Next Generation Parallel Virtual File System

Walter B. Ligon III

October 9, 2001

Brief history

- PVFS v0α
  - born: 1994
  - purpose: provide parallel I/O access pattern data
  - claim to fame: built on PVM
- PVFS v0β
  - born: 1995
  - purpose: research tool for parallel file systems
  - claim to fame: designed for Beowulf

Then it was released

- PVFS v1
  - born: 1996
  - purpose: still a research tool, but supported
  - claim to fame: easy to use and actually works
- PVFS v2
  - born: 200X

NextGen PVFS

Talk outline

- Why re-write PVFS?
- The new architecture
  - Interfaces
  - BMI, flows, jobs, etc.
  - Threads and stuff
- Loadable distributions
  - Data redundancy
  - Adding I/O servers
- MPI_Datatype based requests
- Distributed metadata management

Why re-write PVFS?

- PVFS was written to be a research tool
  - Interfaces designed for user-level
- PVFS lacks features we would like to have
  - Powerful data distributions
  - Data redundancy
  - Use of different/multiple network interfaces
- PVFS does not permit control of ...
  - Network or file I/O scheduling

Goals for PVFS v2

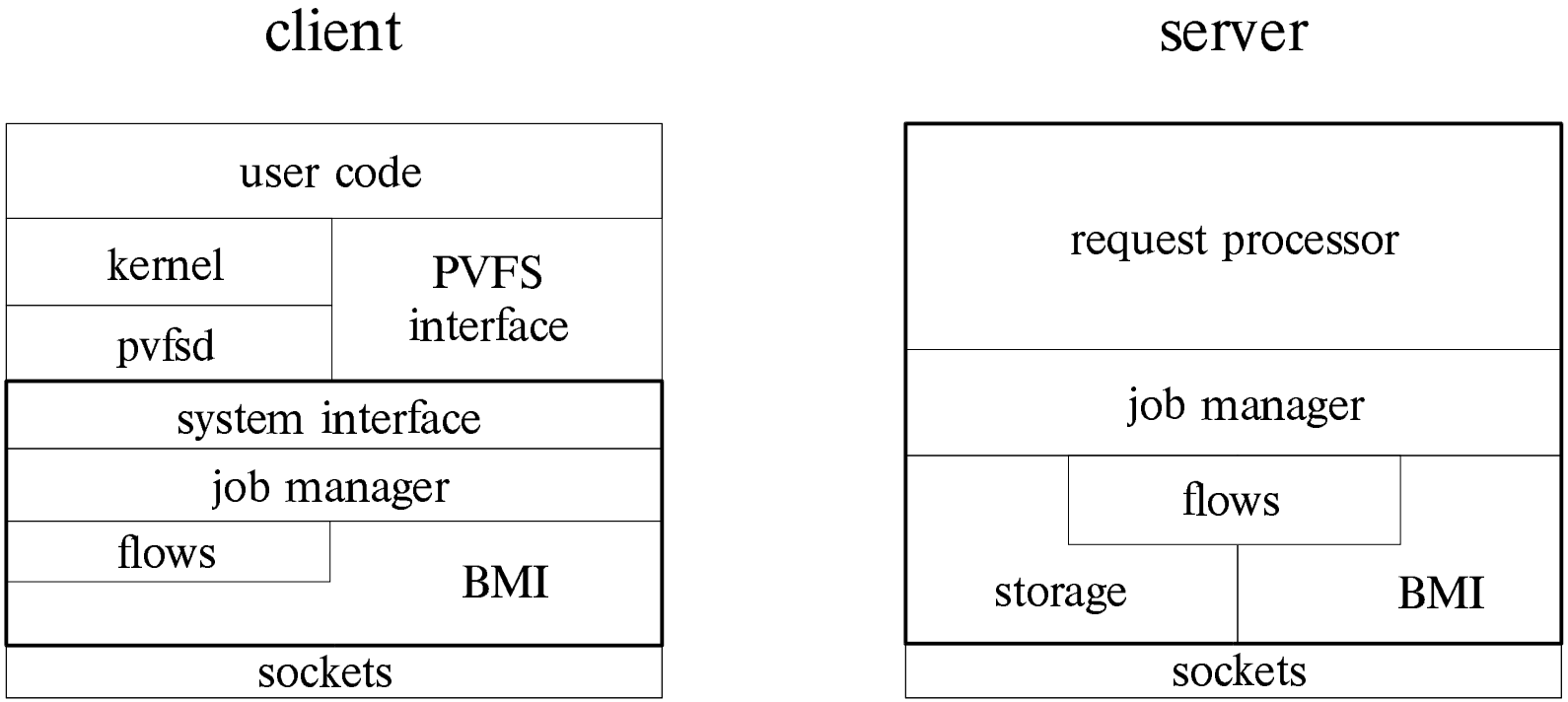
- Maximum performance for large scale parallel I/O
- Flexibility in
  - network transfer
  - Storage media
  - requests
  - data distribution
  - data redundancy
- Support development of experimental features
- Easier to maintain and support

The new architecture

PARL
Parallel Architecture Research Laboratory

- Try a little software engineering!
  - Design first, code second
  - Involve “the community” from the start
  - Consider important abstractions
  - Document everything
- Design for “production” environment
  - Integration with the Linux kernel
  - Completeness
  - Robustness

PVFS v2 architecture



Interfaces

- PVFS user interface
- Linux kernel interface - pvfsd
- System interface
- Request protocol

BMI, flows, jobs, etc.

