The Galley Parallel File System

Nils Nieuwejaar
Dartmouth College

Joint work with David Kotz

Overview

- Background
- File Structure
- System Structure
- Interface
- Case Study
- Conclusion

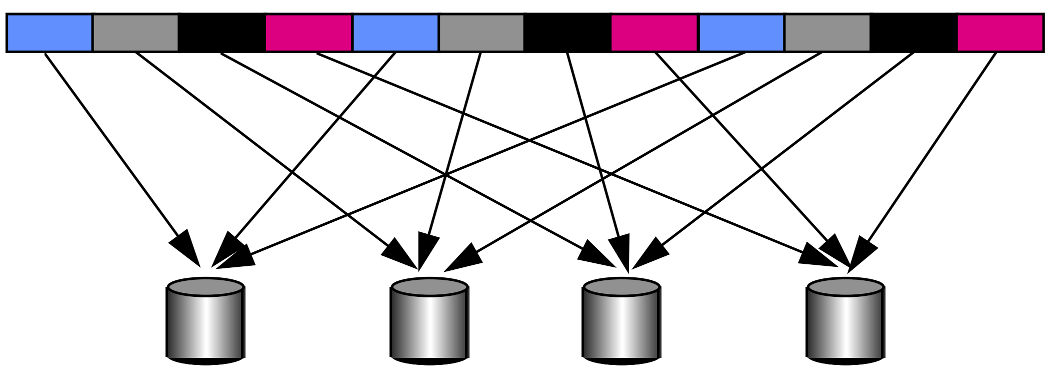
Background

- Most parallel file systems provide:
  - Linear file model
  - Unix-like interface
- Optimized for:
  - Large files
  - Large requests
  - Sequential access

Linear File Model

- Typically *stripe* data across all the disks in the system.
- Good performance for large requests
- Not good for small requests

Partition of a linear file into blocks

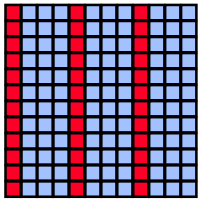


Workload Characterization

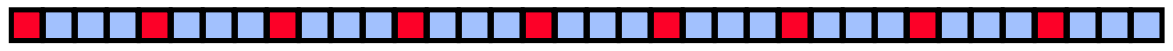
- Traced two production systems
  - iPSC/860 at NASA Ames
    - » 128 Compute nodes
    - » Primarily CFD applications
    - » Dozens of users
    - » Control-parallel
  - CM-5 at NCSA
    - » 512 Compute nodes
    - » Variety of applications
    - » Hundreds of users
    - » Data-parallel

Workload Characterization

- Most requests were small
  - Most fewer than 300 bytes
- Requests were frequently non-contiguous
- Request sizes and intervals were regular
- Strided patterns were common



2D Matrix



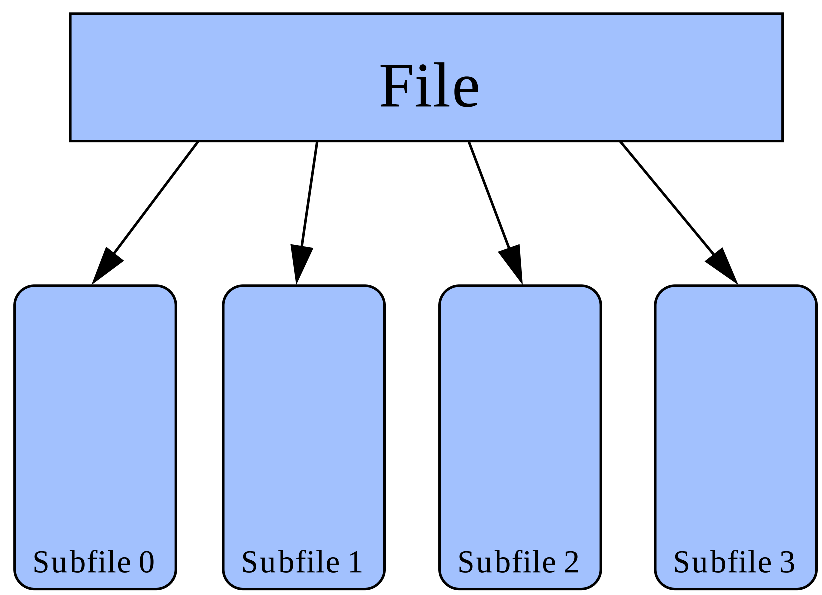
Mapped to linear file

Design Goals

- Efficiently handle many access sizes and patterns
- Allow applications to explicitly control parallelism
- Allow easy and efficient implementation of libraries
- Scalable
- Minimize memory and performance overhead

