An Evolutionary Path to Object Storage Access

David Goodell[+], **Seong Jo (Shawn) Kim***, Robert Latham[+],
Mahmut Kandemir*, and Robert Ross[+]

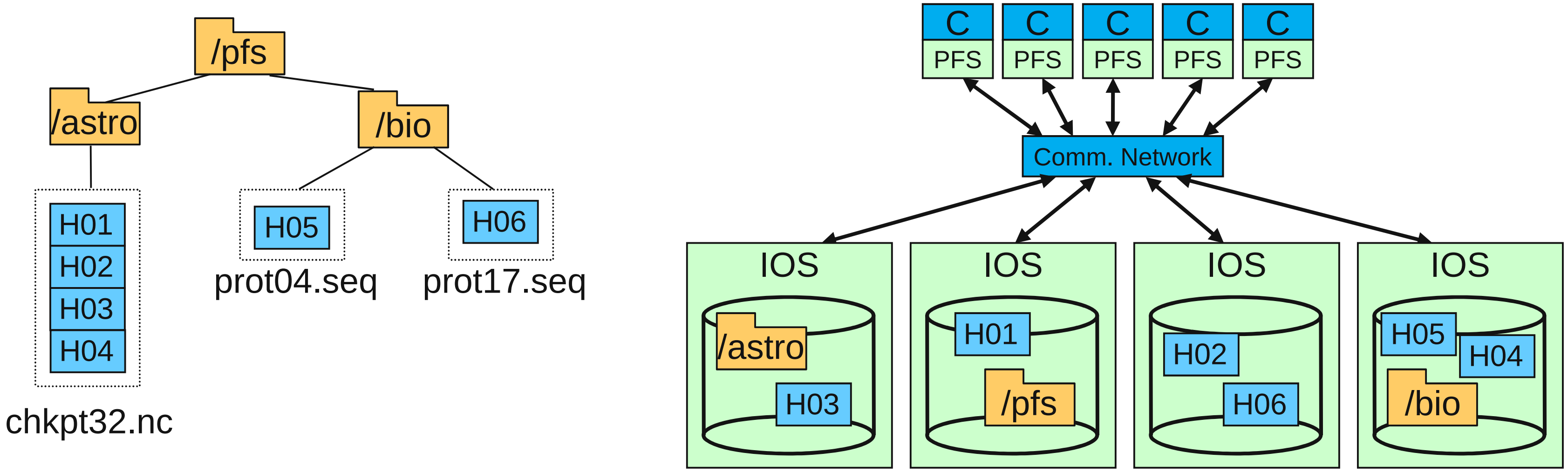
*Pennsylvania State University
[+]Argonne National Laboratory

SC12
Salt Lake City, Utah

Outline

- Introduction
  - Background of parallel file systems
  - Overview of object storage model
  - Goal
- Our approach
  - Supporting object access in PVFS
  - Using objects in HPC I/O libraries: PLFS and PnetCDF
- Conclusions & future work

Parallel File Systems: What do they do?



An example parallel file system, with large astrophysics checkpoints distributed across multiple I/O servers (IOS) while small bioinformatics files are each stored on a single IOS.

