On Implementing MPI-IO Portably and with High Performance

Nov.07, 2002
Yoon, Il-Chul

Contents

Introduction

- Portable parallel programming is hampered by lack of
  - Standard, Portable API
- Most parallel file systems have nonportable Unix-like API
  - Unix API is inadequate for parallel I/O (lack features)
- New API for parallel I/O (MPI-IO) is defined as part of MPI-2 standard
- This paper mentions
  - ADIO: Portable implementation approach for MPI-IO
  - Advice to design file system fit for MPI-IO

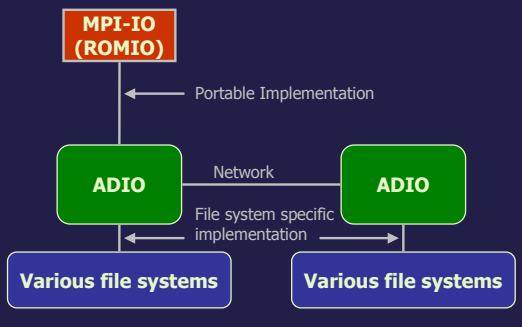
Motivation

- Limitations of Previous MPI-IO implementations on top of Unix I/O interface
  - Basic Unix I/O functions not sufficient for supporting MPI-IO
    - Basically, blocking functions
    - File size limitation (2 Gigabytes)
    - Even if MPI-IO like libraries were implemented, it's not portable
  - Though basic Unix I/O functions are portable, it doesn't show same performance on different file systems
  - If use NFS, it does not guarantee competing write to a common file
- Limitations of Previous MPI-IO implementations on top of POSIX I/O interface
  - It is not widely implemented and not correctly implemented
  - Implementation is different among file systems
  - Some MPI-IO features are missed

New Approach

- Combining large portion of portable code and small portion of separately-optimized code
- ADIO: Abstract-Device Interface for I/O
  - Mechanism designed for implementing portable parallel I/O API
  - Consists of basic functions for parallel I/O
    - Implemented on various file systems
    - Machine dependent and independent parts are separated
    - For each file system, ADIO may be compiled differently
  - Support remote IO through ADIO servers
- ROMIO is implemented on top of ADIO
  - MPI-1 compatible
  - Work with MPICH, HP MPI, and SGI MPI

New Approach



Parallel File Systems design features for MPI-IO

- High-performance parallel file access
  - Concurrent request must not be serialized if possible (performance degrade)
- Data-Consistency Semantics
  - Means clear definition and implementation of concurrent access from multiple processes
  - Sequentially consistent if
    - Operations behave as if they were performed sequentially
    - Each access appears atomic
- Atomicity Semantics
  - Define the result of access to overlapped region of a file from multiple processes
  - Fact: most application do NOT perform concurrent overlapped accesses
  - It's better to support atomic and non-atomic mode for higher performance

Parallel File Systems design features for MPI-IO

- MPI-IO File consistency (sequentially consistent)
  - Case 1: fh1 $\in$ FH1 (set of file handles from collective open of a file)
    - If atomic, then consistent
    - If non-atomic, but if operations are not concurrent and not conflicting, then consistent
  - Case 2: fh1a $\in$ FH1 and fh1b $\in$ FH1
    - If op1 using fh1a and op2 using fh1b are not conflicting, then consistent
    - If conflict..., if atomic mode, consistent.
  - Case 3: fh1 $\in$ FH1 and fh2 $\in$ FH2
    - If op1 using fh1 and op2 using fh2 are not concurrent, then consistent

Parallel File Systems design features for MPI-IO

- Interface Supporting Noncontiguous Accesses
  - It's better to support retrieving regularly scattered data for processes
  - Precondition: the file offset for the data must be monotonically non-decreasing
- Byte-Range Locking
  - Sometimes, locking for part of file is required
  - E.g.) atomicity implementation
- Collective I/O
  - Called by all processes in a communicator
  - Basically, blocking send/recv
  - Used for performance improvement (reduce I/O calls)

Parallel File Systems design features for MPI-IO

- No shared file pointer
  - File system need not know MPI-related things.
  - Can be implemented in MPI lib.
- Nonblocking (Asynchronous) I/O
  - Not mandatory. Can be emulated by blocking I/O threads
- Additional features
  - Support of large files over 2GB
  - Control over File Striping
    - Support to change default parameters for access
  - Variable caching / Prefetching policies
    - Support to change policy
  - File Preallocation
    - Support to pre-allocate disk space for a file.

Conclusion

- POSIX file system semantics and interface is sufficient for implementing MPI-IO correctly, but additional features would be help MPI-IO implementation achieve higher performance
- ROMIO demonstrate that it is possible to implement MPI-IO portably on multiple machines and file systems and also achieving high performance
- ADIO framework is the key component and enables to perform file-system-specific optimizations within a largely portable implementation.