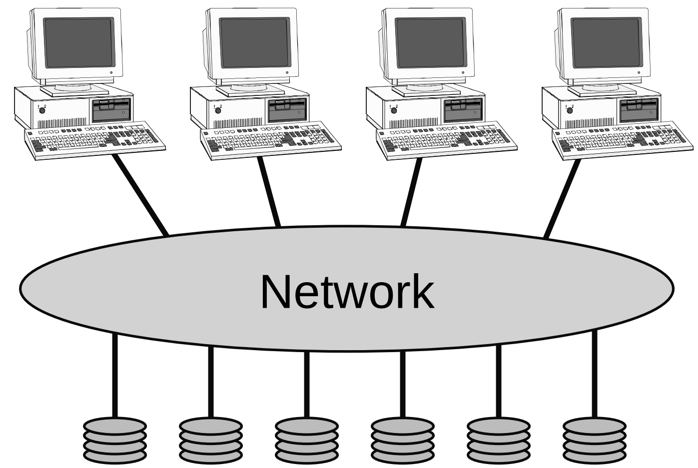
The Swarm Scalable Storage System

John H. Hartman

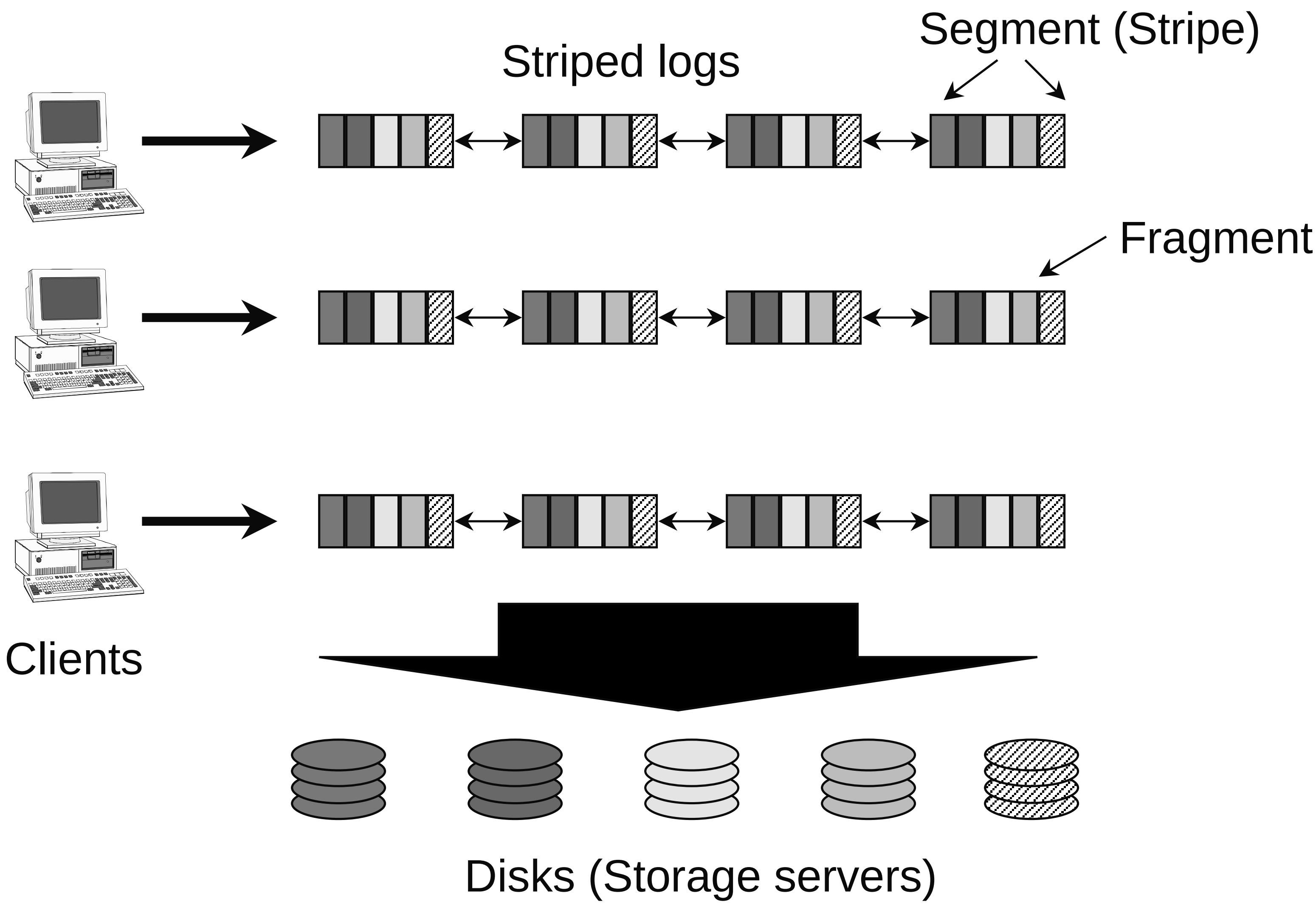
Department of Computer Science
The University of Arizona
jhh@cs.arizona.edu

