Parallel I/O

Kai Shen

I/O for Parallel Programs

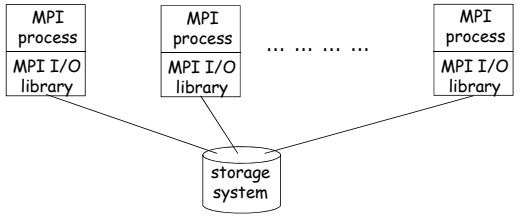
- Reading/writing large amount of data from/to storage
  - Parallel processes read a large matrix of data
  - Parallel processes write output statistics
- Using a dedicated I/O process
  - All I/O is issued at the dedicated I/O process; other processes communicate with the I/O process for I/O
  - Problem: scalability

MPI-I/O

- Often, each process reads/writes from/to a distinct file/data partition
- MPI-I/O
  - providing data structures to partition data, using standard interface to ease programming
  - filetype, displacement, view, offset
  - http://www.mpi-forum.org/docs/mpi-20-html/node173.htm#Node173
  - Limitation: not very helpful for accessing irregular data structures (sparse matrices)

MPI-I/O Performance Enhancements

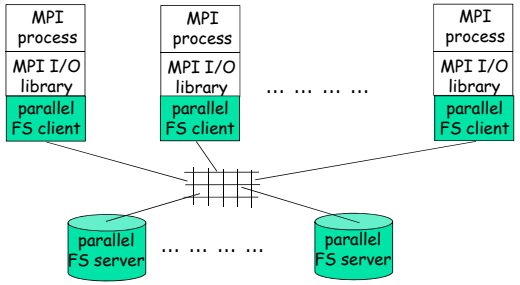
- Improve I/O sequential access patterns:
  - Data sieving – read more (sequentially) than what you need
  - Collective I/O – while each's view is segmented, combined view from all is sequential



I/O Parallelism

- So far, parallelism in issuing I/O operations
  - Still not scalable, e.g., bound by throughput of one I/O device
- Next, parallelism in performing I/O (transparent to users)
  - Intra-device parallelism: RAID
  - Inter-device parallelism: storage cluster, parallel file system

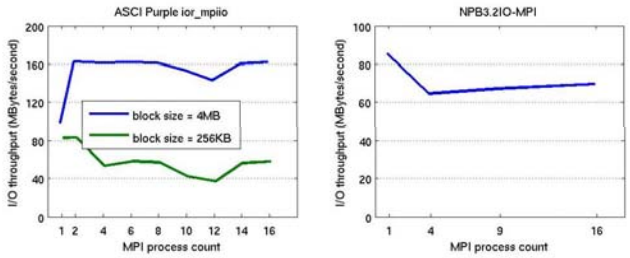
Parallel Storage and Parallel FS



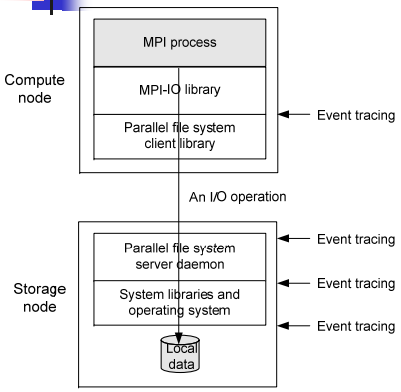
- Parallel file systems (GPFS, PVFS, Lustre)
  - data striping, redundancy encoding, … …

A Quantitative Example

- Up to 16 compute nodes running MPICH2 (MPI-IO)
- 6 striped storage nodes running PVFS2; each run Linux 2.6.12
- Gigabit Ethernet (~80us TCP/IP roundtrip latency)



