CSC 7700: Scientific Computing
Module B: Networks and Data
Lecture 4: Distributed Data Management

Dr. Andrei Hutanu

Distributed Data Management

- Distributed storage/file systems
- Remote I/O
- Data staging and replication
- Data, networks and scheduling

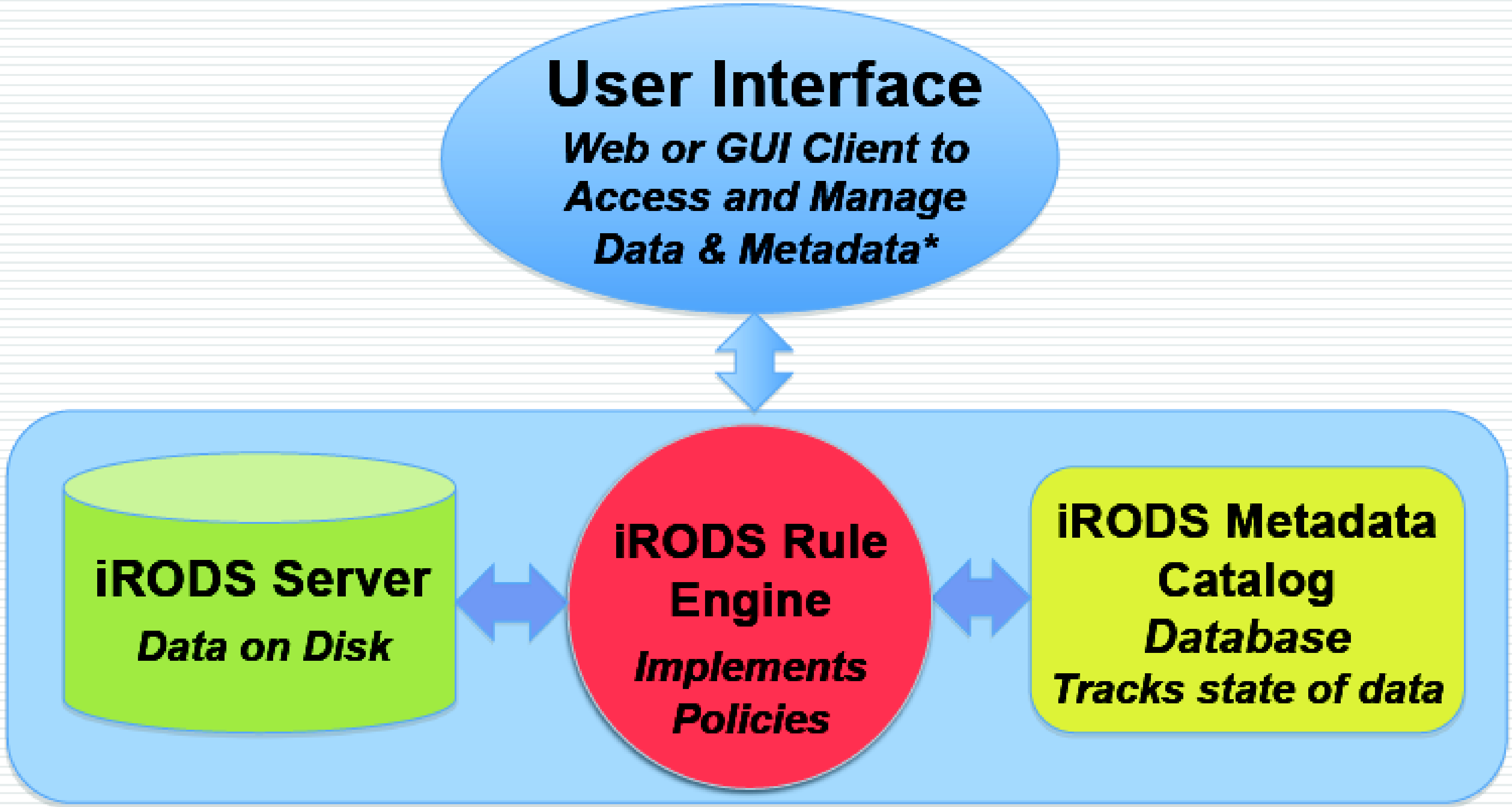
Distributed storage/file systems

- Share data
  - Using diverse, distributed hardware infrastructure
  - Between or within research groups
- Important parameters
  - Performance
  - Ease of access
  - Control (who should access)
  - Supported hardware
  - Stability & support (is the software well maintained?)