Swarm

- Direct client/disk transfers
- Log-based striping
- RAID-style fault-tolerance
- Multiple access protocols
  - NFS, SIO, Swarm, HTTP



Log-based striping



Log-based striping advantages

- Striping independent of file characteristics
  - File size doesn’t affect stripe width
- Fault-tolerance through reconstruction
  - Parity independent of file characteristics
  - Append-only log avoids parity update
- Each client writes its own log
  - Synchronization only for allocation

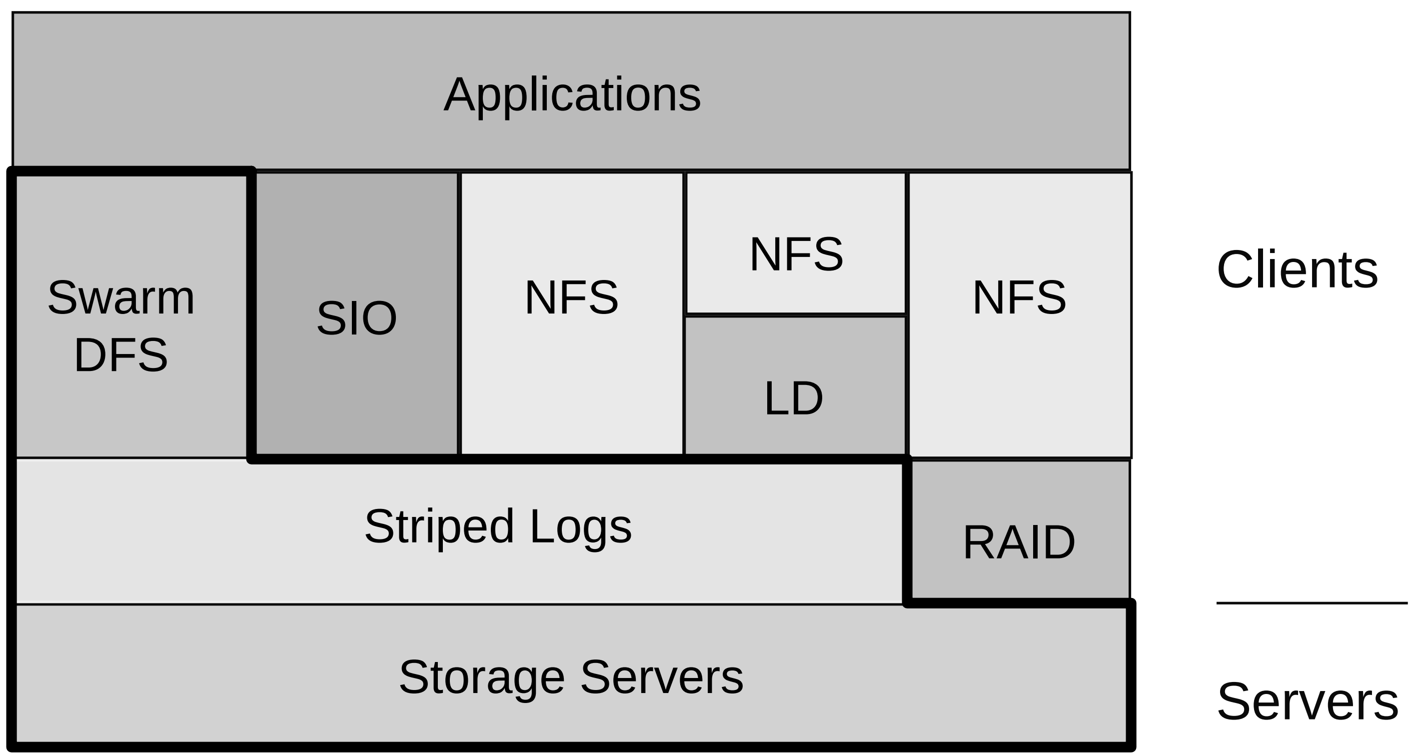
Recovery via roll-forward

- Services store *records* and data blocks in log
  - Records describe state changes
  - Format defined by services
  - Records replayed to service after crash
    - Enables roll-forward to current state
- *Checkpoints* limit extent of replay
  - Each service periodically checkpoints its state
  - Data blocks consistent with records
  - Older records discarded

Swarm clients run the show

- All distributed functionality handled by clients
  - Naming, allocation, coherency
- Why the clients?
  - Scales with number of clients
  - Scales with client performance
  - Better for autonomous clients
  - Simple servers
    - Autonomous
    - Limited interface and functionality

Swarm's layered architecture



Swarm storage server

- Stores log fragments
  - Identified by unique fragment ID

- Supports read, write, delete by clients
  - All operations atomic
  - Overwrite supported, but not optimized
  - Must delete entire fragments (probably stripes)

- ACLs on byte-ranges provide protection
  - Limits clients that may read/write/delete a range
  - Duplicated on parity server for reconstruction

Fragment metadata

- Servers are autonomous, but know the fragments they store form a log
- Stripe group information
  - Ids and servers of other fragments in same stripe (segment)
  - Provided to clients on request