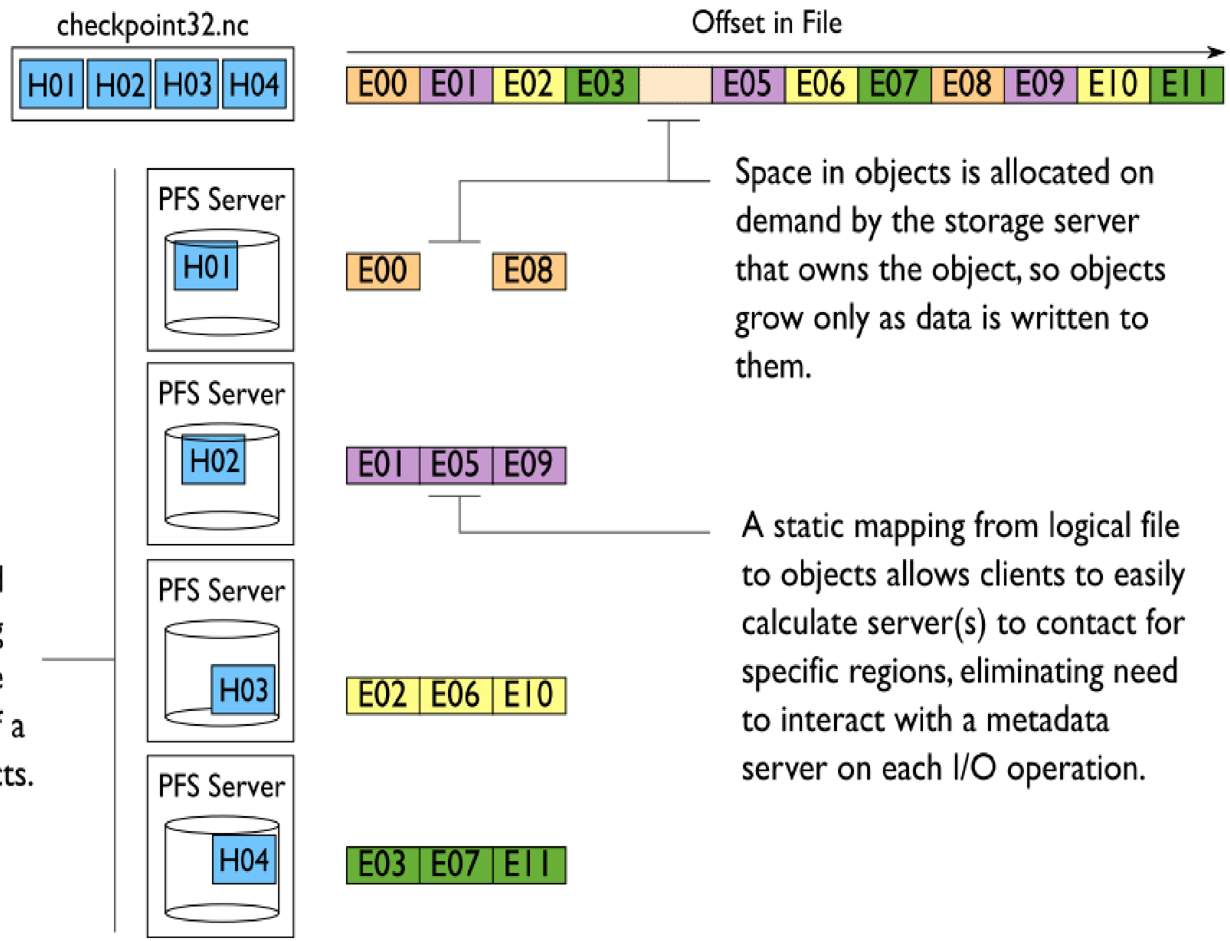
- Manage a name space of directories and user data
- Distribute data across many servers (e.g., by managing large collection of objects)
- Provide a POSIX file “veneer” atop distributed data (e.g., by mapping a POSIX file abstraction onto a set of objects)

Objects in a POSIX Namespace

Logically a file is an extendable sequence of bytes that can be referenced by offset into the sequence.

Metadata associated with the file specifies a mapping of this sequence of bytes into a set of objects on PFS servers.

Extents in the byte sequence are mapped into objects on PFS servers. This mapping is usually determined at file creation time and is often a round-robin distribution of a fixed extent size over the allocated objects.



Space in objects is allocated on demand by the storage server that owns the object, so objects grow only as data is written to them.

A static mapping from logical file to objects allows clients to easily calculate server(s) to contact for specific regions, eliminating need to interact with a metadata server on each I/O operation.

Parallel File Systems: Successes

- Current parallel file system designs scale to tens or a few hundred servers (Big!)
- Individual servers can move data very effectively, given the right patterns (Fast!)
- Name space is not loved, but mostly ok unless we are creating files for every process.

Parallel File Systems: What's the Problem?

- The POSIX file model provides a single byte stream into which data must be stored
- HPC applications create complex output that are naturally multi-stream
  - Structured datasets (e.g., HDF5, netCDF)
  - Log-based datasets (e.g., PLFS, ADIOS BP)
- Dilemma
  - Do I create lots of files to hold many streams?
    - Stresses the metadata subsystem!
  - Do I map many streams into a single file?
    - Now I need to understand distribution and locking!