Subfiles

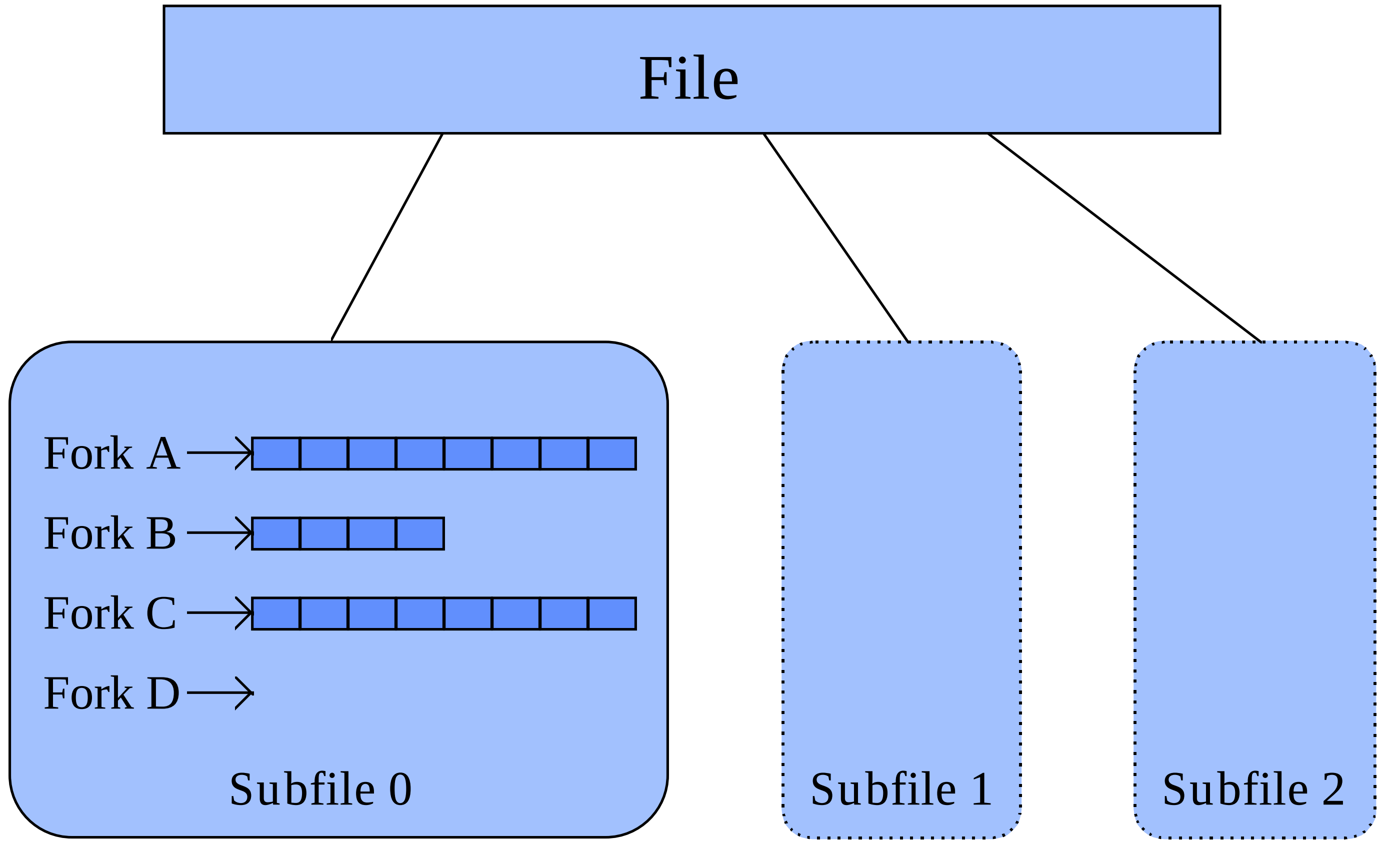
- Each file contains one subfile per disk
- Applications can explicitly access subfiles
- Allows control over:
  - declustering
  - parallelism



