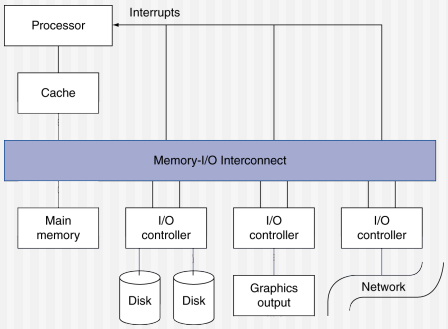
Lecture 7
The Impact of Data

- Storage and I/O
- Parallel File Systems
- Parallel I/O

1

Introduction

- I/O devices can be characterized by
  - Behavior: input, output, storage
  - Partner: human or machine
  - Data rate: bytes/sec, transfers/sec
- I/O bus connections



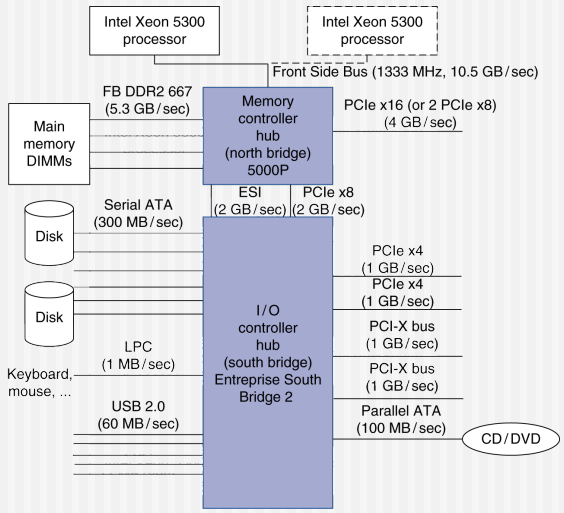
2

I/O System Characteristics

- Dependability is important
  - Particularly for storage devices
- Performance measures
  - Latency (response time)
  - Throughput (bandwidth)
- Desktops & embedded systems
  - Mainly interested in response time & diversity of devices
- Servers
  - Mainly interested in throughput & expandability of devices

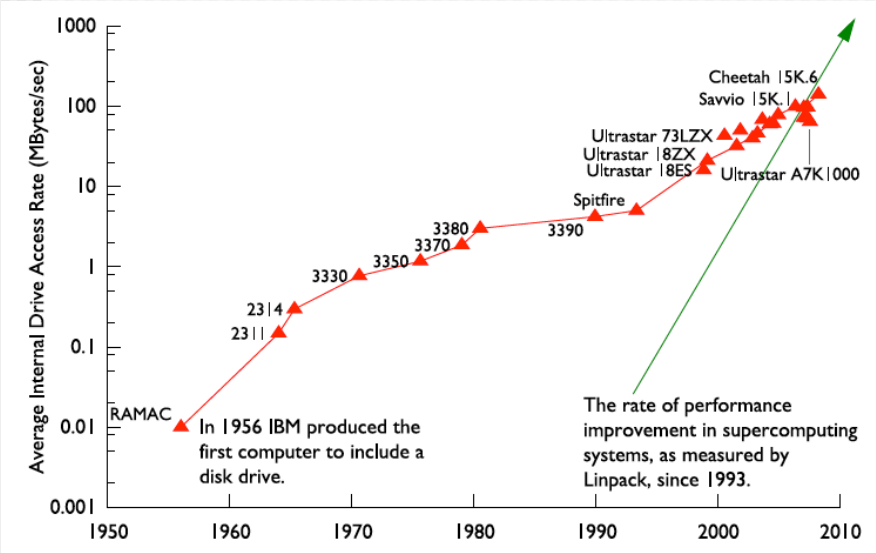
3

Typical x86 PC I/O System



4

I/O Performance



5

I/O vs. CPU Performance

- Amdahl's Law
  - Don't neglect I/O performance as parallelism increases compute performance
- Example
  - Benchmark takes 90s CPU time, 10s I/O time
  - Double the number of CPUs/2 years
    - I/O unchanged

