MPI Implementation

Kai Shen

Message Passing Interface

- De facto standard programming interface for message passing-based parallel programs
  - No shared memory
- Communications
  - Point-to-point: send/receive
  - Group communications: broadcast, gather, scatter, reduce, barrier

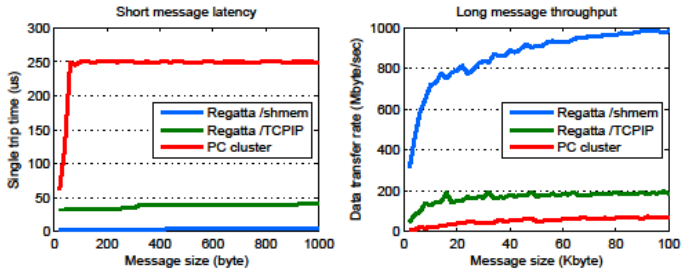
MPI Communications

- Cluster of machines ⇨ TCP/IP/Ethernet
- Performance issues with TCP/IP
  - Connection establishment
  - Congestion control
  - Lots of data copying in systems layers
- Advanced technologies
  - UDP or raw IP with some error detection management
  - RDMA to minimize data copying
- Performance issues with Ethernet
  - Legacy design for shared bus, slow networks
- Advanced technologies
  - Infiniband, Myrinet

MPI Communications in Shared Memory

- MPI communications in shared memory multiprocessors
- TCP/IP
- Shared memory accesses
  - Shared message queue and buffer
  - Synchronization to protect shared accesses
  - Contention may create scalability bottleneck
- Number of memory copying operations
  - two
  - or one?
  - or zero?

Communication Performance



Multiprocessors without Cache Coherence

- Each processor (or a sub-group of processors) has some local memory
- Custom network/bus allow fast access to remote memory
- No support for cache coherence, which allows the system to scale

⇨ Large multiprocessors from Cray, IBM, …

Pipelined Communications

- Heard it from Prof. Chen Ding
- Motivation
  - Large message filled over time at the sender
  - Incrementally useful at the receiver
  - Inefficient to send after the full message is filled
- Pipelined communications
  - Partial message is sent as soon as the data is available
  - When a part of the message arrives, the subsequent computation that only depends on this part of the data can proceed
- Problems
  - At the sender, how do we know that a part of the message is filled?
  - At the receiver, how do we know that a subsequent computation only depends on data that has already arrived?

Collective Communications: Broadcast

- Performance goal:
  - total bandwidth usage
  - latency to reach receivers (worst case or average)
- Many communication system supports native broadcast
  - ethernet broadcast
- Otherwise:
  - root sends to every receiver one by one
  - root sends to one other; both send to two more; then all reached nodes send to some unreached at each round; …
    - does the order matter?

Collective Communications: Reduce

- Approaches
  - all data sent to the root; reduced at root
  - tree-ordered parallel reduction (reduction op is associative)
  - adaptive order based on progress at each node (if op is commutative)

MPI without dedicated processors on Shared Memory Multiprocessor

- 4-processor machine; run 8 MPI processes
- What is going to happen?
- Benefit
  - better utilization despite load imbalance in static task assignment
- Drawbacks:
  - overhead of OS scheduling and context switches (particularly cache pollution)
  - terrible synchronization delays when the MPI implementation spin-waiting

Parallel Program Reliability

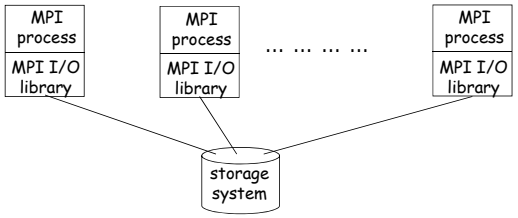
- Checkpointing and restart
- Distributed checkpoint is tricky.
  - The checkpointed snapshot must be *consistent*.
  - If A has sent out a message to B in a snapshot, it should've arrived at B in the snapshot
  - If A has not sent out the message, it should've arrived
- Checkpointing of different parallel processes need to be synchronized (or carefully ordered)

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 (file view, displacement, …)
  - http://www.mpi-forum.org/docs/mpi-3.0/mpi30-report.pdf (Sec 13.3)
  - limitation: not very helpful for accessing irregular data structures (sparse matrices)

MPI-I/O Performance Enhancements

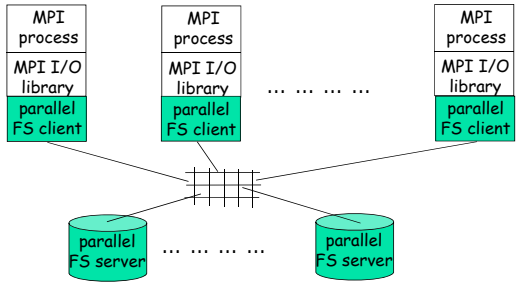
- Improve I/O sequential access patterns:
  - Data sieving – read more (sequentially) than what you need
  - Collective I/O – while each's view is segmented, combined view from all is sequential



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

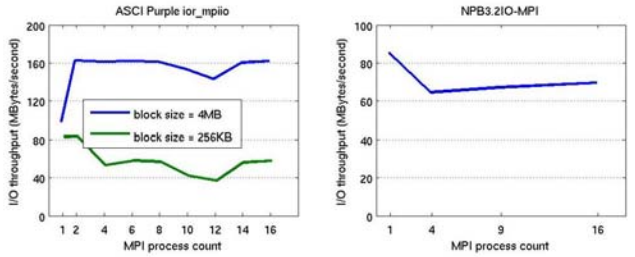
Parallel Storage and Parallel FS



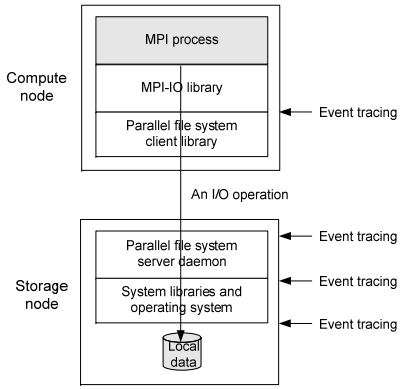
- Parallel file systems (GPFS, PVFS, Lustre)
  - data striping, redundancy encoding, … …

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

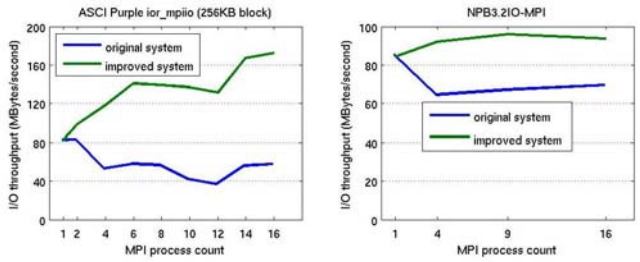


- Many layers of software present possible sources of problems
- Tracing at multiple layer boundaries of I/O stack help pinpoint problem sources
- Traced I/O characteristics
  - Sequentiality of I/O access pattern
  - I/O request size/granularity
  - I/O workload concurrency/idleness
  - I/O throughput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