iRODS

- https://www.irods.org/
- integrated Rule Oriented Data System
- Developed by UNC and UCSD
- Build shareable collections from data distributed across file systems
- iCAT metadata catalog – descriptive metadata enabling searching and management
- Rule engine – user defined policies and rules for management, automation

Overview of iRODS Components



*Access data with: Web-based Browser, iRODS GUI, Command Line clients, Dspace, Fedora, Kepler workflow, WebDAV, user level file system, etc.

iRODS

- Data is viewed as a “Virtual Collection”
  - Stored on multiple resources
  - Having various provenance
- Finding the data
  - First query the metadata catalog
    - System metadata (user details, file details)
    - User-defined metadata (key-value units, domain specific)
  - Find pointer to the actual data server

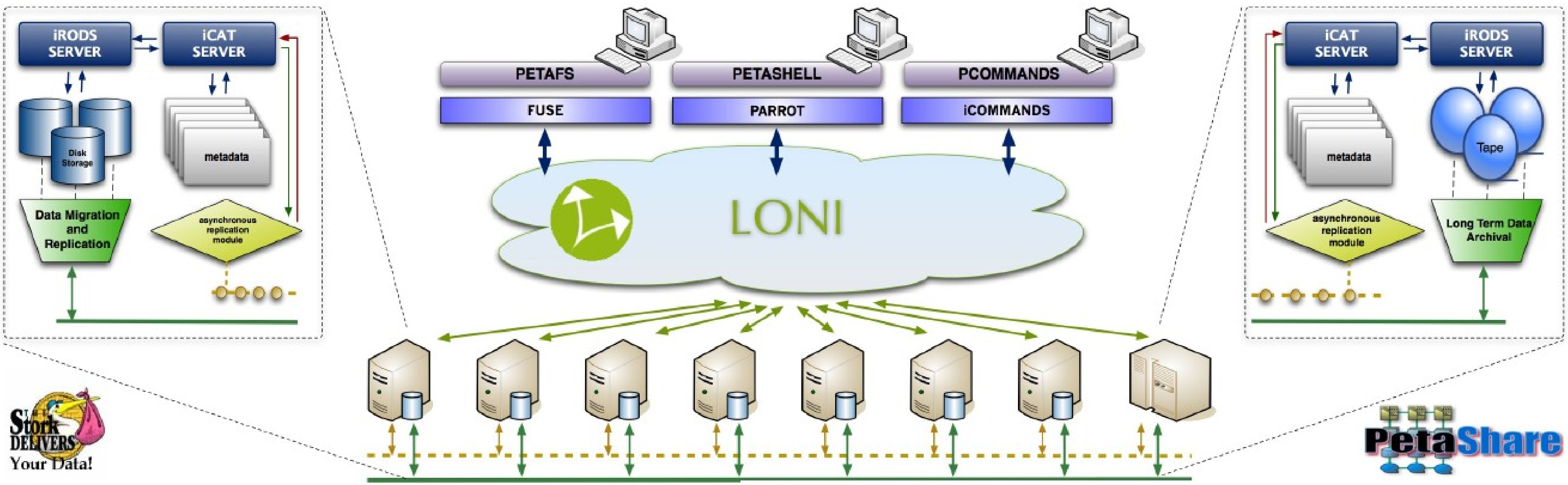
iRODS

- Policies
  - Community goals for sharing, access management
  - Examples: automatically replicate each file, access only with grid certificate
- Micro-services
  - Small functions – well defined operation: computeChecksum, zoom in (image)
  - Can be chained to implement workflows
  - C functions

PetaShare

- Developed at LSU (Dr. Tevfik Kosar)
- Online in LONI
- http://www.petashare.org/
- Provides
  - Global namespace across distributed resources
  - Interfaces to access data
  - Metadata management