Year	CPU time	I/O time	Elapsed time	% I/O time
now	90s	10s	100s	10%
+2	45s	10s	55s	18%
+4	23s	10s	33s	31%
+6	11s	10s	21s	47%

6

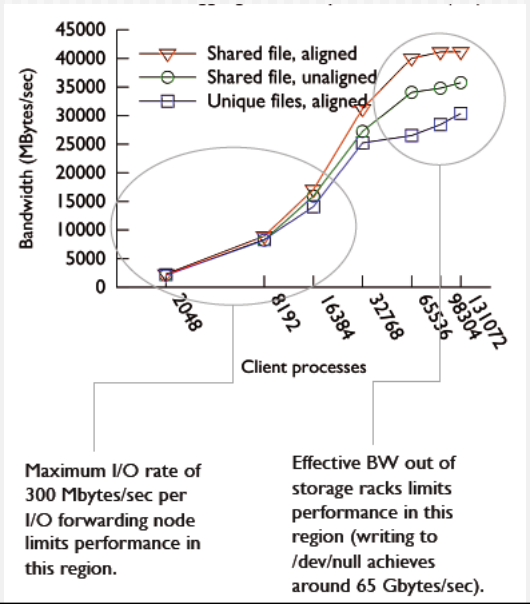
Need Parallel Storage Systems

- Parallelize I/O
- Increase bandwidth by striping files across disks and nodes

- Example parallel file systems
  - GPFS: General Parallel File System for AIX (IBM)
  - Lustre: for Linux clusters (SUN Microsystems)
  - PVFS/PVFS2: Parallel Virtual File System for Linux clusters (Clemson/Argonne/Ohio State/others)
  - PanFS: Panasas ActiveScale File System for Linux clusters (Panasas, Inc.)
  - HP SFS: HP StorageWorks Scalable File Share. Lustre based parallel file system (Global File System for Linux) product from HP
  - BlueGene Parallel Storage System

7

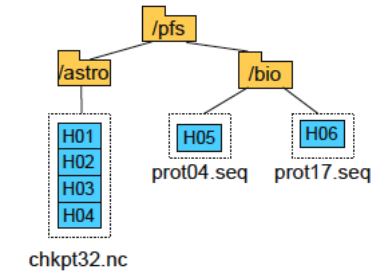
Aggregate Write Performance on BlueGene/P

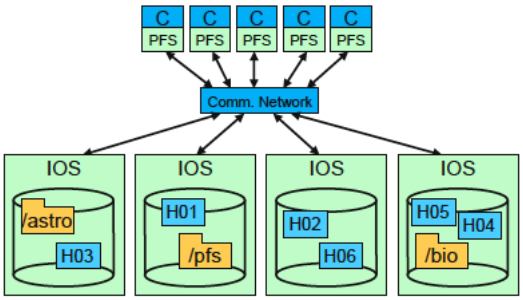


8

Parallel File Systems

- Building block for HPC I/O systems
  - Present storage as a single, logical storage unit
  - Stripe files across disks and nodes for performance
  - Tolerate failures (in conjunction with other HW/SW)
- User interface is POSIX file I/O





9

Features of Parallel File Systems

- Data Distribution
  - Metadata associated with files specifies a mapping of blocks of the file to PFS servers
- Locking
  - Locks are used to manage concurrent access
  - Enables caching - clients can have data on a local cache as long as they have a valid lock
- Fault Tolerance
  - RAID techniques hide disk failures
  - Redundant controllers and shared access to storage
  - Heartbeat software

10

How to get good I/O performance?

- Avoid I/O
- Use a parallel file system if available
  - But scientific applications have complex I/O patterns
  - Accessing the file system from a large number of processes has the potential to overwhelm the storage system
- How can we improve I/O performance further?