The Captain Kirk Solution*

- Expose individual object byte streams for use by I/O libraries (e.g., Parallel netCDF, PLFS)
  - Library becomes responsible for mapping its data structures into these objects.
- Keep the rest!
  - Have directories, organize objects under file names
  - Keep permission, etc.
- When software puts you in a no-win situation, re-code it!

* See http://en.wikipedia.org/wiki/Kobayashi_Maru

Goal

- Propose an alternative interface for applications and libraries that provides direct access to underlying storage objects.
  - Avoiding lock contention w/o creating many separate files
  - Complex data models are easily organized into the multiple object data stream, simplifying the storage of variable-length data
  - Coexist with POSIX files
- Advantages:
  - Separate the creation of multiple data streams from the creation of names in the name space
  - Allow the multiple data streams present in the individual objects to be directly used for organizational purposes.

Our Approach

- Our approach: to expose a set of objects (an ordered list) that is associated with a single file name (a *container*)
- Benefit: to move responsibility of mapping application data structures into the objects from the file system to the libraries or application.
- Assumption:
  - The underlying storage performs consistency management (i.e., locking), if any, on a per-object basis
  - Creating many objects under a single file name is faster than creating multiple files in the name space

Supporting Object Access in PVFS

- We modified PVFS2 (v2.8.2).
- Only client-side modifications were required to facilitate the new model.

Supporting Object Access in PVFS: New API

- It’s a quite small interface (7 routines).
  - `read_contig` and `write_contig` are there only as convenient special cases of `readx` and `writex`, so actually it's 5 routines.
- The interface is stateless.
- The interface provides a "list i/o" interface for more complex data descriptions in both memory and file.

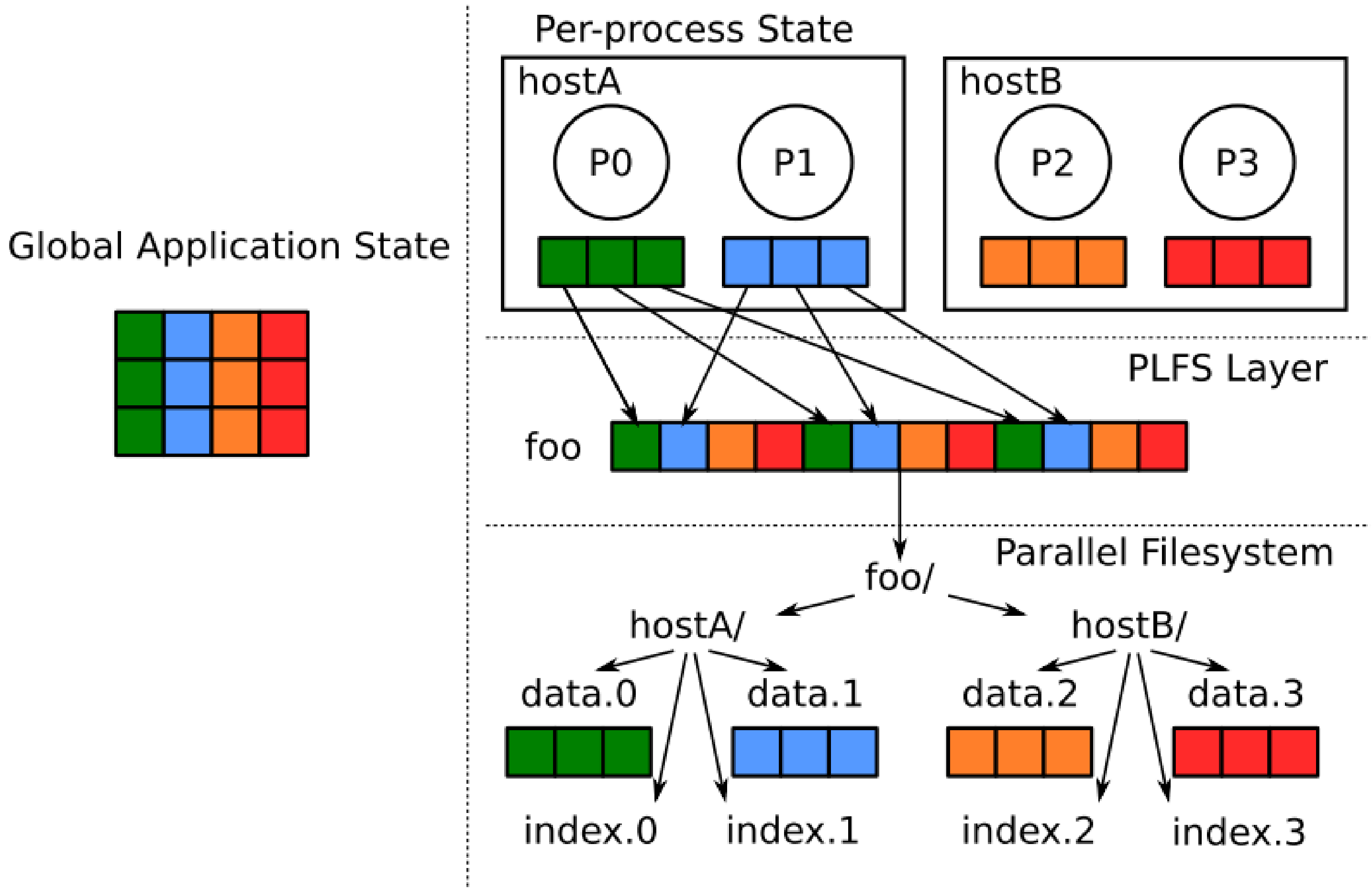
Supporting Object Access in PVFS: PVFS2 Client Implementation Details