Architecture



- From PetaShare tutorial

PetaShare access

- Petafs
  - mount petashare in your file system
  - using FUSE (requires Linux & root privileges)
- Petashell
  - Special shell that allows you to directly access petashare resources
- pcommands
  - Simple file management commands
- Web portal

Google File System

- Design requirements
  - Component failures are the norm
  - Huge files
  - Most files are changed by appending new data (not overwrite)
  - Workload: large streaming reads or small random reads
  - High bandwidth more important than latency

GFS

- Operations
  - Create, delete, open, close, read, write
  - Snapshot – create a copy of a file/directory
  - Record append – multiple clients append to same file
- Architecture
  - Single master (maintains all metadata)
  - Multiple chunkservers
  - Files are divided into fixed size chunks(unique chunk id)
    - Replicated on multiple servers

GFS

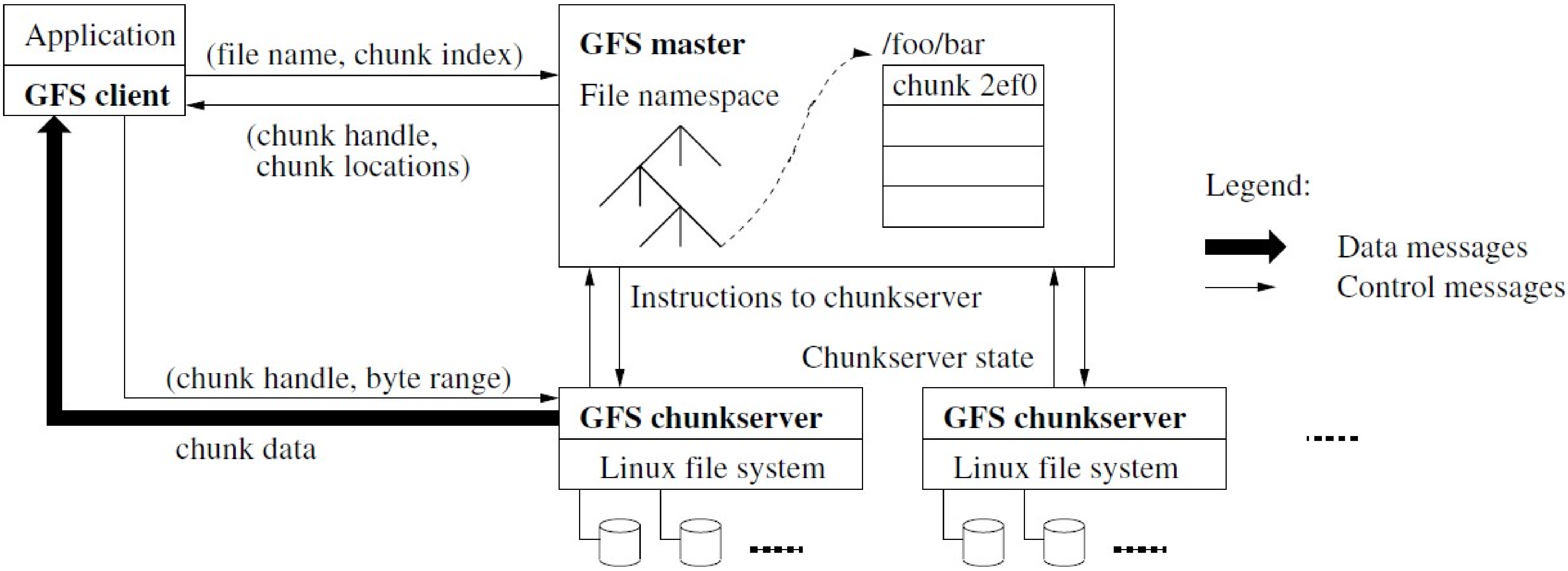


Figure 1: GFS Architecture

- From GFS paper (chunk index = offset/chunk size)

Distributed Hash Tables (DHT)

- Hash-table
  - One of the most important data structures invented
  - Key-value pairs
- DHT
  - Distributed (decentralized)
  - Scales
  - Fault-tolerant (nodes failing)
- Now used for distributed file systems