- BMI - Buffered Method Interface
- Storage Interface
- Flows
- Jobs

BMI

- Abstraction, Modularity, Efficiency
- Simple application interface
- Designed for client/server operation
- Fully non-blocking (scalable)
- Overlap of network I/O and system tasks
- Support for user level and kernel level network APIs
- Supports multiple protocols without recompiling
- Multiple simultaneous networks (heterogeneous)
- Supports buffer optimizations when available

BMI interface

- API methods

```
BMI_initialize, BMI_finalize
BMI_post_send, BMI_post_recv
BMI_unpost
BMI_addr_lookup
BMI_test, BMI_testsome, BMI_testglobal
BMI_testunexpected
BMI_memalloc, BMI_memfree
BMI_set_info, BMI_get_info
```

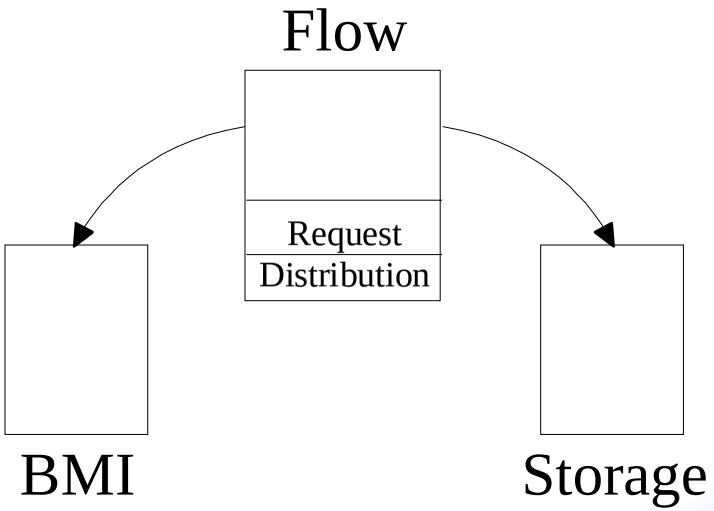
- All methods are non-blocking
- May require methods be called for progress

Storage interface

- Byte-stream storage
  - Contiguous and non-contiguous access
- Key/value pair storage
  - Metadata
- Non-blocking semantics
- Compatibility with flows
- Consistency semantics

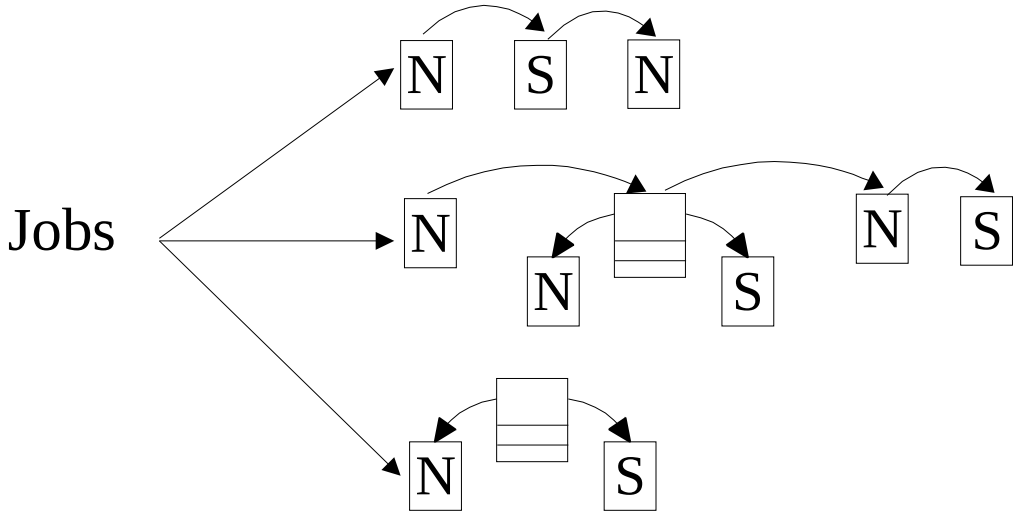
Flows

- Represents a transfer between storage and network
  - Implemented on top of the storage and network abstractions
  - Processes non-contiguous request structures
  - Utilizes data distribution methods
- Presents an interface similar to BMI and storage



Jobs

- Provides asynchronous management of multiple BMI, storage, and flow transactions
- Access point for scheduling decisions
- Allows sequences of operations, with error processing



Threads and stuff

- Jobs, flows, storage requests, and BMI requests can be implemented with threads
- When to implement with threads?
  - Depends on their use (client or server)
  - Depends on efficiency
- Expect to use at least one thread in the server
  - Possibly none in the client
  - We'll be experimenting with this

Loadable distributions

- Flexible physical distribution patterns
  - Fully extensible
  - Efficient implementation
  - Convenient interface
- Based on original PVFS distribution methods
  - Dynamically loadable C functions
    - Convert logical offset to physical offset, IOD number
    - Find next logical offset maped to current IOD
    - Find end of current physical extent
  - Programmer specified tags
  - Parameterizable

Data replication

- Support mirroring
- Extension of distribution methods
- Function specifies where data should be mirrored
- Build on mirroring work going on with PVFS v1

MPI_Datatype based requests

- Flexible request mechanism
  - General mechanism compatible with listio
  - Compact representation for regular patterns
    - strided
    - nested-strided
  - Readily compatible with MPI-IO and ROMIO
- MPI_Datatypes
  - provide needed generality
  - provide desired compact representation
  - Obvious fit to MPI-IO
- Implementation similar to existing MPI packages

Distributed metadata

- Eliminate potential bottleneck
- Merge metadata “mgr” server and IOD
  - All servers process data and metadata requests
- Metadata server selected with hash function

Conclusion

- PVFS redesigned
  - Improved code
    - more robust
    - easier to maintain
    - easier to extend
  - Better features
    - more powerful requests
    - more flexible distributions
    - redundancy
  - Support for future experimentation and deveopment
- Commitment to performance