actually accessed by the client.

File unmapping causes those pages to be unmapped and the client region to be deallocated.

No copying is necessary if virtual memory and file system cache pages are allocated from the same pool.

Interoperation of emulated copy and mapped file I/O

Mapped file to network - no problem.

Network to mapped file - problematic because of alignment constraints:

- If file offset is multiple of page size, data must be received in page-aligned buffer.
- Complicated because of application-layer headers:
 - Neither trailer encapsulation nor header/data splitting strip application-layer headers.
 - Early demultiplexing and buffer snap-off [INFOCOM'97] strip such headers but are not widely available.

4

The question now becomes whether emulated copy and mapped file I/O interoperate well, each providing copy avoidance for certain kinds of I/O.

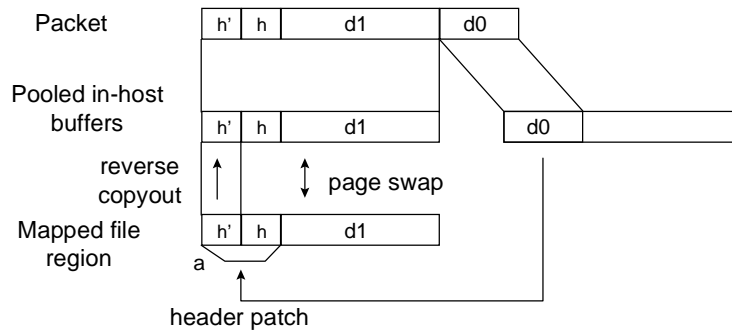
From the point of view of emulated copy, a mapped file region is a region like any other, and its contents can be output using TCOW.

Input with emulated copy's page swapping may be problematic, however, because mapped file I/O imposes a particular alignment for data. For example, assuming that file offset is a multiple of the page size, data would have to be received from the network into page-aligned buffers for it to be possible to swap pages into the client's mapped file region.

This alignment can be hard to achieve because an application-layer header may precede the data. This header may specify, for example, the file, file offset, and length of the data. Common solutions for stripping packet headers -- trailer encapsulation (software) or header/data splitting (hardware) -- strip only transport and lower-layer headers. They do not distinguish between application-layer headers and data.

Special network adapter features -- early demultiplexing and buffer snap-off -- can be used to strip application-layer headers. However, such features are not widely available.

Header patching



→ Page alignment restored even without adapter support.

5

Header patching is a new optimization for restoring data alignment without special network adapter support. Header patching requires collaboration of sender and receiver, stripped or fixed-length transport and lower-layer headers, and MTU greater than or equal to the page size. Copy avoidance without these restrictions requires special adapter support -- early demultiplexing, buffer snap-off, or outboard buffering.

The figure shows how data of length equal to the page size can be received into a mapped file region, using header patching.

The sender transmits the initial part of the data, $d0$, after the remainder of the data, $d1$. $d0$ has length equal to the sum of the lengths of the application (h) and unstripped lower-layer (h') headers.

The receiver first peeks at the application header, determines the data's address a in the corresponding mapped file region, and then inputs $h+d1$ bytes to address $a+h'$, followed by $d0$ bytes to address a . The first $h+d1$ bytes are passed by reverse copyout of h' bytes and page swap. The following $d0$ bytes are then patched on top of the headers at address a . The total amount of copying is $2h'+h$, which is typically much less than the page size. After patching, the data starts at the correct address and runs uninterrupted in correct order.

Summary

- Different copy avoidance schemes may be necessary for network and file I/O (ephemeral vs. cached server buffers).
- Emulated copy interoperates well with mapped file I/O.
- Header patching provides copy-free interoperation even without adapter support.
- Data can be passed between networks and file systems without copying and without changing existing APIs.

6

The performance of networked file systems can be significantly improved with copy avoidance. Because of cache effects, the benefit of combining emulated copy and mapped file I/O is greater than the sum of the benefits of each optimization alone. Benefits are greatest when copying is avoided on the entire end-to-end data path.

There is no need to change existing APIs to achieve these benefits.