DHT

- keyspace: The set of valid keys (example: 160 character strings). Other keys can be mapped to this
- Key partitioning – distribution of keys to nodes
  - Usually according to some distance between keys
- Overlay network
  - Each node has a set of links to other nodes (neighbors)
  - To find a key, each node X passes a request to the node that is “closer” to that key than X (key-based routing)
  - Trade-off between degree (number of neighbors) and route length

Other systems

- Amazon S3
  - Commercial storage solution
  - http://aws.amazon.com/s3/
- HPSS
  - High-performance storage system (not distributed)
  - http://www.hpss-collaboration.org/technology.shtml

Remote I/O

- Accessing files over a network, options:
  - Copy the file locally (staging)
  - Access the file from where it is located (remote I/O)
  - Moving application to data or move both application and data (not widely used)
- Options for Remote I/O implementations
  - Generic middleware
  - GridFTP
  - Parrot
  - More

Remote I/O Using Middleware

- Use CORBA, Java RMI, SOAP or other RPC mechanisms to implement remote data objects
- Example: Network File System (NFS)
  - Implemented using SUN RPC
    - Initially designed for NFS, but generic enough
- NFS Concepts
  - Possible to use UDP or TCP for transport
  - “dumb” server, “smart” client
    - client responsible to check for reliability
  - stateless server (doesn't store client state)

NFS

- On top of RPC
  - XDR (External Data Representation)
    - Describes common data types
  - Procedure calls
    - get/set attributes, read, write
    - remove, create
    - link
    - readdir, rmdir
  - the Mount protocol
    - Map a path on the server to a handle that client can use
    - in NFSv4 mount is included in the NFS protocol

Remote I/O / Middleware

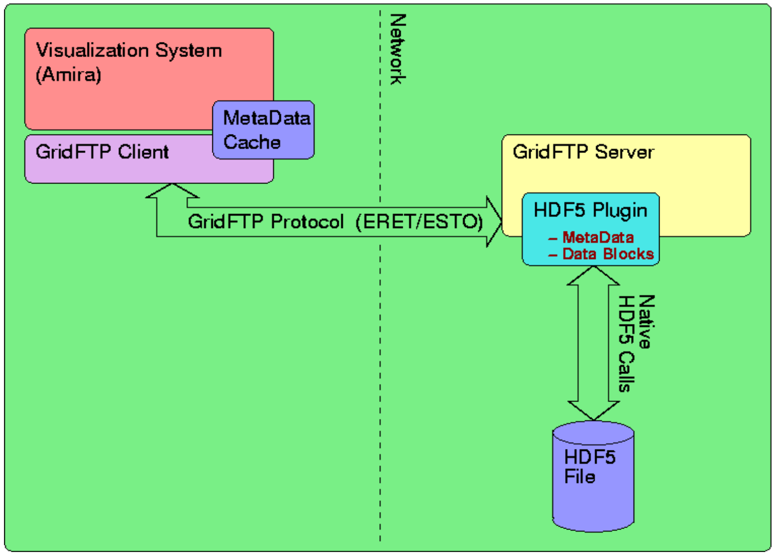
- Advantages
  - Anybody can implement its own remote I/O
  - Flexible choice of remote operations
  - Low implementation effort
- Disadvantage
  - Possible reduced data transport performance over WAN because lack of support for fast protocol
  - Latency issues (see previous lecture)
  - Middleware overhead (not always useful)

GridFTP

- Discussed in Lecture 2 as protocol for file transfer
- Has extension for Remote I/O
    - Including flexible server-side processing feature, allowing specification of custom operation on remote data
    - ERET <module> <parameters> <filename>
    - Simple partial remote I/O module (offset, length)
        - You used this in your assignment