11

Parallel I/O

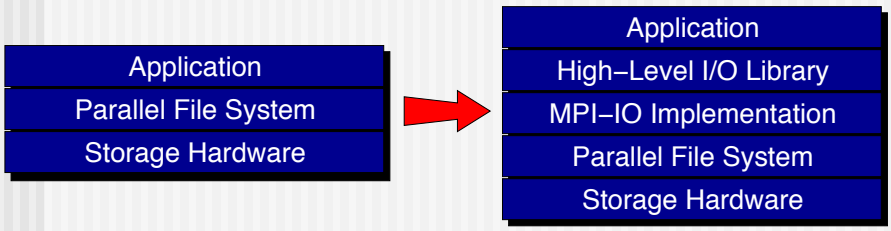
12

Application View of Data

- Applications have data models appropriate to domain
  - Multidimensional typed arrays, images composed of scan lines, variable length records
  - Headers, attributes on data
- Parallel file system API is an awful match
  - Bytes
  - Blocks or contiguous regions of files
  - Independent access
- Need more software!

13

I/O for Computational Science

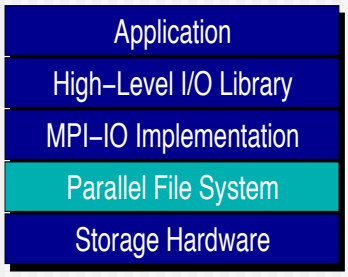


- Break up support into multiple layers:
  - High level I/O library maps app. abstractions to a structured, portable file format (e.g. HDF5, Parallel netCDF)
  - Middleware layer deals with organizing access by many processes (e.g. MPI-IO, UPC-IO)
  - Parallel file system maintains logical space, provides efficient access to data (e.g. PVFS, GPFS, Lustre)

14

Parallel File System

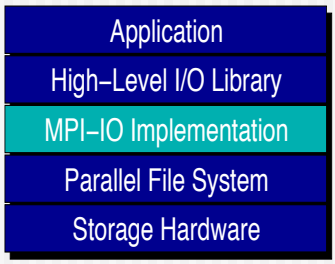
- Manage storage hardware
  - Present single view
  - Focus on concurrent, independent access
  - Knowledge of collective I/O usually very limited
- Publish an interface that middleware can use effectively
  - Rich I/O language
  - Relaxed but sufficient semantics



15

I/O Middleware

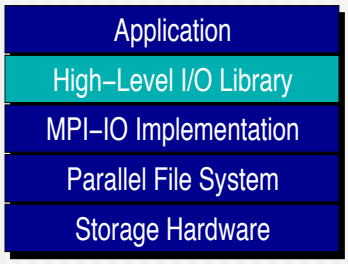
- Facilitate concurrent access by groups of processes
  - Collective I/O
  - Atomicity rules
- Expose a generic interface
  - Good building block for high-level libraries
- Match the underlying programming model (e.g. MPI)
- Efficiently map middleware operations into PFS ones
  - Leverage any rich PFS access constructs



16

High Level Libraries

- Provide an appropriate abstraction for domain
  - Multidimensional datasets
  - Typed variables
  - Attributes
- Self-describing, structured file format
- Map to middleware interface
  - Encourage collective I/O
- Provide optimizations that middleware cannot



17

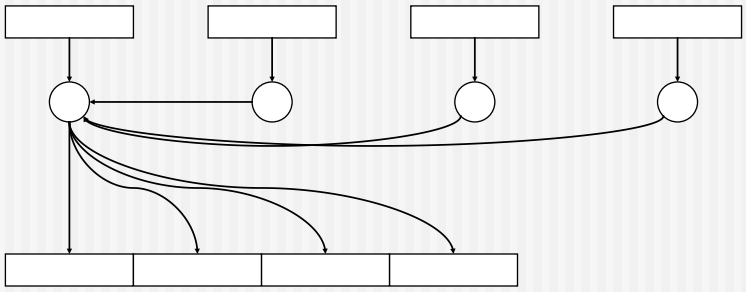
POSIX I/O Interface

- Standard I/O interface across many platforms
- Mechanism almost all serial applications use to perform I/O
- No way of describing collective access
- Warning: semantics differ between file systems!
  - NFS is the worst of these, supporting API but not semantics
  - Determining FS type is nontrivial
- POSIX interface is a useful, ubiquitous interface for building basic I/O tools
- No constructs useful for parallel I/O
- Should not be used in parallel applications if performance is desired

