Synchronized Send Operations for Efficient Streaming Block I/O over Myrinet

Evangelos Koukis, Anastassios Nanos and Nectarios Koziris
{vkoukis, ananos, nkoziris}@cslab.ece.ntua.gr

National Technical University of Athens
School of Electrical and Computer Eng.
Computing Systems Laboratory

CAC 2008, Miami, FL, April 14, 2008
Outline

- Introduction
  - The need for an efficient disk sharing layer
  - The gmblock direct disk-to-NIC framework
- Synchronized Myrinet/GM sends
  - Disk with network I/O overlapping
- Experimental evaluation
- Conclusions - Future Work
Outline

◆ Introduction
  ➡ The need for an efficient disk sharing layer
  ➡ The gmblock direct disk-to-NIC framework

◆ Synchronized Myrinet/GM sends
◆ Experimental evaluation
◆ Conclusions - Future Work
Introduction

- Scalable shared block-level storage
  - Shared-disk filesystems over shared-nothing architectures

- gmblock: nbd over Myrinet
  - direct disk-to-NIC transfers over PCI
  - block-device driver independent
  - IPDPS 2007
Shared-disk cluster filesystems

Cluster interconnect (Myrinet)
Generic nbd implementation

- Application
- VFS
- Linux blockdev layer
- nbd client
- NIC

User

- POSIX syscalls

Kernel

- nbd userspace server
- VFS
- Linux blockdev layer
- block device driver
- Storage device

Hardware
gmblock datapath (logical)

User
- userspace nbd server
- POSIX I/O
- TCP
- VFS
- netdev drv
- block layer
- blockdev drv

Kernel
- userspace buffer (mapped)
- page cache
- TCP
- IP
- block layer
- blockdev drv

Hardware
- SRAM
- NIC
- Storage
Introduction

Synchronized Myrinet/GM sends
  ➔ Disk with network I/O overlapping

Experimental evaluation

Conclusions - Future Work
GM synchro sends: Motivation, goals

- I/O rate lower than expected
  - lags behind disk and network bandwidth
- Need to work around architectural limitations
  - RAID storage bandwidth vs. request size
  - limited SRAM on NIC, few ops on the fly
  - poor request pipelining
- Proposed solution: overlapping disk with network I/O for a single request
  - Synchronized GM sends, to ensure correctness of data on the wire
Sustained throughput (base)

- Disk to SRAM
- Disk to RAM
- GMBlock over SRAM
- GMBlock over RAM
- GMBlock SRAM, no disk
- GMBlock RAM, no disk

Bandwidth (MB/s)

Request Size (KB)
The effect of request size

- Request size limited by SRAM on NIC
- There is a performance tradeoff
  - Fewer, larger requests
    - Better spot on the 3Ware bandwidth curve
    - But causes idle periods due to worse pipelining
  - More, smaller requests
    - Better pipelining due to many in-flight requests
    - Worse spot on the 3Ware bandwidth curve
Proposed Solution

- Improve intra-request overlapping for read operations
  - Overlap disk read with network send for a single large request
  - Stream data from disk to wire over the Lanai
- GM send overlapped with block I/O
  - Unsafe, depends on disk and network I/O rates
- Synchronized send operations
  - Ensure that only valid data are sent over the wire
Synchronized GM sends (1)

- NIC works in lockstep with external agent
  - Send tokens marked as *synchronized*
  - Token ignored when disk fails to keep up

- Design goals
  - Minimal overhead, no CPU intervention
  - Seamless integration into existing GM framework
  - Compatible with GM send semantics
  - Independent of block device type and driver
Synchronized GM sends (2)

Non-overlapped network I/O

Synchronized sends for overlapped network I/O
Implementation Issues (1)

- Need Lanai notification for external writes into SRAM
  - memory divided in chunks, e.g., 4 kB
  - "dirty memory" bitmap, bits set by hardware

- Software emulation
  - 32-bit intermediate markers, one per chunk
  - polling to detect markers being overwritten
  - one extra marker used to ensure correctness in the worst case ($P \approx 6 \times 10^{-8}$)
Implementation Issues (2)

- Integration with existing GM services
- Initialization
  - `gm_synchro_prepare_buffer()`, `gm_synchro_send_with_callback()`
- Transmission phase
  - The firmware polls the markers, updates “valid data” counter
- Finalization
  - `gm_synchro_finalize_buffer()`
Outline

- Introduction
- Synchronized Myrinet/GM sends
- Experimental evaluation
- Conclusions - Future Work
Experimental evaluation platform

- Pentium III@1266MHz, 1GB PC133 SDRAM
- Serverworks ServerSet III HC-SL
- Myrinet M3F-PCIXE-2, 64-bit/66MHz PCI
- 3Ware 9550SXU-16 SATA RAID controller
- Linux kernel 2.6.16.5
- GM-2.1.26 with custom modifications

Base gmblock vs.
gmblock with GM synchro sends
Sustained throughput (synchro)

Bandwidth (MB/s) vs Request Size (KB)

- disk to sram
- disk to ram
- gmblock over sram
- gmblock over ram
- synchro over sram
- synchro, no disk I/O

RAID-0 artifact
Latency breakdown per req. size

- **STATE_FINALIZE** (return receive buffer)
- **STATE_SEND** (non-overlapped send time)
- **STATE_SEND_INIT** (prepare buffer, post event)
- **STATE_READ** (block read)
- **STATE_INITIALIZE** (wake, unpack)

Ram, sram, synchro, bogus_synchro
Outline

- Introduction
  - The need for an efficient disk sharing layer
  - The gmblock direct disk-to-NIC framework
- Synchronized Myrinet/GM sends
- Experimental evaluation
- Conclusions - Future Work
Conclusions - Future Work

- Generic framework for NIC synchronization with external agents
  - has other applications, e.g. video streaming
- Evaluate performance on newer PCI-X / PCI Express
- End-to-end operation, performance?
- Parallel fs over gmblock
  - gmblock visible as Direct-attached Storage
  - will allow evaluation of performance with real applications running on cluster
- Server-side data manipulation?
  - Encryption, compression on intelligent NICs
Thank You!

Questions?