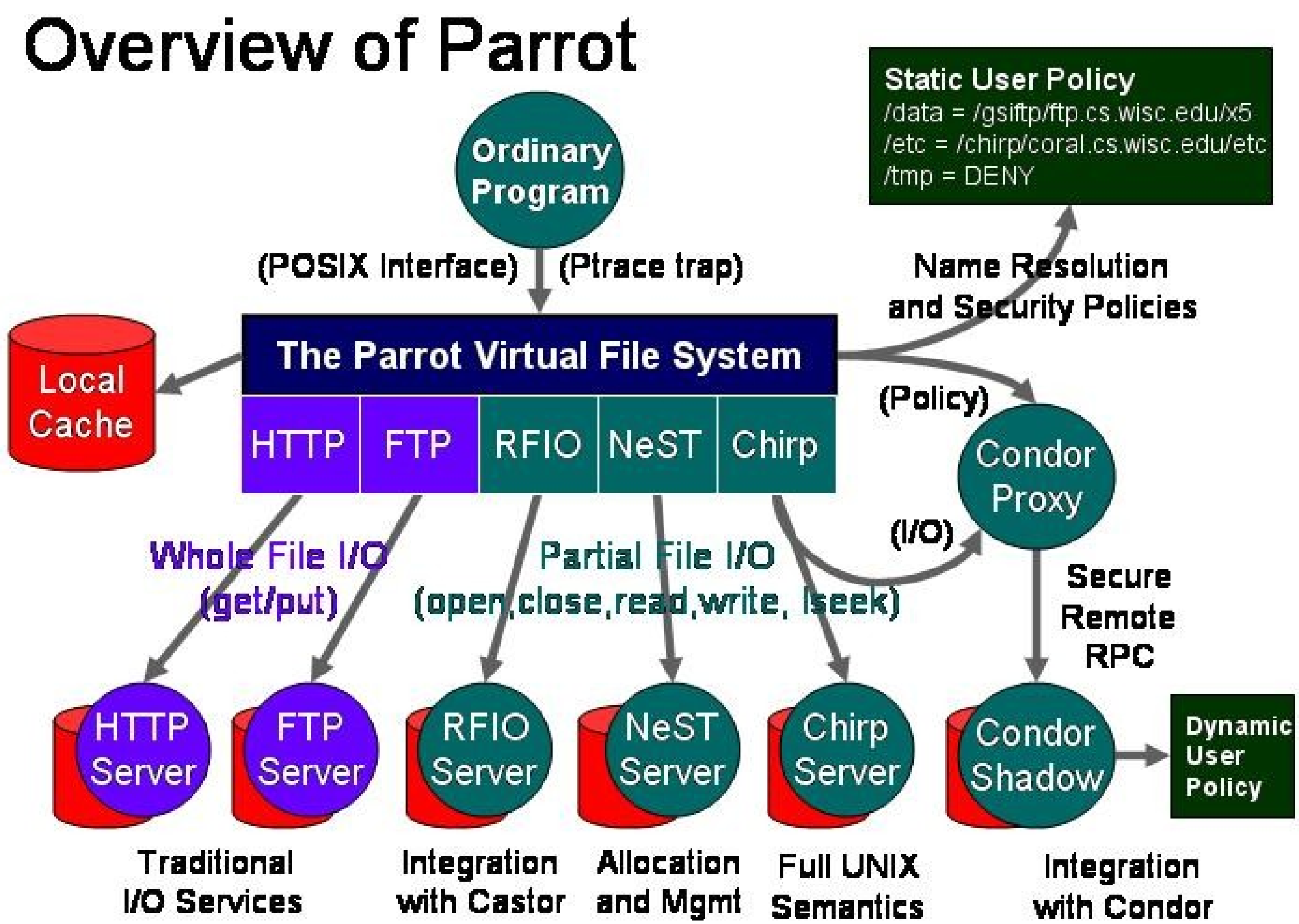
GridFTP

- But can be used to implement custom server-side processing modules
- Our team used it in the past to implement server-side selection of datasets in HDF5 files
- Advantage
  - Performance, maturity
- Disadvantage
  - System not built for remote I/O, somewhat difficult to use for custom I/O



Parrot

- Intercepts app I/O calls and redirects them to a remote file system
- Supports various protocols
  - GridFTP
  - Chirp
- http://www.nd.edu/~ccl/software/



