

The Performance of the Container Shipping I/O System

Eric W. Anderson
Joseph Pasquale
{ewa, pasquale}@cs.ucsd.edu

Department of Computer Science and Engineering
University of California, San Diego
9500 Gilman Drive MS 0114
La Jolla, CA 92093-0114

The Container Shipping I/O system [1] provides fast, copy-free I/O to user-level processes in a uniform, device-independent way. We have refined and implemented the Container Shipping design [2], and have measured its performance for local IPC, FDDI datagrams, and video display [1, 3]. Measured throughput with Container Shipping is up to seven times greater than with conventional Unix I/O.

Container Shipping is an I/O data transfer mechanism composed of six system calls. Implemented as a supplement to or a replacement of a user-level I/O API, Container Shipping provides copy-free I/O through selective virtual memory remapping. Unlike related efforts, Container Shipping is not limited to one type of device, does not restrict access to data, and does not perform unnecessary memory mappings or copy-on-write operations.

Container Shipping unconditionally eliminates copies by moving all data with virtual memory remapping. I/O data (in blocks of memory called Containers) is mapped in and out of process address spaces. Copy elimination provides a large reduction in latency and a corresponding improvement in throughput. Container Shipping also eliminates unnecessary memory mapping operations, by mapping data only as requested by a process, thereby providing additional latency and throughput gains. Because mapping is directed by user request, no expensive page faults are required to access data. Finally, Container Shipping makes possible additional optimizations which exploit the predictable behavior of long-lasting I/O pathways, for further performance gains.

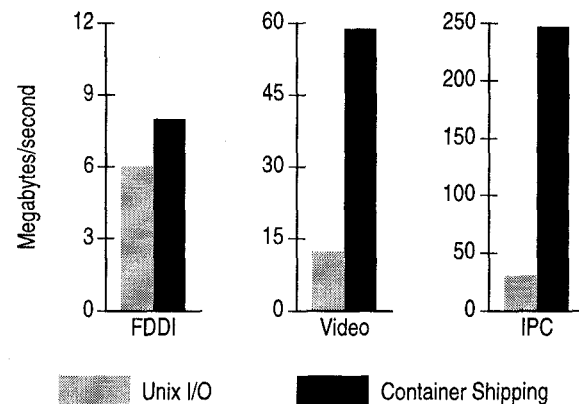
We have implemented Container Shipping alongside the traditional Unix I/O API in DEC's OSF/1 v2.0 operating system. Tests were conducted by running our modified kernel on DEC 3000 model 800 ("Alpha") workstations. First, we measured local IPC performance between two processes. These processes exchanged 32 megabytes of data through AF_UNIX sockets, so source and target memory was largely uncached. Container Shipping provided over 700% more throughput than conventional Unix I/O.

Permission to make digital/hard copy of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage, the copyright notice, the title of the publication and its date appear, and notice is given that copying is by permission of ACM, Inc. To copy otherwise, to republish, to post on servers, or to redistribute to lists, requires prior specific permission and/or a fee.

SIGOPS '95 12/95 CO, USA
© 1995 ACM 0-89791-715-4/95/0012...\$3.50

Local IPC I/O makes use of no actual device. To demonstrate the viability of Container Shipping for real devices, we connected three workstations using two FDDI networks, with one system forwarding data from one network to the other. Despite high overheads from context switching, system calls, protocol processing, and device driver interrupt handling, Container Shipping provided up to 30% more throughput than conventional I/O on a system under load. With Container Shipping, a user-level process was able to forward data reliably at the network limit, while with Unix I/O a significant fraction of the packets were lost.

To demonstrate Container Shipping with a very high speed device, we implemented local video display to a frame buffer. Six times faster than the FDDI interface, the frame buffer is a true device, accessed by DMA transfer. Container Shipping could sustain over 59 megabytes per second to the display (enough for six full-size full-motion color video windows), while Unix I/O could manage at most 12 megabytes per second.



Measured throughput of Container Shipping and conventional I/O in DEC OSF/1 on DEC 3000 model 800 workstations, for devices of three different types.

We have demonstrated substantial performance improvements obtained by using Container Shipping instead of conventional Unix I/O. Container Shipping is a viable alternative to copy-intensive user-level I/O interfaces.

References

1. Eric Anderson, "Container Shipping: A Uniform Interface for Fast, Efficient, High-bandwidth I/O", Ph.D. dissertation, Department of Computer Science and Engineering, University of California, San Diego, July 1995.
2. Joseph Pasquale, Eric Anderson, and P. Keith Muller, "Container Shipping: Operating System Support for I/O-Intensive Applications", IEEE Computer, Volume 27, Number 3, pp. 84-93, March 1994.
3. Eric Anderson and Joseph Pasquale, "The Performance of the Container Shipping I/O System", Technical report CS95-441, Department of Computer Science and Engineering, University of California, San Diego, August 1995.