18

Common Ways of Doing I/O in Parallel Programs

- Sequential I/O:
  - All processes send data to process 0, and 0 writes it to the file



19

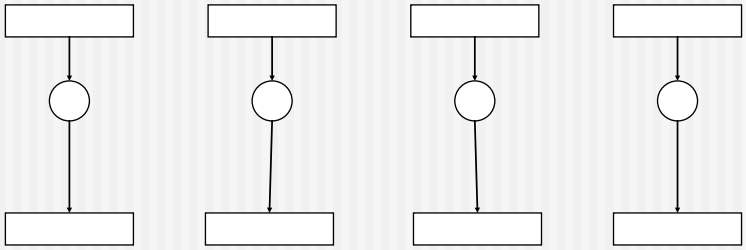
Pros and Cons of Sequential I/O

- Pros:
  - parallel machine may support I/O from only one process
    - (e.g., no common file system)
  - Some I/O libraries (e.g. HDF-4, NetCDF) not parallel
  - resulting single file is handy for **`ftp`**, **`mv`**
  - big blocks improve performance
  - short distance from original, serial code
- Cons:
  - lack of parallelism limits scalability, performance (single node bottleneck)

20

Another Way

- Each process writes to a separate file

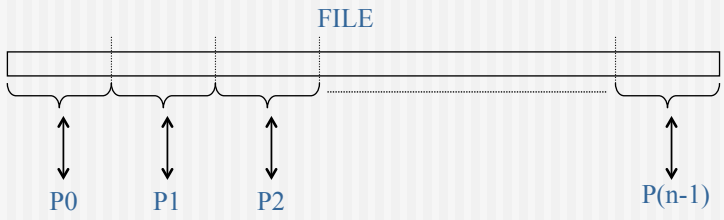


- Pros:
  - parallelism, high performance
- Cons:
  - lots of small files to manage
  - difficult to read back data from different number of processes

21

What is Parallel I/O?

- Multiple processes of a parallel program accessing data (reading or writing) from a *common* file



22

Why Parallel I/O?

- Non-parallel I/O is simple but
  - Poor performance (single process writes to one file) or
  - Awkward and not interoperable with other tools (each process writes a separate file)
- Parallel I/O
  - Provides high performance
  - Can provide a single file that can be used with other tools (such as visualization programs)

23

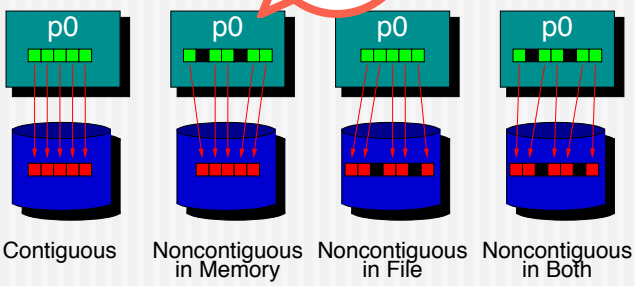
Considerations for Parallel I/O

- Contiguous and noncontiguous I/O
- Collective I/O
- I/O Aggregation

24

Noncontiguous

Complicated data structures

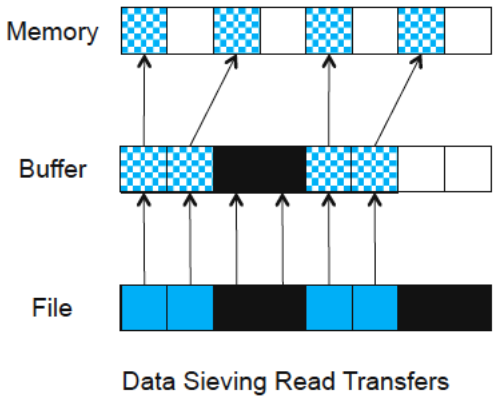


- Contiguous I/O moves data from a single block in memory into a single region of storage
- Noncontiguous I/O has three forms:
  - Noncontiguous in memory, noncontiguous in file, or noncontiguous in both
- Structured data leads naturally to noncontiguous I/O