  - Used to boot-strap system
    - Stripe group information also stored in a (more efficient) data structure in the log
  - Implicitly deleted when fragment is deleted

Stripe metadata

- Links between stripes that form log
  - IDs of previous and subsequent stripes in log
  - Stored in each fragment of stripe
  - Enable log replay after crash
- Explicitly deleted separately from fragments
  - Fragment deleted when cleaned; links may still be needed to replay log
  - Links deleted when replay is no longer needed (stripe is older than checkpoint)

Client's layered architecture

- Striped log
  - Creates, accesses, and cleans log
- Atomic Recovery Units
  - Atomicity for multiple writes to log
- Logical disk
  - Local virtual disk

Striped log layer

- Forms log and stripes it across servers
- Computes and stores parity
- Reconstructs missing fragments
  - Uses stripe group information or fragment metadata
- Cleans and deletes fragments
- Deletes links for stripes older than checkpoint

Atomic Recovery Unit layer

- Multiple local writes within a single ARU are atomic
- ARU records denote beginning/end of ARUs in log
- Records from higher-level services intercepted and tagged with ARU ID
- During replay only records from successful ARUs passed on to higher-level services

Logical disk

- Provides illusion of local disk
- Smaller namespace, allows overwrite
  - Mapping from disk address to log address
- Better than a real disk:
  - Provides ARUs
  - Interprets layout information
  - Striped across multiple real disks
- Supports standard file systems with minimal modification

Swarm prototype

- Proof of concept
- Platform for further research
- Hardware:
  - Servers: 90MHz Pentiums, Scout OS
  - Clients: 200MHz PentiumPros, Linux
    - Local file system only
  - Network: 100Mb/sec switched Ethernet
- Internal release by end of semester