- PVFS2 object model
  - Decompose a logical POSIX file into a single *metafile* and multiple *datafiles*
  - A distribution function maps logical file extents into extents in datafiles, identified by a `PVFS_object_ref`.
- Our prototype reuses these existing concepts.
- Two new state machines were added to the PVFS2 prototype.
  - Object collection creation
  - Read/write operations to a single object

Using Objects in HPC I/O Libraries: Parallel Log Structured File System (PLFS)

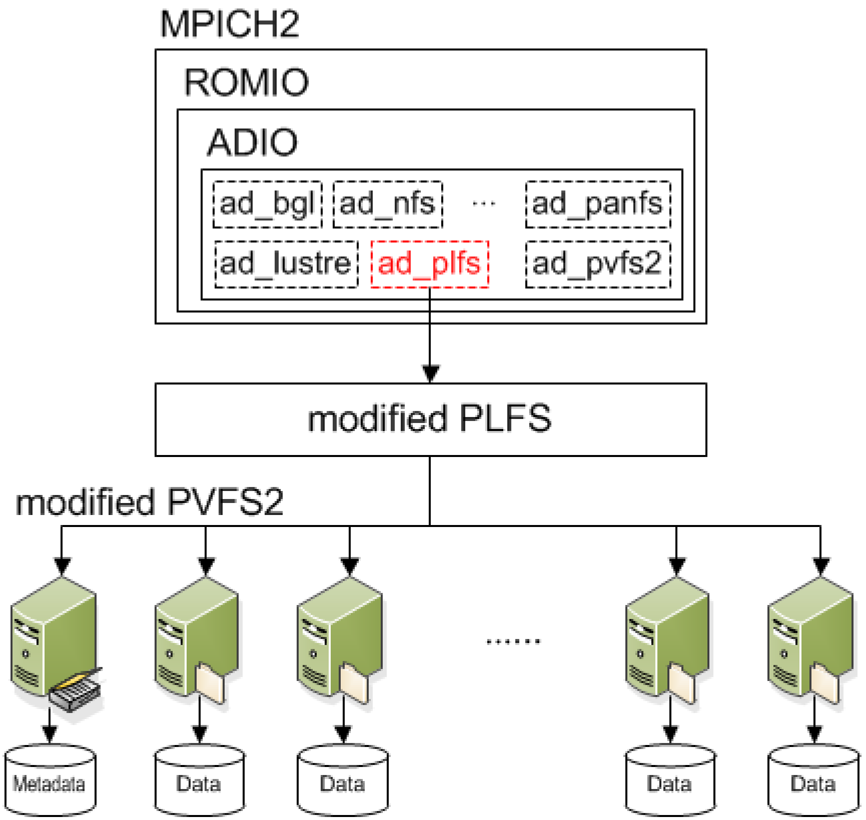
- PLFS is designed to improve write bandwidth for checkpoint.
- PLFS is implemented as a user-space file system, exposed through FUSE or MPI-IO.
- After writing to a data file, the metadata information is appended to the associated index file.
- By remapping writes to a non-shared data files, PLFS converts an N-1 strided access pattern into an N-N.