25

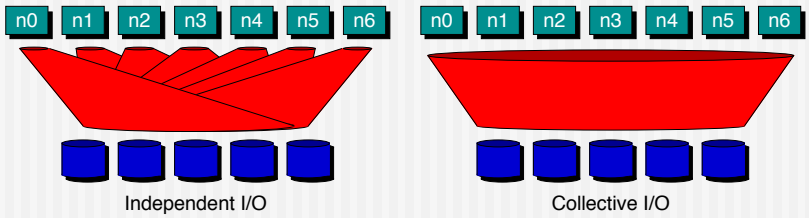
Data Sieving

- Useful for noncontiguous I/O
- Combine lots of small accesses into a single larger one
  - Reduce file system latency
  - Reduce nr of operations
- Generally very efficient but having a PFS supporting noncontiguous access preferable



26

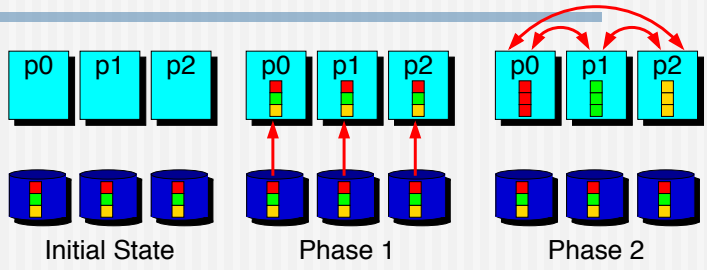
Collective I/O



- Many applications have phases of computation and I/O
- During I/O phases, all processes read/write data
  - We can say they are collectively accessing storage
- Collective I/O is coordinated access to storage by a group of processes
  - Collective I/O functions must be called by all processes participating in I/O
  - Allows I/O layers to know more about access as a whole
- Independent I/O is not organized in this way
- No apparent order or structure to accesses

27

Collective I/O: Two-Phase I/O



- Problems with independent, noncontiguous access
  - Lots of small accesses
  - Independent data sieving reads lots of extra data
- Idea: Reorganize access to match layout on disks
  - Single processes use data sieving to get data for many
  - Often reduces total I/O through sharing of common blocks
- Second ``phase'' moves data to final destinations

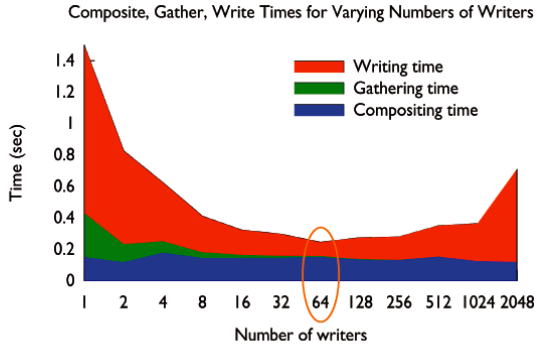
28

I/O Aggregation

- As the number of nodes on system grows, the access patterns seen by underlying file system appear increasingly chaotic. Reducing the apparent number of clients before hitting the file system can significantly improve performance

In this case we have 2K nodes rendering and compositing a 2048^2 image on the Blue Gene/P.