Forks

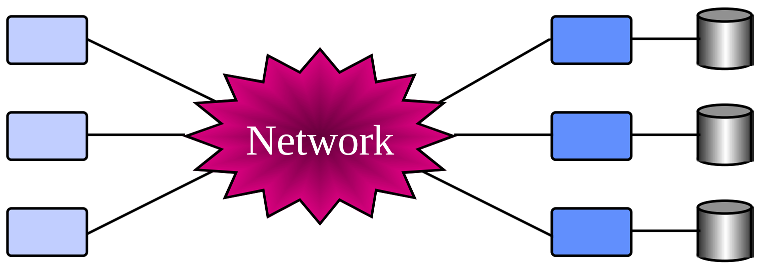
- Each subfile contains one or more forks
- Each fork is a named, linear stream of bytes
- Uses:
  - Library-defined metadata
    - » e.g., indexing information
  - Structuring data
    - » e.g., temperature in one fork, pressure in another
  - Store code for accessing data
    - » e.g., Python, Java, Tcl, traditional object code

File Structure



System Structure

- Compute Processors
  - User applications
  - Galley run-time library
- I/O Processors
  - Control disks
  - Run Galley's system code

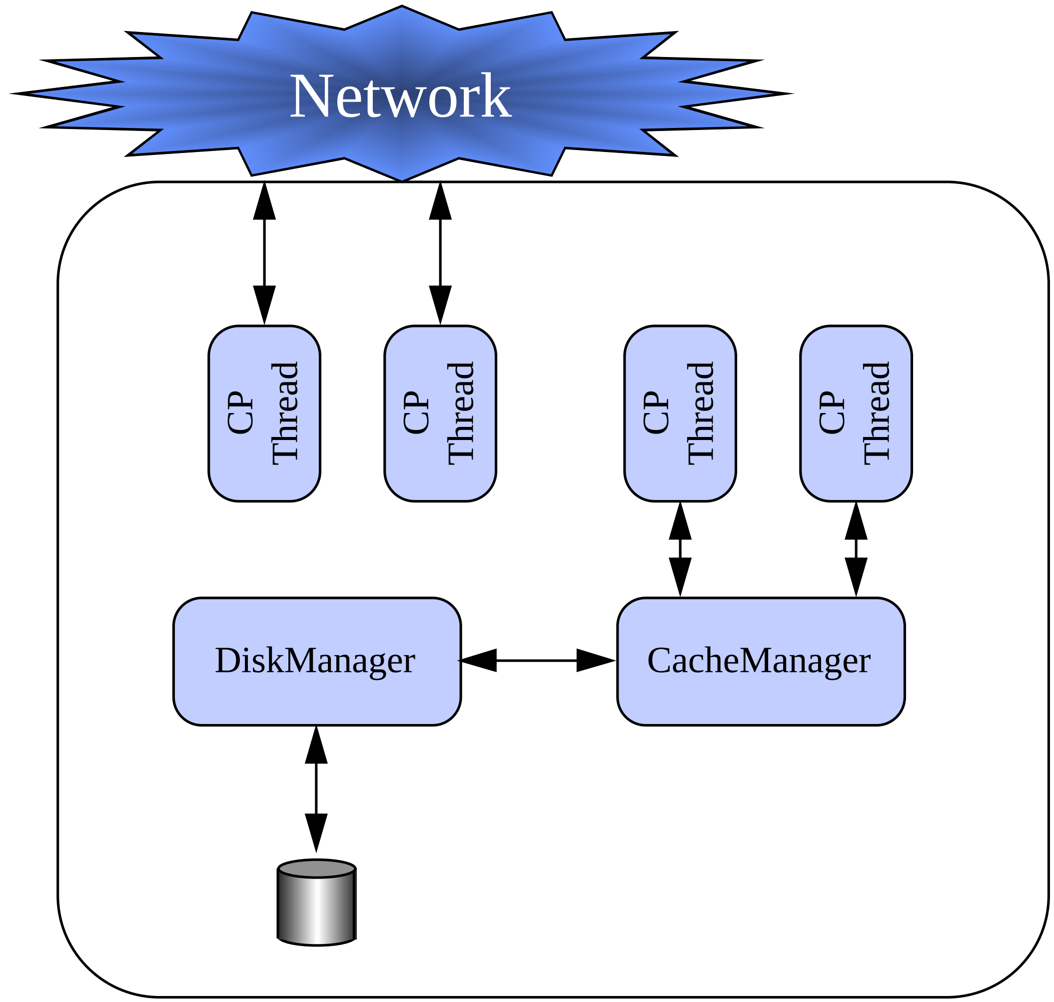


Compute Processors

- Galley run-time library
  - Package application requests and send to IOPs
  - Handle transfer of data between IOPs and application's memory
- Client applications can use
  - C or C++
  - Any message passing library
  - Future:
    - » Fortran
    - » ViC*

I/O Processors

- Functional units
  - CP Threads
  - CacheManager
  - DiskManager
- Implemented as multiple threads



CP Threads

- Each CP has a dedicated thread
- Given a request, generates list of all blocks needed to satisfy request
- Passes whole block list on to the CM
- Waits on a buffer ‘ready queue’
- CP Thread moves data between buffer and CP

CacheManager

- Maintains buffer cache
- Uses LRU replacement policy
  - Future: allow CP-specified policies
- Service requests from CP threads
  - One block at a time
  - In round-robin order
- If block isn't in cache, issue request to DiskManager

DiskManager

- Controls layout of data on disk
  - Logically partitions disk into 32KB blocks
  - Future: multiple disks
- Uses Unix files, raw devices, or simulated disks
- Uses Cyclic-Scan disk scheduling
- When idle, writes back dirty blocks from buffer cache

File Operations

- `gfs_create_file`
  - Hash name to find metadata
  - Reserve name, create subfiles, commit name
- `gfs_open_file`
  - Cache subfile headers in CP memory
- `gfs_close_file`
- `gfs_delete_file`

Fork Operations