Using Objects in HPC I/O Libraries: Parallel Log Structured File System (PLFS) (cont'd)

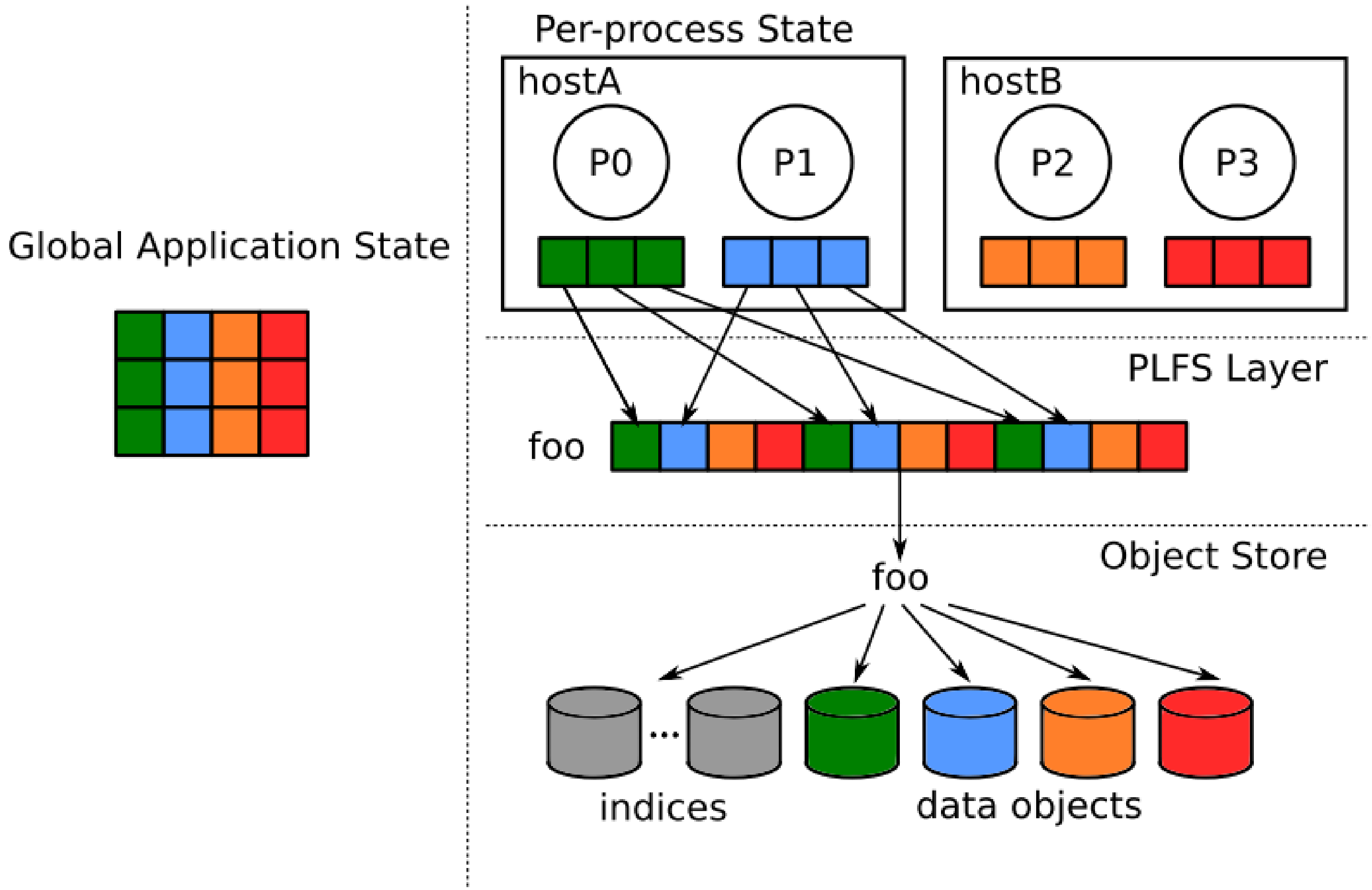


Using Objects in HPC I/O Libraries: PLFS over Object Storage Model

- For our prototype, we plugged the `ad_plfs` interface into ROMIO ADIO layer of MPICH2-1.5, porting PLFS.
- Application program directly make MPI-IO calls to reach PLFS.
- PLFS is modified to support the new API for object-based access.



