The Direct Access File System (DAFS)

Matt DeBergalis, Peter Corbett, Steve Kleiman,
Arthur Lent, Dave Noveck, Tom Talpey, Mark Wittle

Network Appliance, Inc.

Usenix FAST ’03

Tom Talpey

tmt@netapp.com

Outline

- DAFS
- DAT / RDMA
- DAFS API
- Benchmark results

DAFS – Direct Access File System

- **File access protocol, based on NFSv4 and RDMA, designed specifically for high-performance data center file sharing (local sharing)**
- **Low latency, high throughput, and *low overhead***
- **Semantics for clustered file sharing environment**

DAFS Design Points

- Designed for high performance
  - Minimize client-side overhead
  - Base protocol: remote DMA, flow control
  - Operations: batch I/O, cache hints, chaining
- Direct application access to transport resources
  - Transfers file data directly to application buffers
  - Bypasses operating system overhead
  - File semantics
- Improved semantics to enable local file sharing
  - Superset of CIFS, NFSv3, NFSv4 (and local file systems!)
  - Consistent high-speed locking
  - Graceful client and server failover, cluster fencing
- http://www.dafscollaborative.org

DAFS Protocol

- Session-based
- Strong authentication
- Message format optimized
- Multiple data transfer models
- Batch I/O
- Cache hints
- Chaining