MPI-IO; RDMA

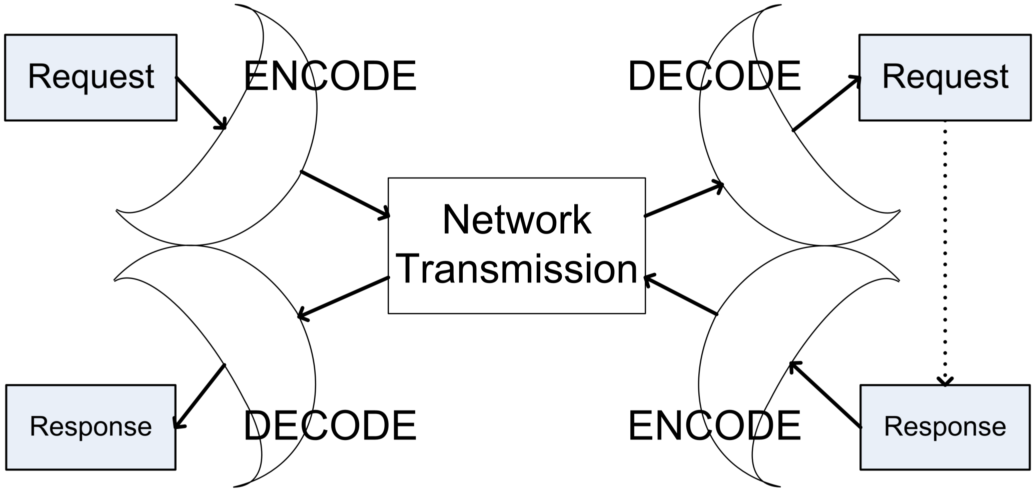
- MPI standard includes I/O interfaces
  - ADIO layer
    - Can implement I/O operations using remote file access
- RDMA (remote direct memory access)
  - Bypass the operating system
  - Direct access to remote memory
  - Implemented for high-speed interconnects
    - But some can work over wide area networks
  - Low latency, zero copy
  - Infiniband, Myrinet, Quadrics

eavivdata

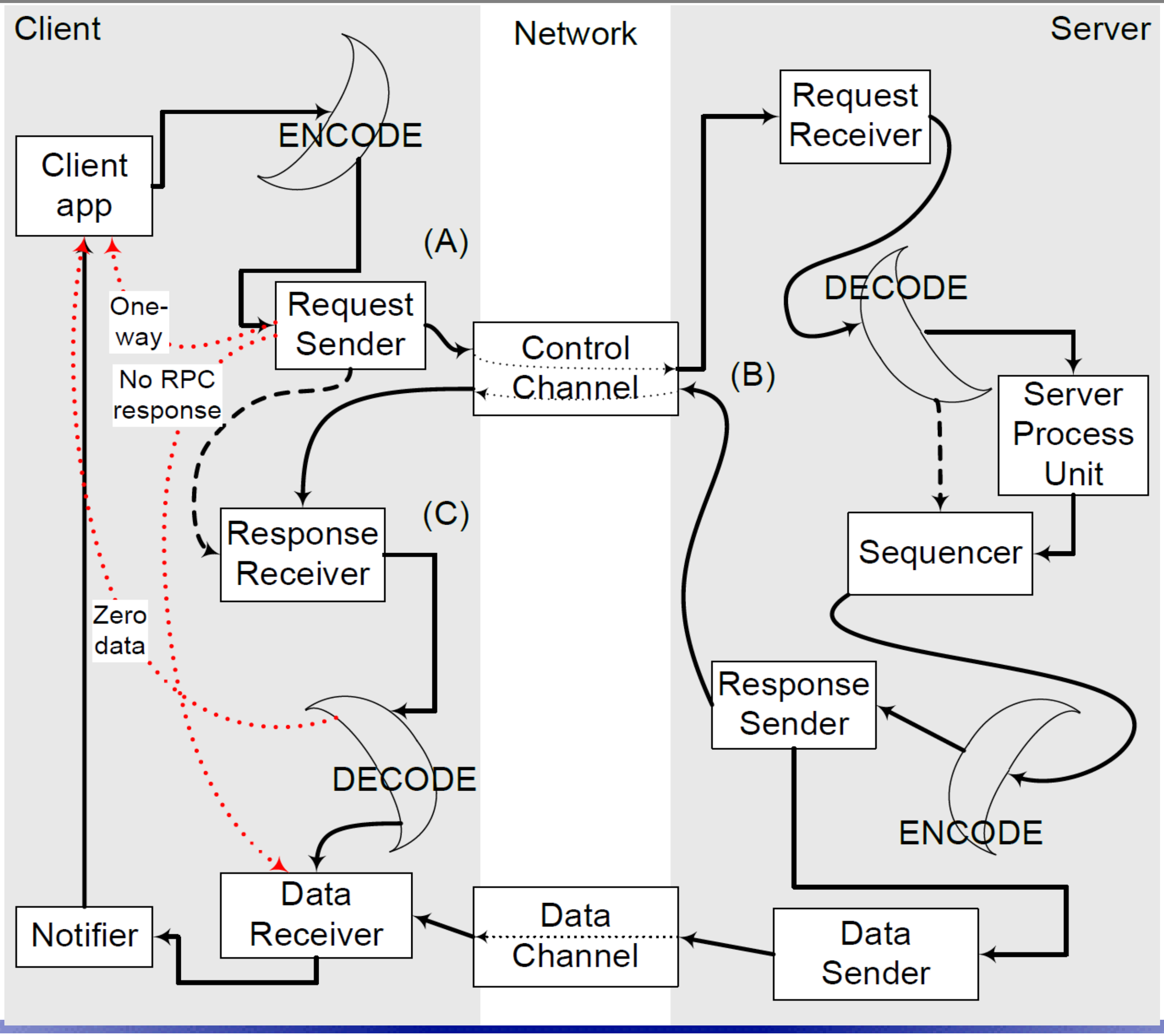
- Developed at LSU
  https://wiki.cct.lsu.edu/eaviv/Software#Data_Server
- Designed for
  - Speed (support of high-performance transport protocols)
  - High throughput (pipeline & bulk architecture - see middleware discussion)
  - Configurable remote operations
  - Asynchronous execution (non-blocking)
  - Applications (use as a library)
  - Read-only remote I/O