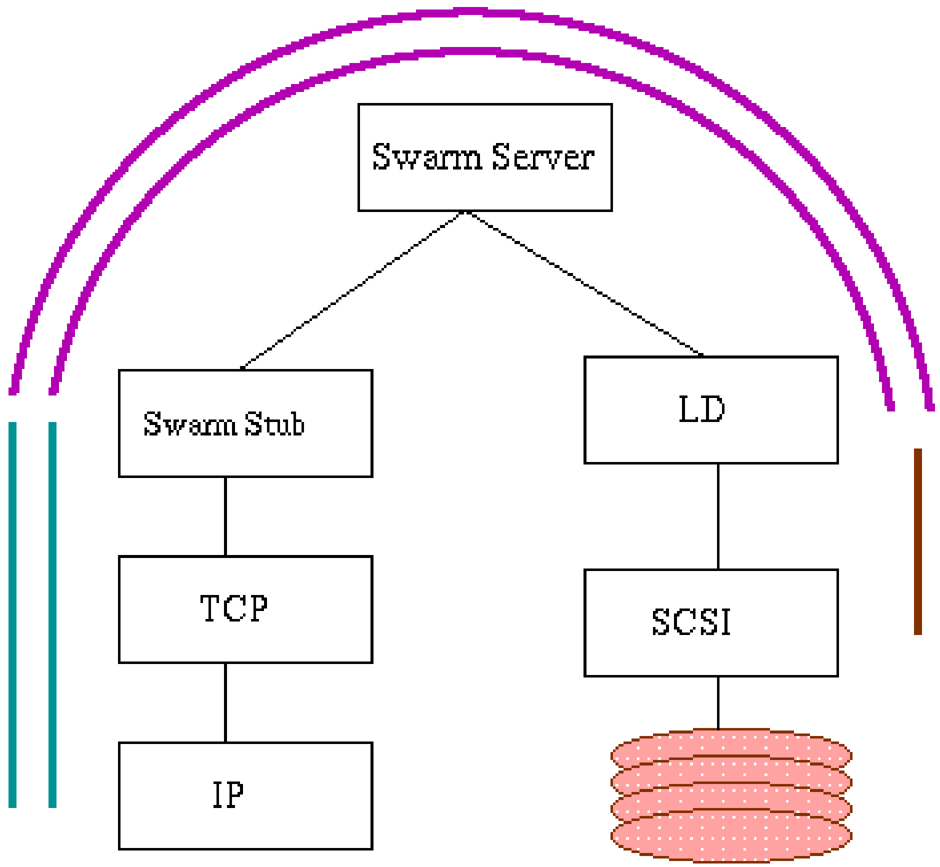
Client

- Linux
  - Sometimes a struggle
- Local FS uses log layer directly
  - Standard FS operations; no recovery
- Simple cleaner nearly working
  - Cleans local log only
- Size
  - ~12K lines of C
  - ~63KB of x86 binary