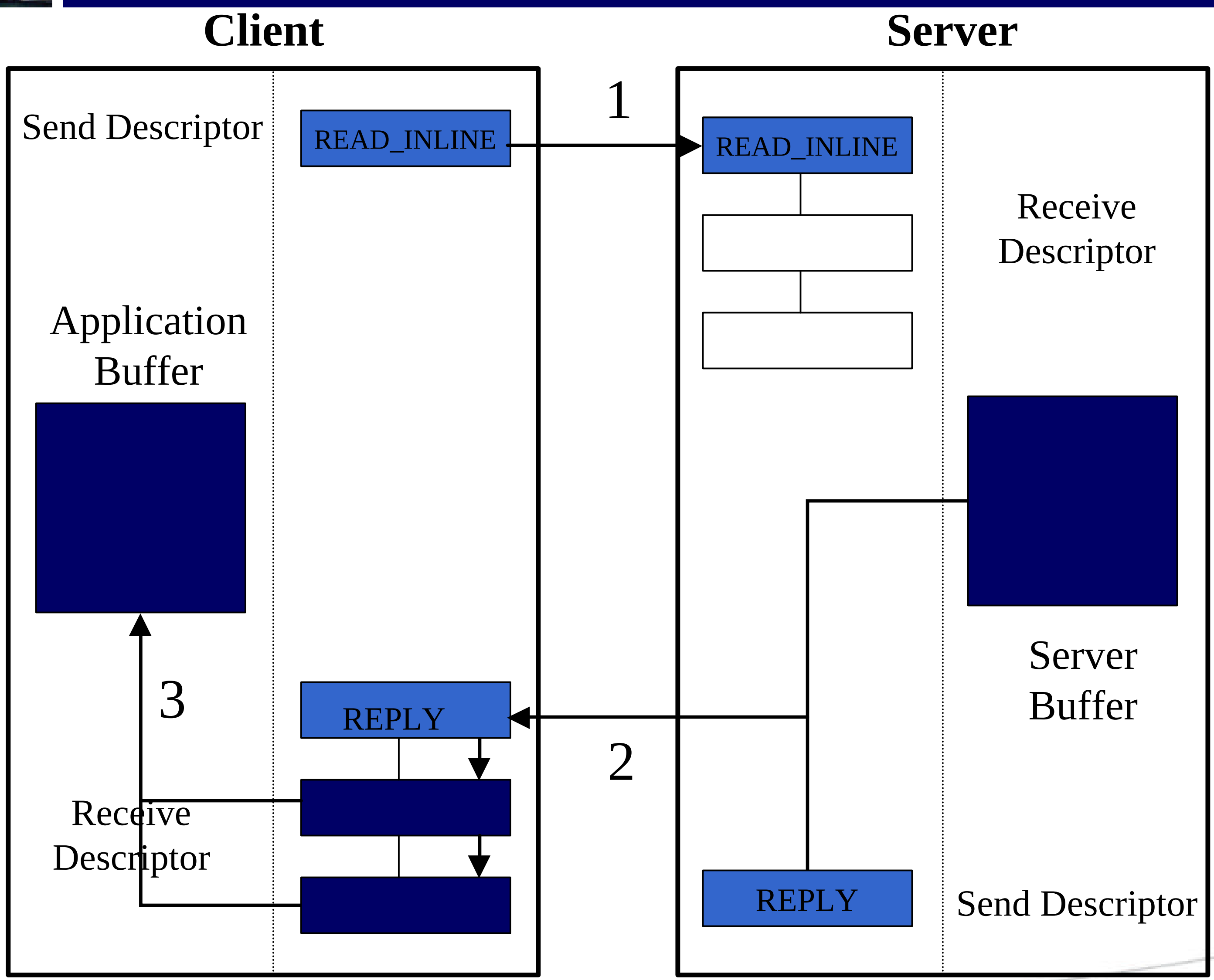
DAFS Protocol Enhanced Semantics

- Rich locking
- Cluster fencing
- Shared key reservations
- Exactly-once failure semantics
- Append mode, Create-unlinked, Delete-on-last-close