Problem Diagnosis – I/O Trace Collection

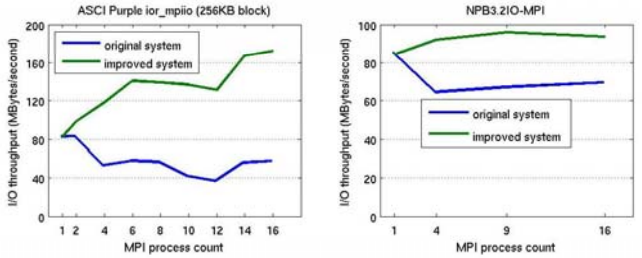


- Many layers of software present possible sources of problems
- Tracing at multiple layer boundaries of I/O stack help pinpoint problem sources
- Traced I/O characteristics
  - Sequentiality of I/O access pattern
  - I/O request size/granularity
  - I/O workload concurrency/idleness
  - I/O throughput
  - Wait between causal I/O events

Results of Trace Analysis

- Result #1: interleaved I/O under concurrent operations
- Further analysis within the operating system
  - prefetching in general-purpose OS is insufficient
  - anticipatory I/O scheduling does not work properly due to the lost of remote process context at storage nodes
- Result #2: slow return of I/O that should hit the cache
- Further analysis within the C library
  - PVFS uses one open file to issue I/O operations on the same file
  - all asynchronous I/O operations using the same open file are serialized by the C library

Performance Results After Problem Fixes



- Resolved anomalous performance degradations
- 39-156% throughput improvement for four applications

Summary on Parallel I/O Performance

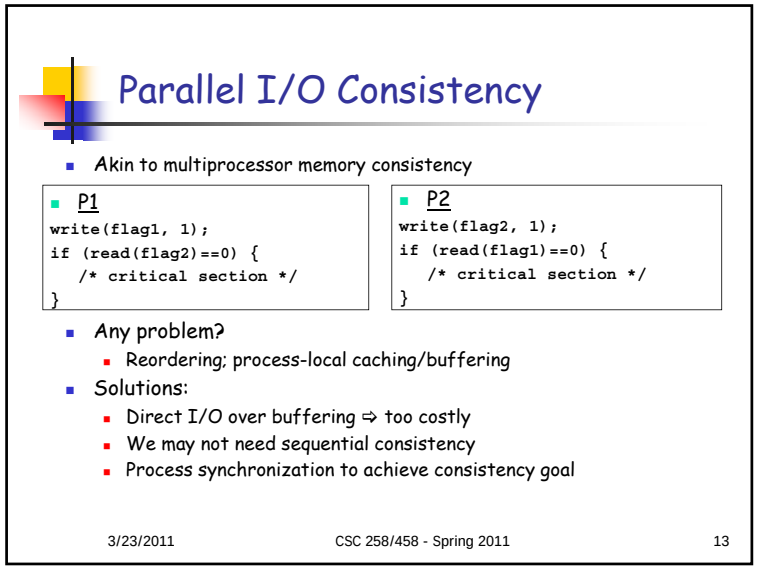
- Parallel I/O may deliver scalable performance
  - constrained by underlying storage device and networking performance
- Also, performance problems exist in parallel I/O systems
  - multiple layers of software interacting in complex ways
  - performance semantics are not well exposed through layer interface
  - problems often relate to parallelism and concurrency in the system

Embarrassingly Parallel I/O

- FAWN
  - Server apps without inter-request dependencies
  - Parallel I/O on compact Flash over many wimpy nodes

  ⇨ energy efficiency
- MapReduce
  - Parallel map functions with little dependency at the end
  - Suitable for large cluster of machines

  ⇨ extreme scalability, worry of component reliability


Parallel I/O Consistency
Akin to multiprocessor memory consistency
P1
write(flag1, 1);
if (read(flag2)==0) {
/* critical section */
}
P2
write(flag2, 1);
if (read(flag1)==0) {
/* critical section */
}
Any problem?
Reordering; process-local caching/buffering
Solutions:
Direct I/O over buffering ⇨ too costly
We may not need sequential consistency
Process synchronization to achieve consistency goal
3/23/2011
CSC 258/458 - Spring 2011
13