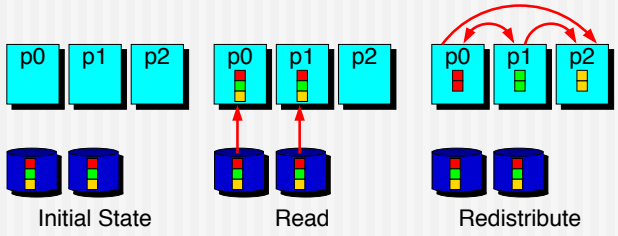
There are 64 compute nodes per I/O node. 64 writers corresponds to 2 writers per I/O node.



T. Peterka et al, "Assessing Improvements in the Parallel Volume Rendering Pipeline at Large Scale," SC08 Ultrascale Visualization Workshop, November 2008.

29

Aggregation



- Aggregation refers to the more general application of the concept of moving data through intermediate nodes
  - Different #s of nodes performing I/O
  - Could also be applied to independent I/O
- Can also be used for remote I/O, where aggregator processes are on an entirely different system

30

What is MPI-IO

- I/O interface specification for use in MPI applications
- Data model is a stream of bytes in a file
  - Same as POSIX and stdio
- Features
  - Noncontiguous I/O with MPI datatypes and file views
  - Collective I/O
  - Nonblocking I/O
  - Language bindings

31

Why MPI-IO

- Writing is like sending a message and reading is like receiving
- Any parallel I/O system will need a mechanism to
  - define collective operations (*MPI communicators*)
  - define noncontiguous data layout in memory and file (*MPI datatypes*)
  - Test completion of nonblocking operations (*MPI request objects*)
- I.e., lots of MPI-like machinery

32

General Guidelines for Achieving High I/O Performance

- Buy sufficient I/O hardware for the machine
- Use fast file systems, not NFS-mounted home directories
- Do not perform I/O from one process only
- Make large requests wherever possible
- For noncontiguous requests, use derived datatypes and a single collective I/O call

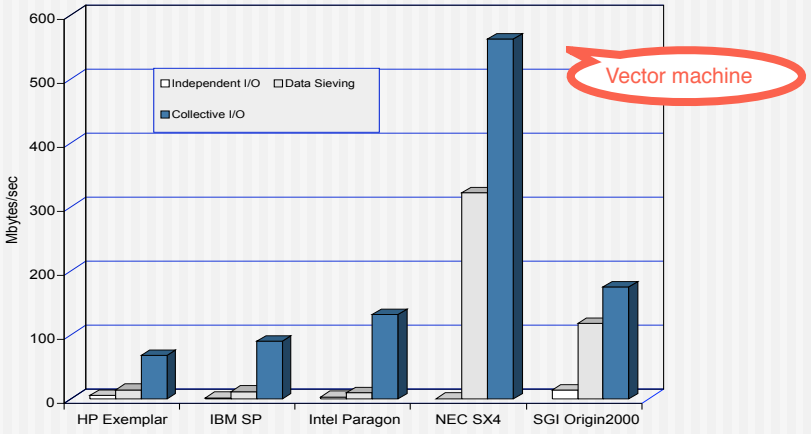
33

Achieving High I/O Performance

- Any application has a particular "I/O access pattern" based on its I/O needs
- The same access pattern can be presented to the I/O system in different ways depending on what I/O functions are used and how
- Given complete access information, an implementation can perform optimizations such as:
  - Data Sieving: Read large chunks and extract what is really needed
  - Collective I/O: Merge requests of different processes into larger requests
  - Improved prefetching and caching

34

Distributed Array Access: Read Bandwidth



Array size: 512 x 512 x 512

35

Summary and Outlook

- For high performance we need to parallelize I/O
- Parallel File Systems
- Parallel I/O
- Different Parallel Programming Models

36

Further Readings

- **Computer Organization and Design - The Hardware/Software Interface,** 4th Edition, David A. Patterson and John L. Hennessy
  - Chapter 6
- **Parallel I/O in Practice**, Tutorial, Rob Ross
- **Introduction to Parallel I/O and MPI-IO**, Tutorial, Rajeev Thakur

37