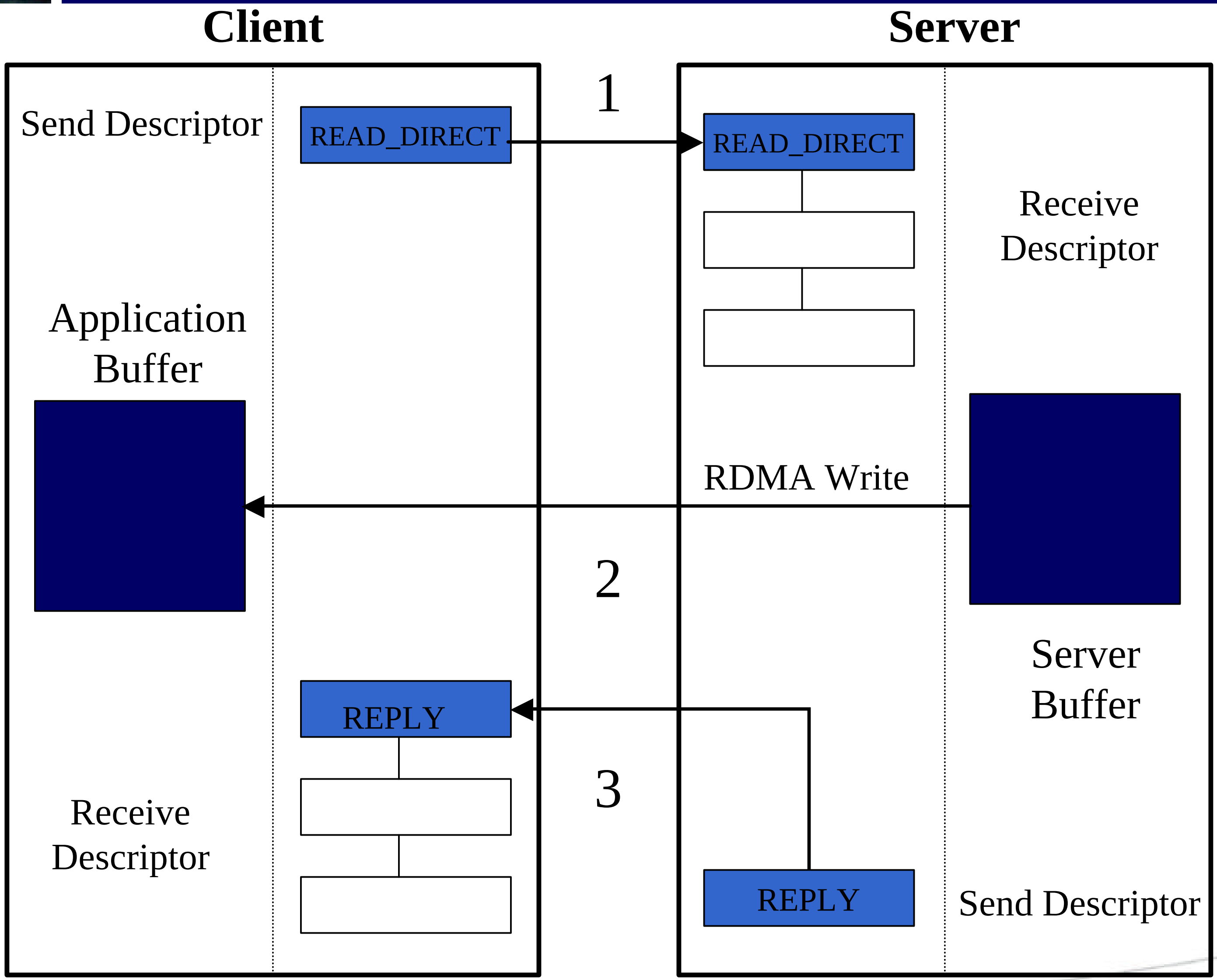
DAT – Direct Access Transport

- Common requirements and an abstraction of services for RDMA - Remote Direct Memory Access
  - Portable, high-performance transport underpinning for DAFS and applications
  - Defines communications endpoints, transfer semantics, memory description, signalling, etc.
- Transfer models:
  - Send (like traditional network flow)
  - RDMA Write (write directly to advertised peer memory)
  - RDMA Read (read from advertised peer memory)
- Transport independent
  - 1 Gb/s VI/IP, 10 Gb/s InfiniBand, future RDMA over IP
- http://www.datcollaborative.org