- `gfs_create_fork`
  - Creates fork in one subfile
- `gfs_all_create`
  - Creates fork in each subfile
- `gfs_open_fork / gfs_all_open`
- `gfs_close_fork`
- `gfs_delete_fork`

Data Transfer Operations

- Traditional Unix-like read/write interface
- Galley allows applications to make several kinds of batched requests
  - Strided
  - Nested-strided
  - Nested-batched
  - List I/O

Case Study: FITS

- Flexible Image Transport System
  - Standard format for astronomical data
  - ASCII header
  - Binary data: Series of records
  - Each record
    - Has a key with one or more fields
    - Has one or more data elements
    - Is the same size, and has the same structure

FITS at NRAO

- Each key contains six fields
  - Total: 24 bytes
- Each data element contains
  - FP triples for each of 31 frequencies
  - Total: 744 bytes
- Data sets are sparse and multidimensional
- Queries typically involve subranges or slices in one or more dimensions

FITS on Galley

- Sorted records by time
- Distributed records
  - Cyclically across subfiles
  - In 1024-record blocks
- Three forks: header, keys, and data
  - Allows us to scan keys cheaply, identifying relevant records
- Used `gfs_listio()` to efficiently extract relevent records from the data fork

Summary

- Based on analyses of production workloads we have designed a new parallel file system
  - Designed to meet needs of parallel scientific applications
  - Designed to ease library implementation
  - Exposes the full parallelism of the system to the application
- Showed how Galley's features were useful in practice

Future Work

- Porting benchmarks, applications, libraries, and compilers to Galley.
- Examine how to support multi-application workloads fairly and efficiently.
- Long-term project: examine possibility of moving application code to I/O nodes.

More information

- IOPADS
  - Discuss data transfer interface and its impact on performance
- WWW:
  - `http://www.cs.dartmouth.edu/~nils/galley`