Using Objects in HPC I/O Libraries: PLFS over Object Storage Model (cont'd)

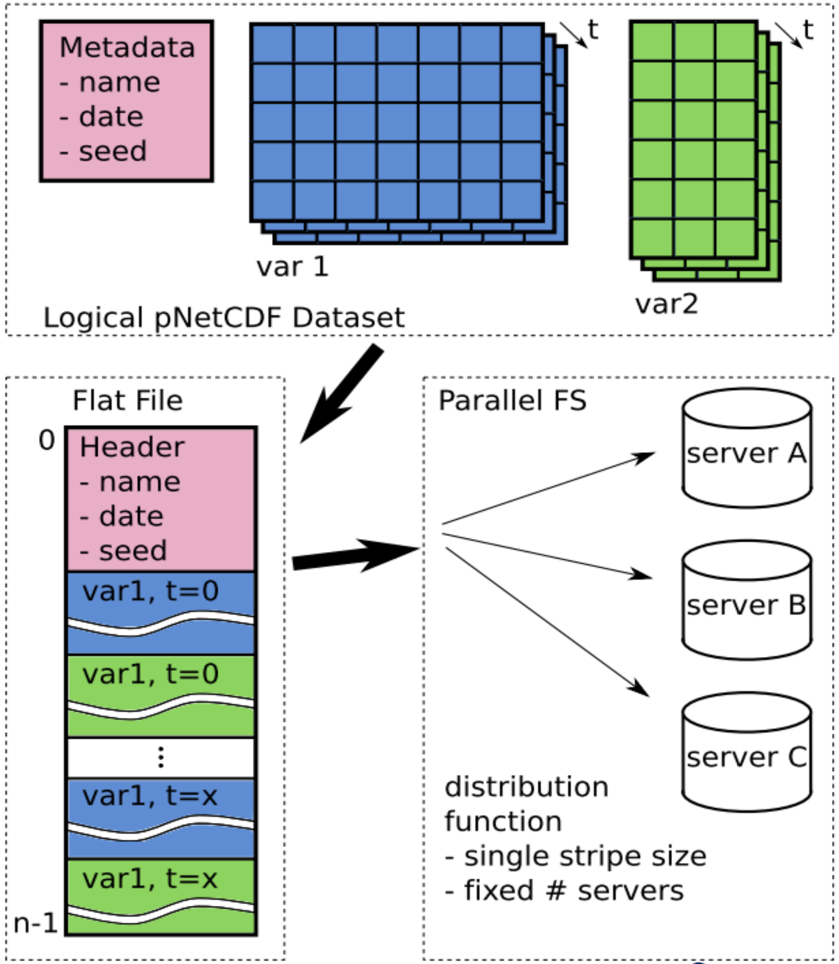


Using Object in HPC I/O Libraries: Parallel netCDF

- PnetCDF provides an interface for parallel reading and writing of data in the netCDF file format.
- Array can be of fixed dimensions (non-record arrays) or have one dimension in which they may grow (record arrays).
- Tiles of these record arrays are interleaved in the file so that space may be allocated as the record arrays grow.