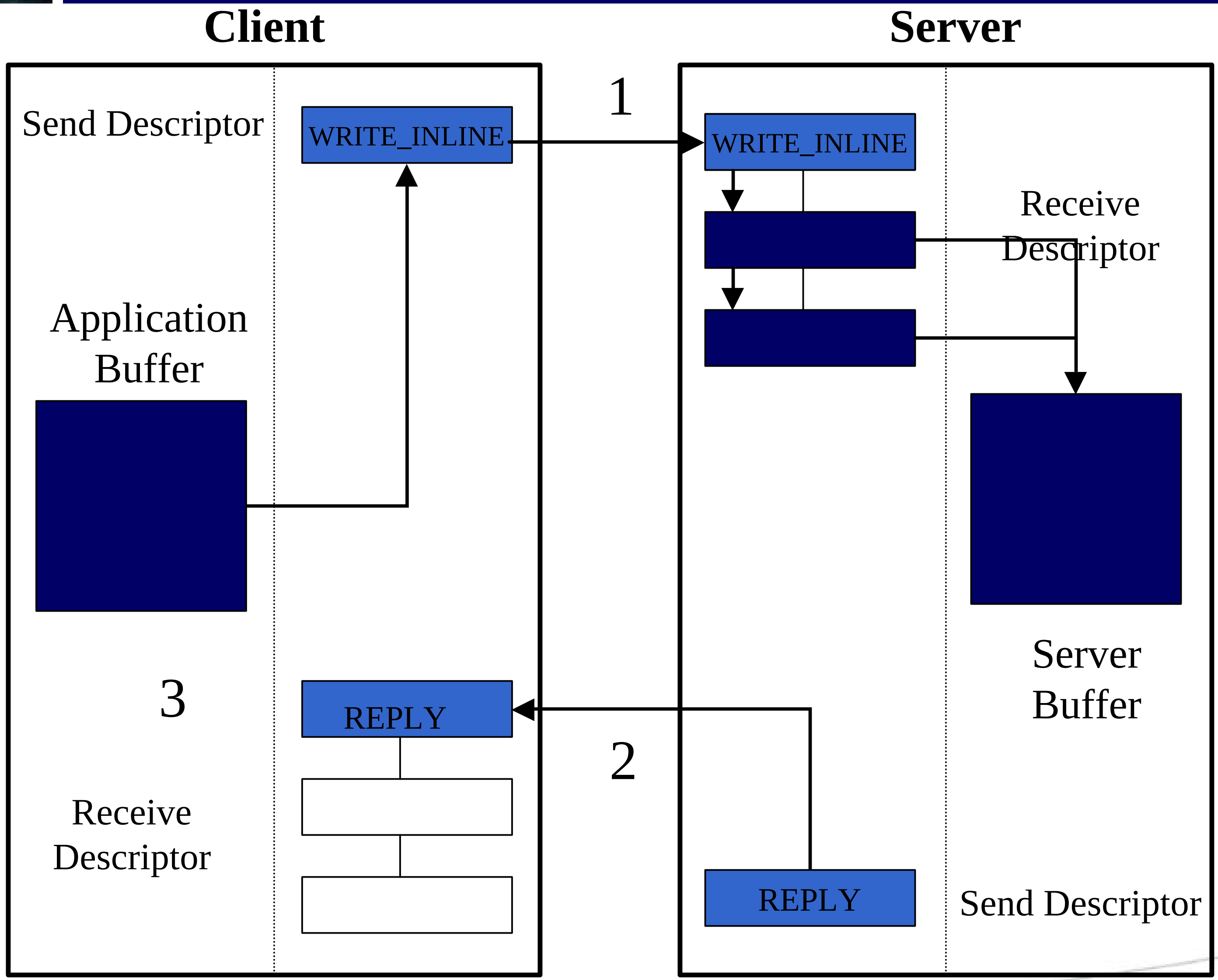
DAFS Inline Read



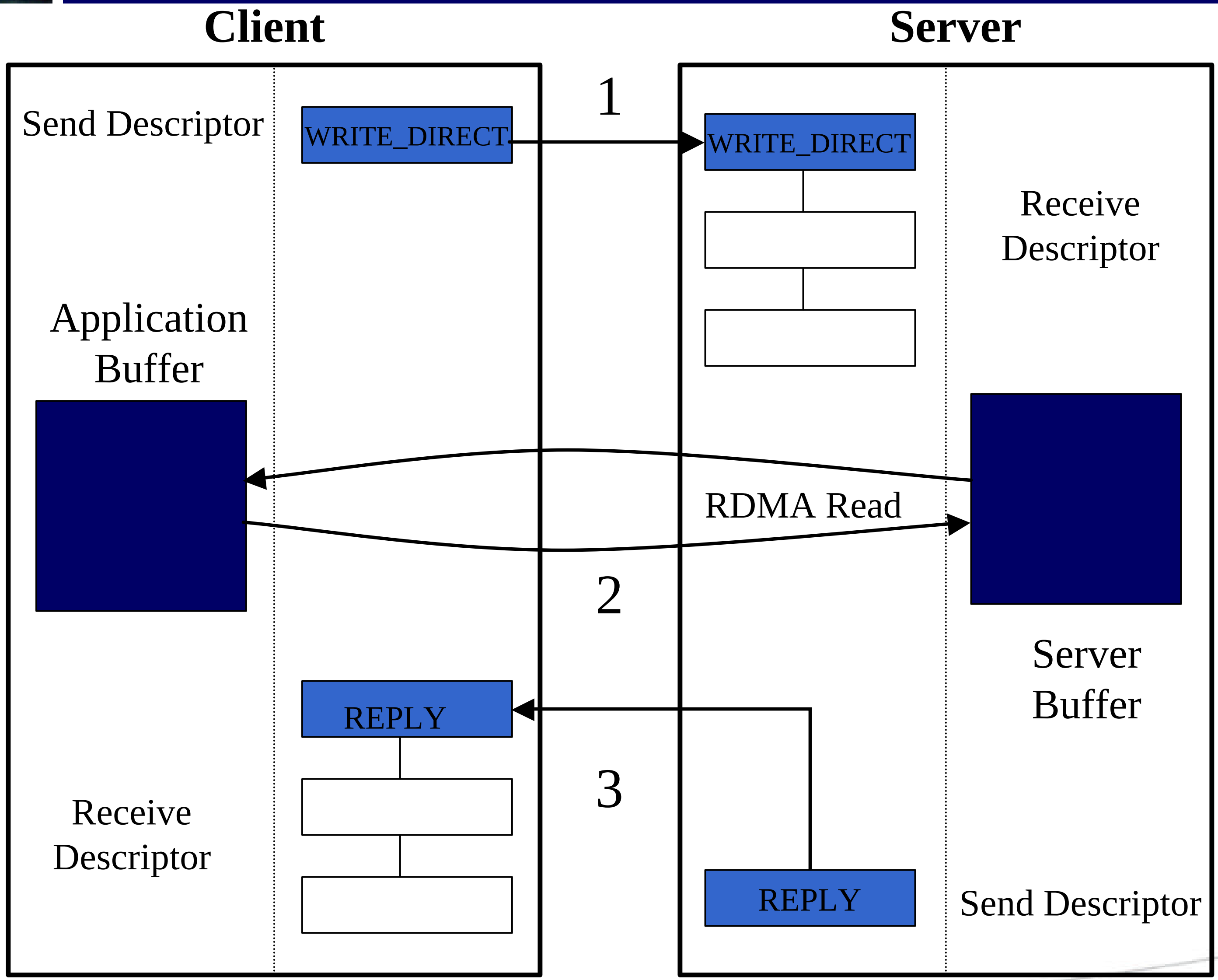
DAFS Direct Read



DAFS Inline Write

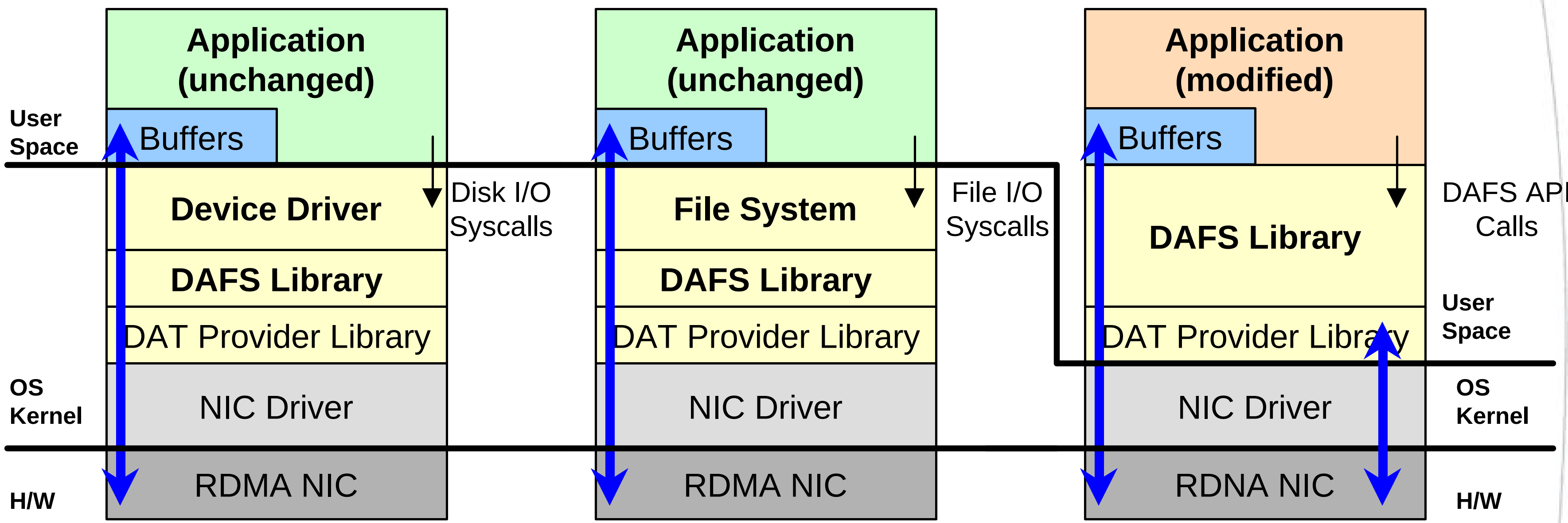


DAFS Direct Write



DAFS-enabled Applications

Raw Device Adapter | **Kernel File System** | **User Library**



Raw Device Adapter

- Kernel-level plug-in
- Looks like raw disk
- App uses standard disk I/O calls
- Very limited access to DAFS features
- Performance similar to direct-attached disk

Kernel File System

- Kernel-level plug-in
- Peer to local FS
- App uses standard file I/O semantics
- Limited access to DAFS features
- Performance similar to local FS

