NFSv4 as the Building Block for Fault Tolerant Applications

Alexandros Batsakis

Hopkins Storage Systems Lab, Department of Computer Science

Overview

Goal:

To provide support for client recoverability and application fault tolerance through the NFSv4 file system

Motivation:

Conventional file systems do not meet the requirements of parallel applications

Performance:

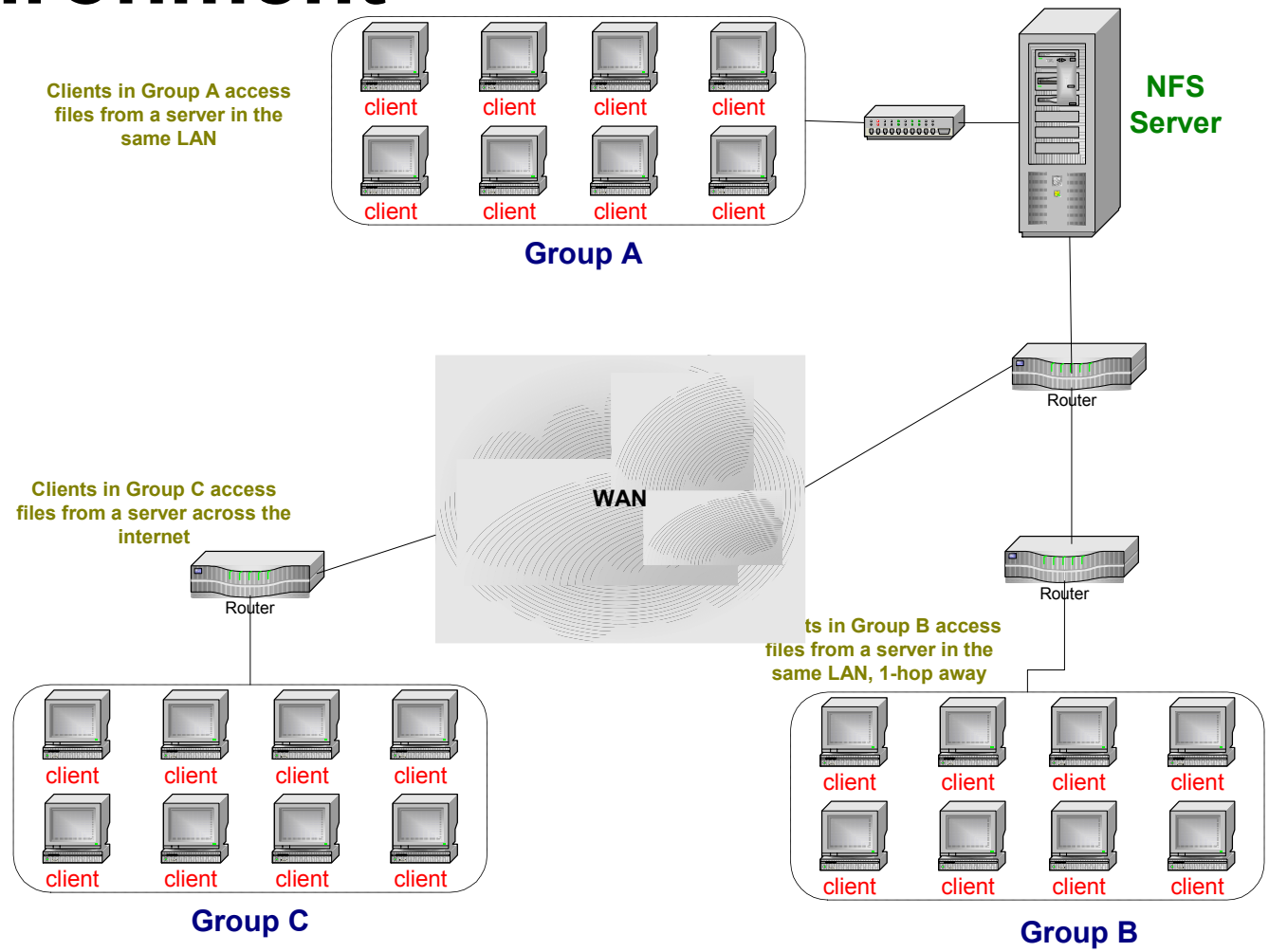
Mechanisms for recoverability enable efficient write-sharing among clients and improve file system performance

