Parallel Data Storage and File System

Kai Shen

Storage I/O for Big Data Applications

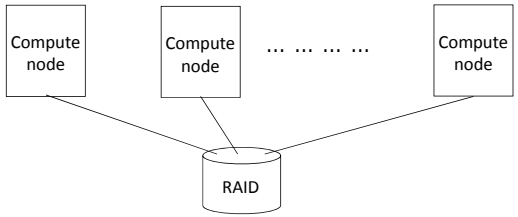
- When data is too large (or expensive) to fit into memory
  - The crawled web
  - Inverted web indexes for search
  - Online image repository
  - Movie databases
- Collected/output data that needs to be durable
  - Financial transactions, medical records
  - Outputs of data processing
- Characteristics
  - Reads (more often) and writes
  - Often (not always) in large chunks
  - Must be parallelized for high throughput and scalability

RAID

- Redundant Array of Inexpensive (Independent) Disks
- RAID supports parallelism and reliability for storage I/O
  - Parallelism through striping
  - Reliability through redundancy (mirror or redundancy coding like parity)
- RADI levels

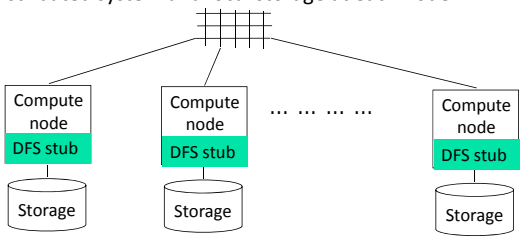
Parallel I/O Architecture

- RAID has limited scalability, constrained by a single I/O access point



Parallel I/O Architecture

- Distributed system and local storage at each node



- Parallel/distributed file system to organize the distributed storage resources
  - Again, parallel I/O performance through striping, and reliability through redundancy

Google File System

- Google File System (GFS) works on such assumptions
  - A modest number of large files (100s of MBs or GBs or more)
  - Reads are more often than writes
  - "Big data" accesses are common; throughput is more important than latency
  - Build on many cheap commodity components (don't buy exotic hardware, reliability through replication)
- Design/implementation
  - Chunk-based, a single large file contains multiple chunks which are distributed (allowing parallel accesses)
  - One master node and many chunk servers
  - Based on large chunks (64MB per chunk)

Reliability

- Google file system employs replication (default 3) for reliability, but consistent replication for mutable data is hard
  - Writes have to reach every replica
  - Must be done in certain ways (order matters), done in the same order on all replicas, or if the writes are commutative …
- Availability
  - Single master design requires quick, consistent recovery from failure
  - Replication of the master, but only one allows writes

HDFS

- Hadoop Distributed File System (HDFS)
  - Inferior to GFS, particularly poor support for mutable data replication
  - Strong integration with Hadoop, but incompatible with POSIX standard

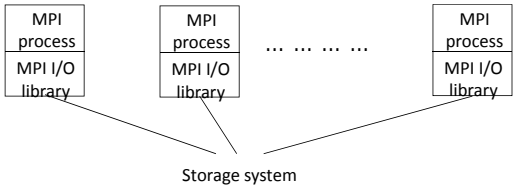
Implication for Big Data Programmers

- Few large files are better than many smaller files
- Large-grained, sequential accesses are better than piecemeal I/O accesses
- Reads are faster than writes
  - Because of replication
  - Especially if running on Flash storage
- Custom control of parallel/distributed file systems:
  - Chunk size (tradeoff?)
  - Replication degree (tradeoff?)

MPI-I/O

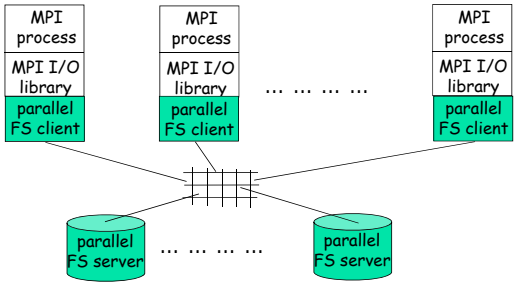
- In traditional MPI programs, each process reads/writes from/to a distinct file/data partition
- MPI-I/O
  - provide data structures to partition data, using standard interface to ease programming
  - filetype, displacement, view
  - http://www.mpi-forum.org/docs/mpi-2.2/mpi22-report/node274.htm#Node274
  - 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

MPI-IO on Parallel File Systems



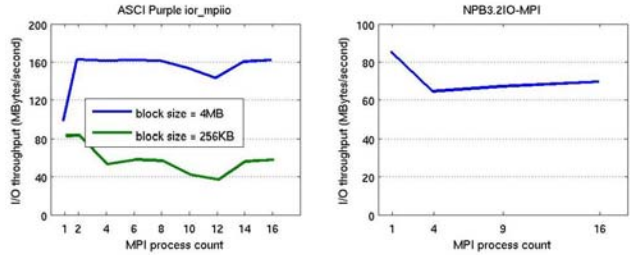
- Parallel file systems (GPFS, PVFS, Lustre)
  - data striping, redundancy replication, … …

GPFS

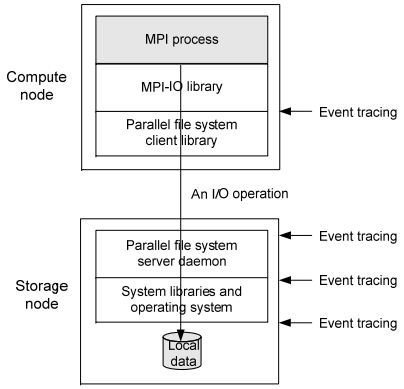
- IBM's General Parallel File System
- Earlier than GPS and HDFS
- Some key differences
  - Intended for full POSIX compliance, more general-purpose
  - Not intended for cheap, commodity components (e.g., specialized storage area network)
  - Location-transparent to applications
  - Smaller chunks

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

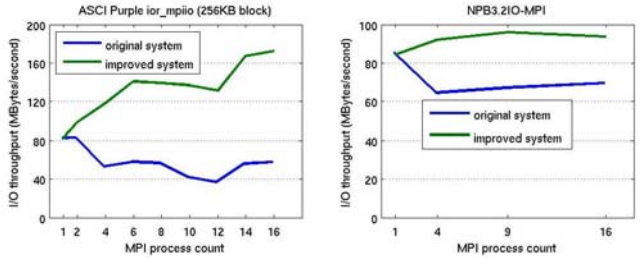


- Many layers of software present possible sources of problems
- Tracing at multiple layer boundaries of I/O stack help pinpoint problem sources
- Traced I/O characteristics
  - Sequentiality of I/O access pattern
  - I/O request size/granularity
  - I/O workload concurrency/idleness
  - I/O throughput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