User Library

- User-level library
- Best performance
- Full application access to DAFS semantics
- *Paper focuses on this style*

NetworkAppliance®

DAFS API

- File based: exports DAFS semantics
- Designed for highest application performance
- Lowest client CPU requirements of any I/O system
- Rich semantics that meet or exceed local file system capabilities
- Portable and consistent interface and semantics across platforms
  - No need for different mount options, caching policies, client-side SCSI commands, etc.
  - DAFS API interface is completely specified in an open standard document, not in OS-specific documentation
- Operating system avoidance

The DAFS API

- **Why a new API?**
  - **Backward compatibility with POSIX is fruitless**
    - **File descriptor sharing, signals, fork()/exec()**
  - **Performance**
    - **RDMA (memory registration), completion groups**
  - **New semantics**
    - **Batch I/O, cache hints, named attributes, open with key, delete on last close**
  - **Portability**
    - **OS independence and semantic consistency**

Key DAFS API Features

- Asynchronous
  - High performance interfaces support native asynchronous file I/O
  - Many I/Os can be issued and awaited concurrently
- Memory registration
  - Efficiently prewires application data buffers, permitting RDMA (direct data placement)
- Extended semantics
  - Batch I/O, delete on last close, open with key, cluster fencing, locking primitives
- Flexible completion model
  - Completion groups segregate related I/O
  - Applications can wait on specific requests, any of a set, or any number of a set

Key DAFS API Features

- **Batch I/O**
  - Essentially free I/O: amortizes costs of I/O issue over many requests
  - Asynchronous notification of any number of completions
  - Scatter/gather file regions and memory regions independently
  - Support for high-latency operations
  - Cache hints
- **Security and authentication**
  - Credentials for multiple users
  - Varying levels of client authentication: none, default, plaintext password, HOSTKEY, Kerberos V, GSS-API
- **Abstraction**
  - server discovery, transient failure and recovery, failover, multipathing

Benchmarks

- **Microbenchmarks to measure throughput and cost per operation of DAFS versus traditional network I/O**
- **Application benchmark to demonstrate value of modifying application to use DAFS API**