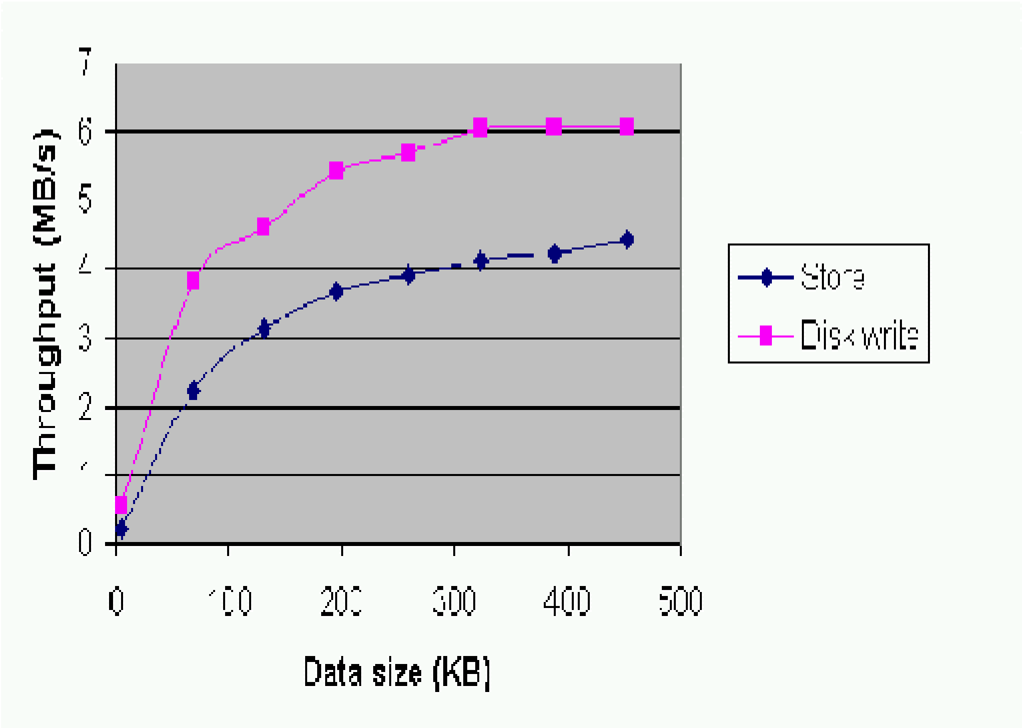
Server prototype

- Scout OS
  - Modular, communication-oriented OS
  - *Path* abstraction
- MIT Logical Disk (LD) software
- TCL-based protocol
  - Disk I/O swamps command interpretation
- Size:
  - ~27K lines of server-specific C code
  - ~407KB of x86 binary (entire kernel)

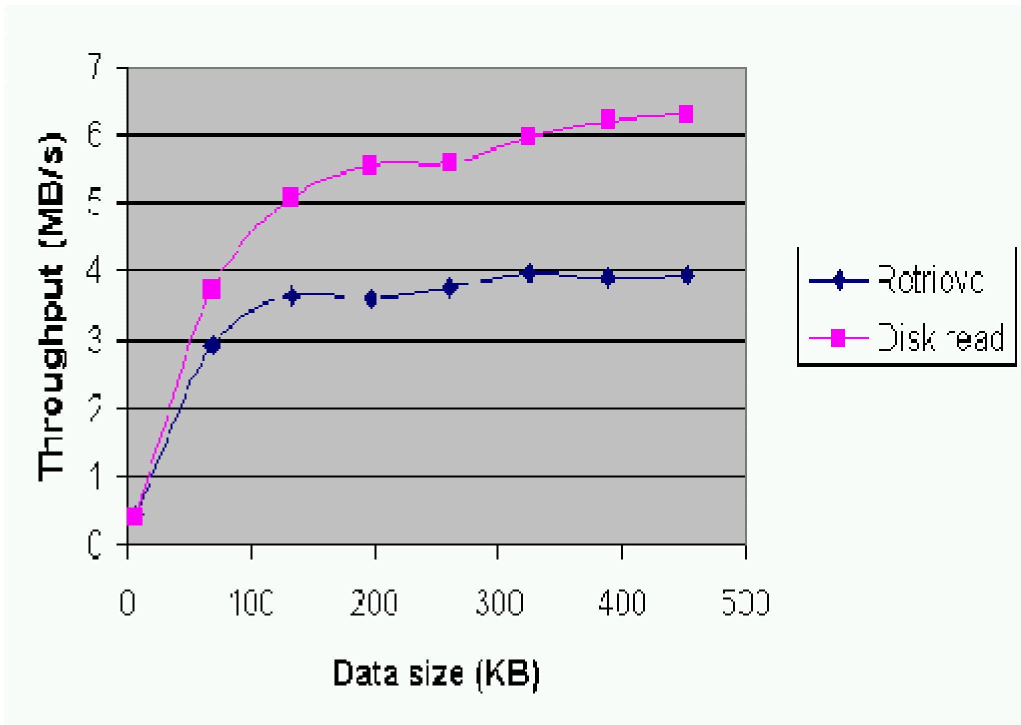
Scout Swarm server



Fragment write performance



Fragment read performance



Performance summary

- Write: 70% disk bandwidth @ 200KB
- Read: 66% disk bandwidth @ 200KB
- Bottlenecks:
  - LD software too complex
  - Extra copies in Scout SCSI driver

Client/server similarities

- Client software functionality similar to server:
  - Log layer provides efficient fragment storage
  - ARU layer provides efficient atomic updates of data and metadata
  - Logical disk layer simplifies server metadata
- Server will be reimplemented using client software

Other Swarm research

- Support for parallel file systems
  - SIO support being developed
  - Expressing and retaining relationships between data blocks
- Secure reconstruction on untrusted clients
- Distributed cleaning
- Locking
  - Add lock service to servers?