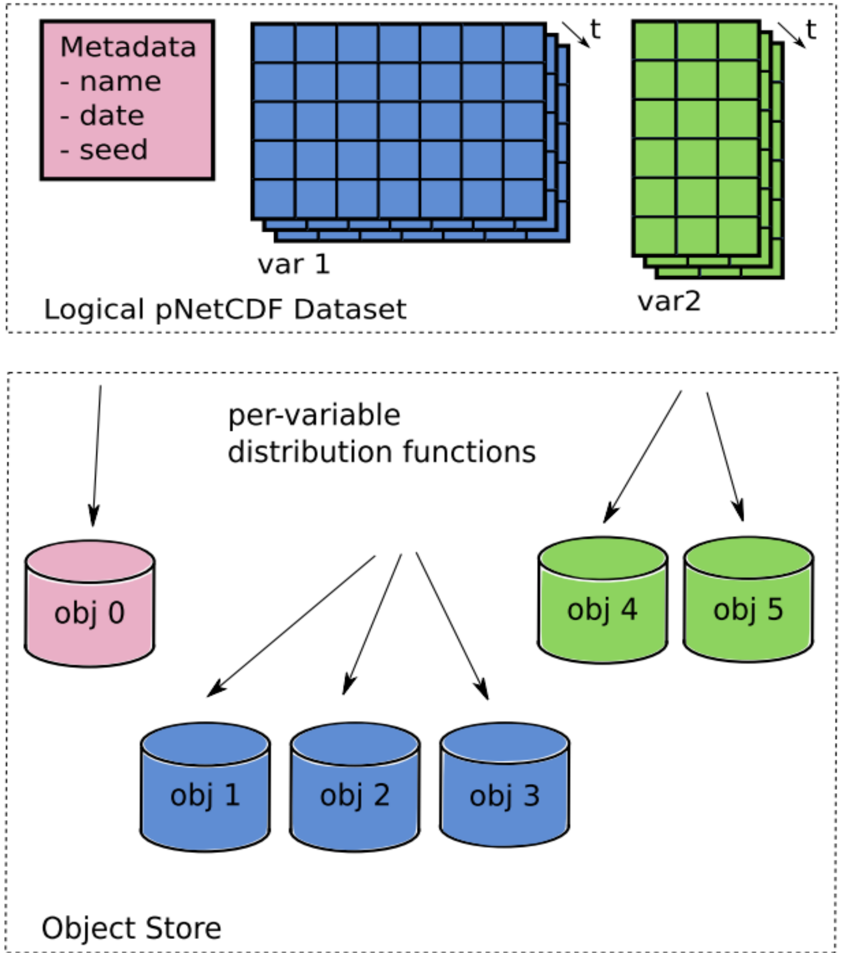
Using Objects in HPC I/O Libraries: Parallel netCDF

- Mapping a PnetCDF dataset into a POSIX file
  - Header data & non-record arrays come in the POSIX file's byte stream.
  - Two record arrays are interleaved.
  - The flat file is distributed to servers w/o regard to compatibility btw FS distribution params and the layout of netCDF arrays.
- Performance drawbacks: irregularly aligned access, misaligned data, and record variable storage



Using Objects in HPC I/O Libraries: PnetCDF over Object Storage Model

- PnetCDF prototype maps the same dataset into the set of objects.
  - The header and each array are mapped to the set of one or more objects.
- Benefits:
  - simplify the implementation in reading/writing from/to variables for non-contiguous access.
  - PnetCDF controls the data distribution on a per-variable basis.
  - Avoid misaligned data access



Using Objects in HPC I/O Libraries: PnetCDF over Object Storage Model (cont'd)

- In our prototype, each PnetCDF variables has its own distribution function.
- Data is striped byte-wise in a row-major fashion.
- More complex distribution could be easily implemented.

Other Considerations

- File size
  - Our approach moves the role of the distribution function into application or library space.
  - PFS returns the total size of data stored in constituent objects, which may not deal with "sparse files" accurately.
- Access control and extended attributes
  - These two pieces of POSIX functionality should be unchanged.
- Copying collections
  - A collection could be copied by creating a new set of objects of the same size as the collection of objects in the source, and
  - Copying the contents of each object into the corresponding object in the new list.

Conclusions and Future Work

- We've presented a new abstraction for storage that enables higher performance for HPC applications while coexisting with the legacy POSIX name space.
- Our containers of object models maps more closely to both application/library needs as well as modern storage systems.
- Moving the responsibility of mapping application data structures into storage objects from the storage system
  - Applications control performance
  - Simpler implementation
- Is there value in mapping to thousands of objects? *vs.* an exploration of the design space for the storage system itself.
- I/O forwarding stacks

Questions?