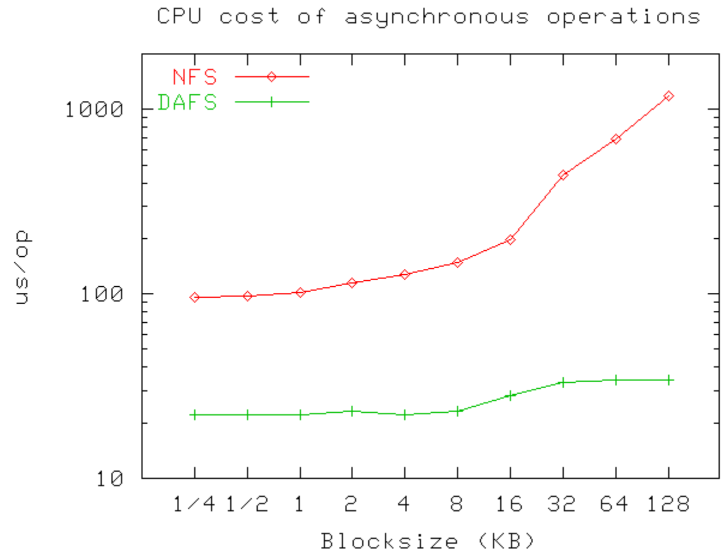
Benchmark Configuration

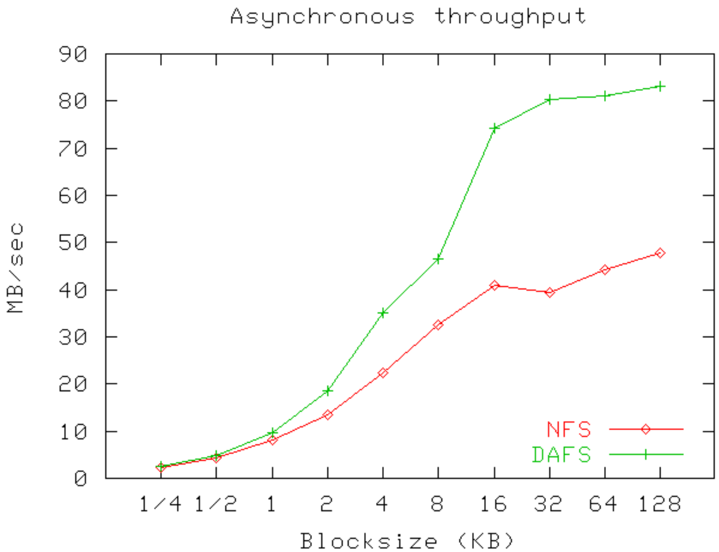
- User-space DAFS library, VI provider
- NetApp F840 Server, fully cached workload
  - Adapters (GbE):
    - Intel PRO/1000
    - Emulex GN9000 VI/TCP
  - NFSv3/UDP, DAFS
- Sun 280R client
  - Adapters:
    - Sun “Gem 2.0”
    - Emulex GN9000 VI/TCP
- Point-to-point connections

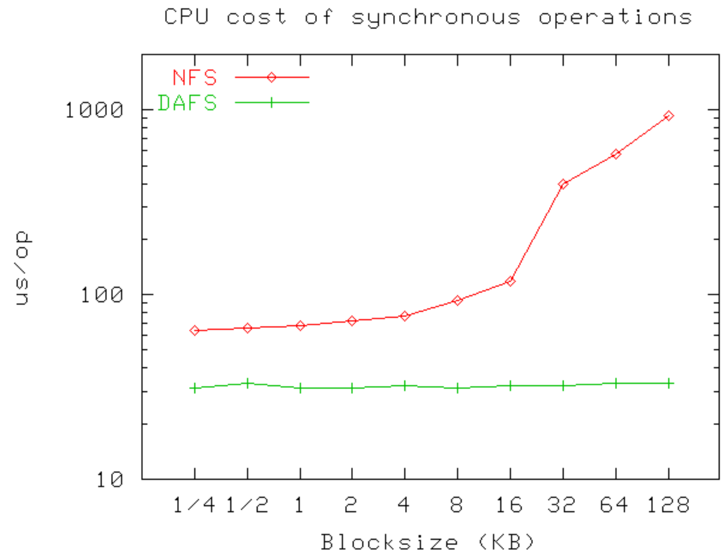
Microbenchmarks

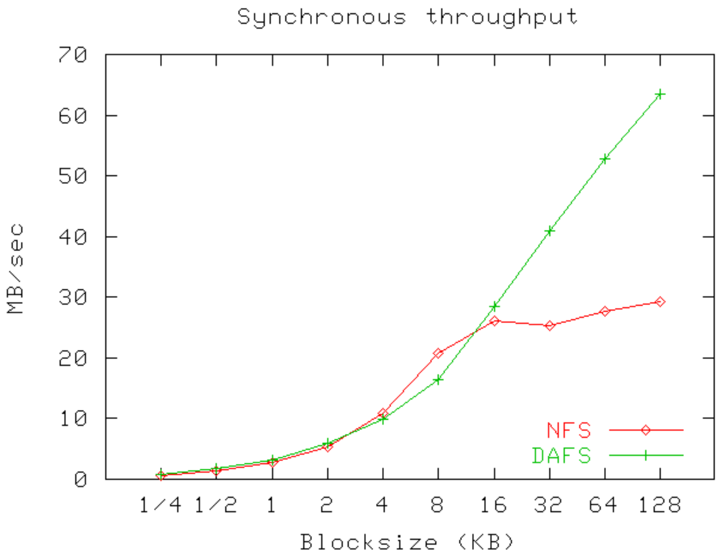
- Measures read performance
- NFS kernel versus DAFS user
- Asynchronous and Synchronous
- Throughput versus blocksize
- Throughput versus CPU time
- DAFS advantages are evident:
  - Increased throughput
  - Constant overhead per operation