What we propose:

"the file system aids in the recovery of failed clients"

- Application support
  - Facilitate fault-tolerance operations
- NFSv4 extensions
  - Extend the delegation concept to multiple clients
  - Add client-to-client operations

Environment



Environment

- High-performance parallel applications running on PC clusters
- Data access from “remote” NFS file server
- Applications communicate through file sharing and/or message passing
- Applications should be able to reproduce the prior-to-failure state at recovery time

Problem statement

- Rollback recovery protocols incur substantial overhead when nodes communicate through file sharing
  - **Read logging**
    - File read is logged to disk to guarantee availability at recovery
  - **Output Commit**
    - Application state is logged to disk before write
    - Ensure that state that generated the write is reproducible
- Conventional file systems do not support write sharing efficiently
  - **Synchronous write-back**

Clients have to wait for I/O completion (blocking)

NFSv4 & Parallel applications

- Does NFSv4 support file sharing efficiently?
  - Yes, when read-only sharing
  - No, when file sharing involves writing
- Does NFSv4 support recoverability?
  - No built-in support for repeatable reads or fast output commits

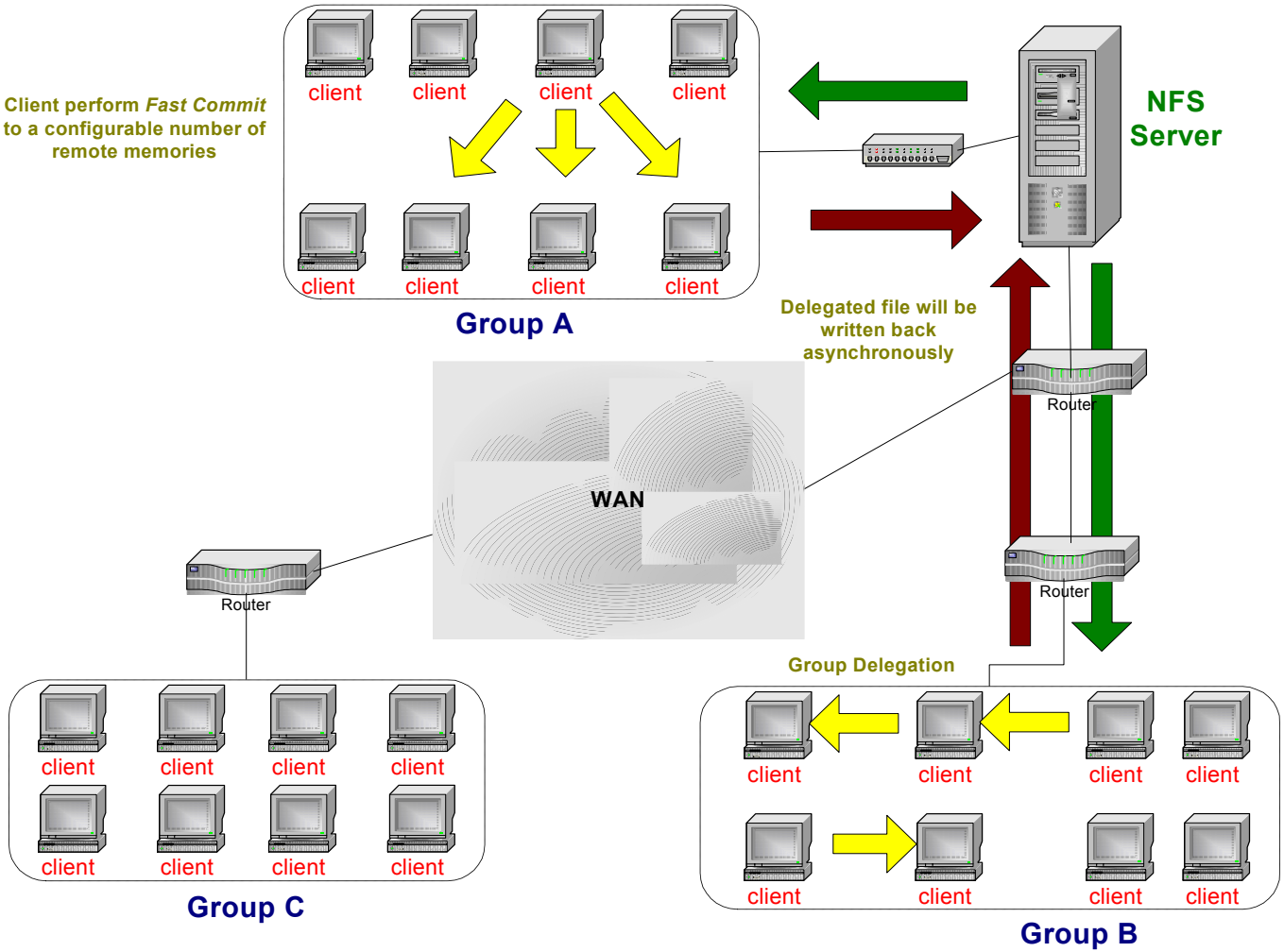
Server vs. Client Recoverability

- Server fault tolerance
  - Data availability through replication or shared disk
  - Preserves file system data only
- Application fault tolerance
  - Logging; *preserves application state*
  - Repeatable reads; *avoid read logging*
  - Fast output commit; *avoids synchronous write-back*

Mechanisms

- Group delegation
  - Current delegation model does not efficiently support file sharing
  - File sharing takes place directly between clients
  - Server is unaware of group delegation
  - Client that initiated the delegation acts as group representative
- Fast commit
  - Clients perform *COMMIT* operation without contacting server
  - Updates logged immediately to client remote memories
  - Reach stable storage at later time
- Cooperative caching
  - Coordinates access to content in caches

Mechanisms


Client perform *Fast Commit* to a configurable number of remote memories
client
client