eavivdata Remote Data Access System

- Two channels: control, data
- Control channel
  - Configurable operations
  - RPC encoding (XML-RPC)



architecture diagram

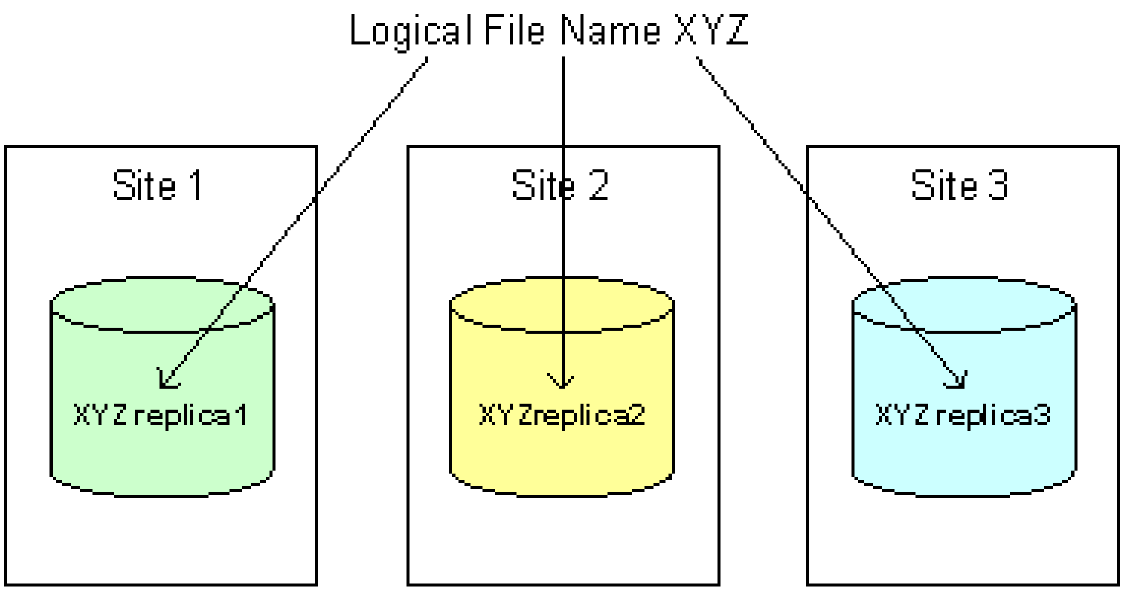


Data Staging and Replication

- Staging – copying the entire file for local access
- Compared to remote I/O
  - Useful if most of file is needed
  - Can analyze when staging or remote I/O is preferred (Ibrahim Suslu's LSU PhD thesis has this topic: http://etd.lsu.edu/docs/available/etd-06082010-092441/)
- Replication
  - Widely used in conjunction with staging
  - Have access to nearby replicas (LHC Grid)
  - Have redundant copies for reliability

Replica Systems

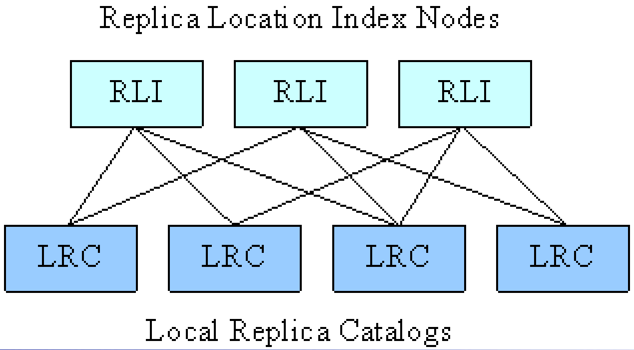
- Generally working by mapping a “logical file name” to one or more “physical file names” pointing to locations of the replicated file on storage systems



from www.globus.org

Replica System Options

- Consistency
  - Some guarantee that the physical file name points to a valid location, and all copies are identical
  - No guarantee, just a catalog, user is responsible for consistency
- Redundancy
  - The mapping from logical to physical names could be stored on multiple servers
  - Can have multiple tiers



Replica Systems

- Replica Placement
  - Static, policy-based
  - Dynamic, using some optimization criteria
- Partial Replicas
- Versions
- Metadata catalog
  - Search logical file names by attributes (physical variable, date, size, etc..)
  - From logical file name find physical file names

Scheduling & Resource Selection

- Scheduling appears at all levels
  - Operating system
  - Cluster/parallel system
  - Grid/distributed system
- The process of assigning work to resources (processor/node/cluster)
- Two main approaches to scheduling
  - Maximize resource utilization
  - Maximize application utility

Maximize resource utilization

- Want to fill out the resources as best as possible
- Might mean some jobs get delayed