Microbenchmark Results





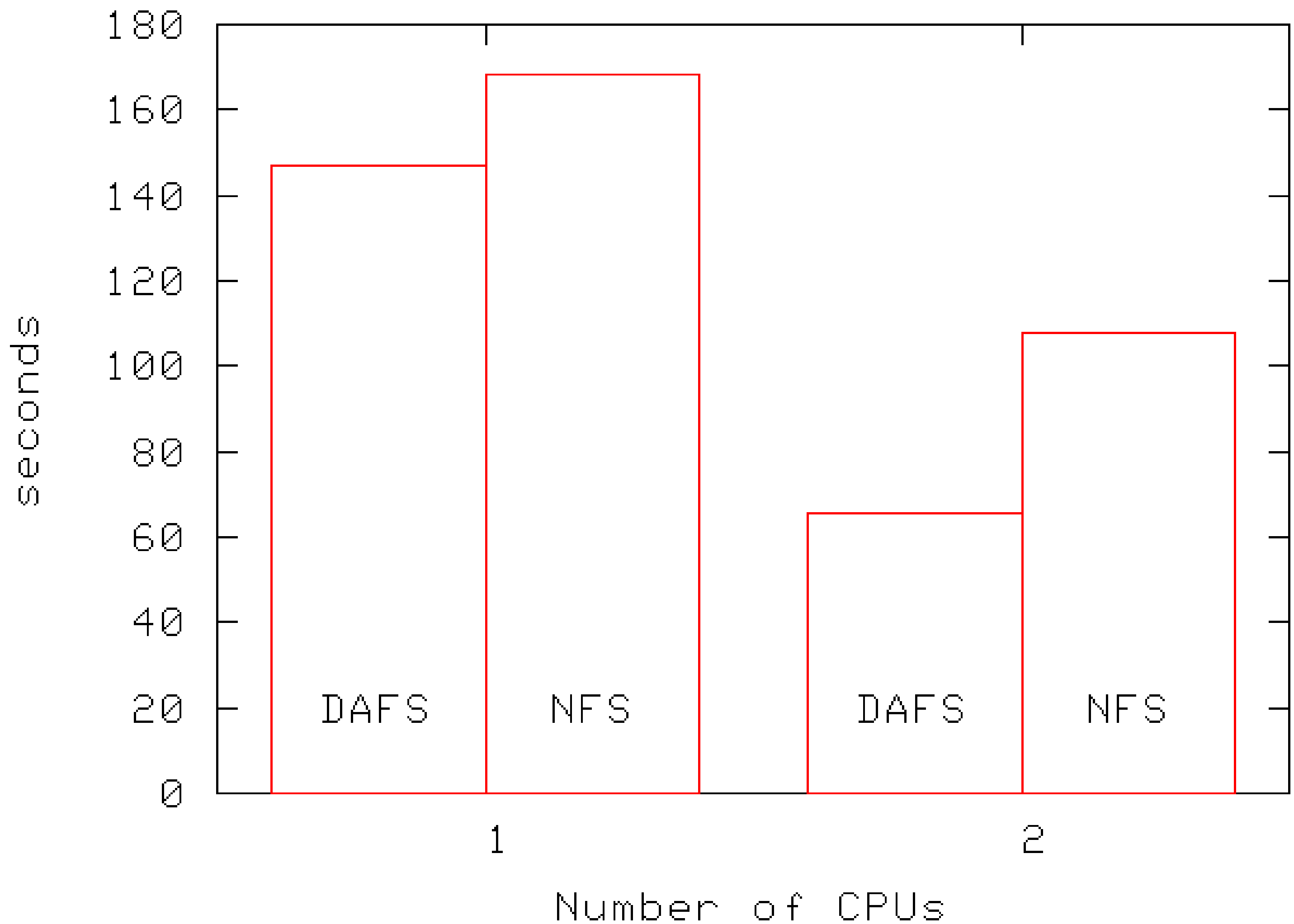




Application (GNU gzip)

- Demonstrates benefit of user I/O parallelism
- Read, compress, write 550MB file
- Gzip modified to use DAFS API
  - Memory preregistration, asynchronous read and write
- 16KB blocksize
- 1 CPU, 1 process: DAFS advantage
- 2 CPUs, 2 processes: DAFS 2x speedup

GNU gzip Runtimes



Conclusion

- DAFS protocol enables high-performance local file sharing
- DAFS API leverages benefit of user space I/O
- The combination yields significant performance gains for I/O intensive applications