client
client
client
client
client
client
Group A
NFS Server
Delegated file will be written back asynchronously
Router
WAN
Router
Router
client
client
client
client
client
client
client
client
Group C
Group Delegation
client
client
client
client
client
client
client
client
Group B

Client recoverability

- Support for shared logging
- Application level protocols have two options:
  a. Clients log to a common file
     - Group delegation → serializability
     - Cooperative caching → sharing
     - Fast commit → performance
  b. Clients use their own log
     - Use fast commit to replicate log to other clients

Client recoverability (2)

- Elimination of read logging
  - Support for repeatable reads through versioning
- Checkpointing to reduce log size
- Applications are still in charge of semantics
  - When and what to log or checkpoint

A nice side-effect: Performance

- Data sharing works better
  - Eliminates write-backs to server
  - Significant boost when clients are close to each other
- Offload server
  - Write-sharing without server interactions
- Fast commit (asynchronous write-back)
  - Efficient file sharing
  - Keeps data closer to application

Considerations

- Security
  - Is it acceptable to assume that clients have the same privileges?
  - If not, solution depends on the security model used by each system
- Callback efficiency
  - Harder to break delegation as client group gets larger
- Cooperative caching & scalability

NFSv4 protocol modifications

- No server modifications are required
- With server modifications:
  - More natural support for group delegation:
    - Adjust delegation policy
    - Persistent delegation
      - Server can contact any node in group to query state
      - Delegation will not be revoked when initial client fails
  - Might provide solution to security problem
    - Clients use server for access control and authentication when sharing files

Conclusions

- Recoverability requirements for parallel applications
  - semantic extensions to NFSv4
- Client modifications to provide support for recoverability
  - *Group delegation, fast commit, cooperative caching*
- Modified NFSv4 supports:
  - shared logging, repeatable reads, fast output commit
- Performance benefits
  - Even when recoverability is not the goal