- Scheduling at higher layers usually means under-utilization of resources at lower layers
  - Cluster scheduling usually works by reserving complete nodes for jobs
    - Goal is to maximize cluster usage
    - Means OS scheduler cannot optimize node utilization
  - Distributed application scheduling may conflict with cluster resource utilization
  - Think global optimization vs. local optimization

Maximize application utility

- Goal is to run one (our) application as fast as possible
  - Resource utilization will likely suffer
- Scheduler trade-off decision
- Well known in OS scheduling
  - Trade-off cpu utilization and interactive performance
- Cluster scheduling
  - Option of interactive job queue
- Distributed computing scheduling
  - ? Not that many distributed interactive apps (yet)

Resources taken into consideration

- Focusing on distributed applications
- Traditionally CPUs are the main factor
    - What CPUs/cycles are free, how many
- If CPU wait time is the only one optimized(minimized), waiting for data/networks will have a detrimental effect
    - Get CPU immediately, but moving the data to the computation takes a long time
    - If many data transfers will be scheduled at the same time, the performance will suffer

Data & Networks

- Data and Networks should be taken into account when scheduling distributed applications
- Data scheduling, make sure that storage systems are not overloaded with data requests
  - Distribute data access to multiple storage systems
  - Run data accesses sequentially
    - If they all run once, utility is reduced
- Network scheduling
  - Possible with network circuit services (mentioned in previous lecture)
  - Ensure that network is reserved for the transfer

Co-allocation

- Application uses multiple resources and multiple types of resources (including data and networks)
- Need all the resources to be allocated at the same time (co-allocation) – or app may not be able to run
- All-or-nothing
- Need support for hold/ release/ commit resource allocation