Remote File System Suite

Softwarepraktikum für Fortgeschrittene

Michael Kuhn

Parallele und Verteilte Systeme
Institut für Informatik
Ruprecht-Karls-Universität Heidelberg

2009-02-05

1 Introduction
- Introduction

2 Remote File System Daemon

3 Remote File System Library

4 Global Remote File System

5 Evaluation

6 Conclusion and Future Work

FUSE

- Goal was to implement a global network file system
  - Needed to implement the underlying network file system first
- Should be implemented as a FUSE file system
  - Runs in user space
  - Relatively easy to implement
  - Relatively easy to maintain

Overview

- `rfsd` – Remote File System Daemon
  - Low-level network file system
- `librfs` – Remote File System Library
  - Abstracts protocol implementation
- `rfsc` – Remote File System Client
  - Basically a simple throughput and metadata benchmark
- `grfs` – Global Remote File System
  - High-level global network file system

1 Introduction

2 Remote File System Daemon
- Motivation
- Overview
- Implementation

3 Remote File System Library

4 Global Remote File System

5 Evaluation

6 Conclusion and Future Work

- A separate protocol was designed
- Existing protocols did not meet the requirements
- SSH
  - Does not support separate control and data channels
  - Data encryption makes transfers too slow
    - Not possible to deactivate the encryption
- FTP
  - Only possible to write a complete file or append data to it
  - File listings are hard to parse, because their format is not well-defined

- Implement our own protocol
- Separate control and data channels
  - No encryption
  - Control channel can be encrypted via SSH forwarding
- Should be as fast as possible
  - Microscope pumps out 1 GB/s
  - $6 \cdot 2$ servers
  - $\Rightarrow$ 100-200 MB/s
- Should be as transparent as possible
  - Use underlying local file system
  - Do not stripe files across servers
- Should be as safe as possible
  - Support replication

- Basically provide remote access to the local file system
  - Protocol very similar to POSIX
    - `pread()`, `pwrite()`, ...
  - Plus some fancy features, of course :-)
- Fully multi-threaded
  - Each connection handled in its own thread
  - Long-running operations do not block other connections
- Background replication
  - Master-slave concept
  - One master, multiple slaves
  - All operations are replicated in a background thread
  - Write operations are barriers
    - We do not need to allocate additional memory for background replication
    - We do not need to read from the file to preserve memory (race conditions)

1 Introduction

2 Remote File System Daemon

3 Remote File System Library
- Motivation

4 Global Remote File System

5 Evaluation

6 Conclusion and Future Work

- Hide all the “ugly” implementation details :-)
- Good error reporting via GError
  - Part of GLib
- Some operations require multiple steps
  - For example: `rfs_read()`, `rfs_read_do()`, `rfs_read_end()`

1 Introduction

2 Remote File System Daemon

3 Remote File System Library

4 Global Remote File System
- Overview

5 Evaluation

6 Conclusion and Future Work

- Merge multiple file systems into one global namespace
- Example:
  - `serv1` has directory `/foo`, `serv2` has directory `/bar`
  - `$ grfs serv1:6666 serv2:6666 /grfs`
  - `$ ls /grfs`
  - `> foo bar`

1 Introduction

2 Remote File System Daemon

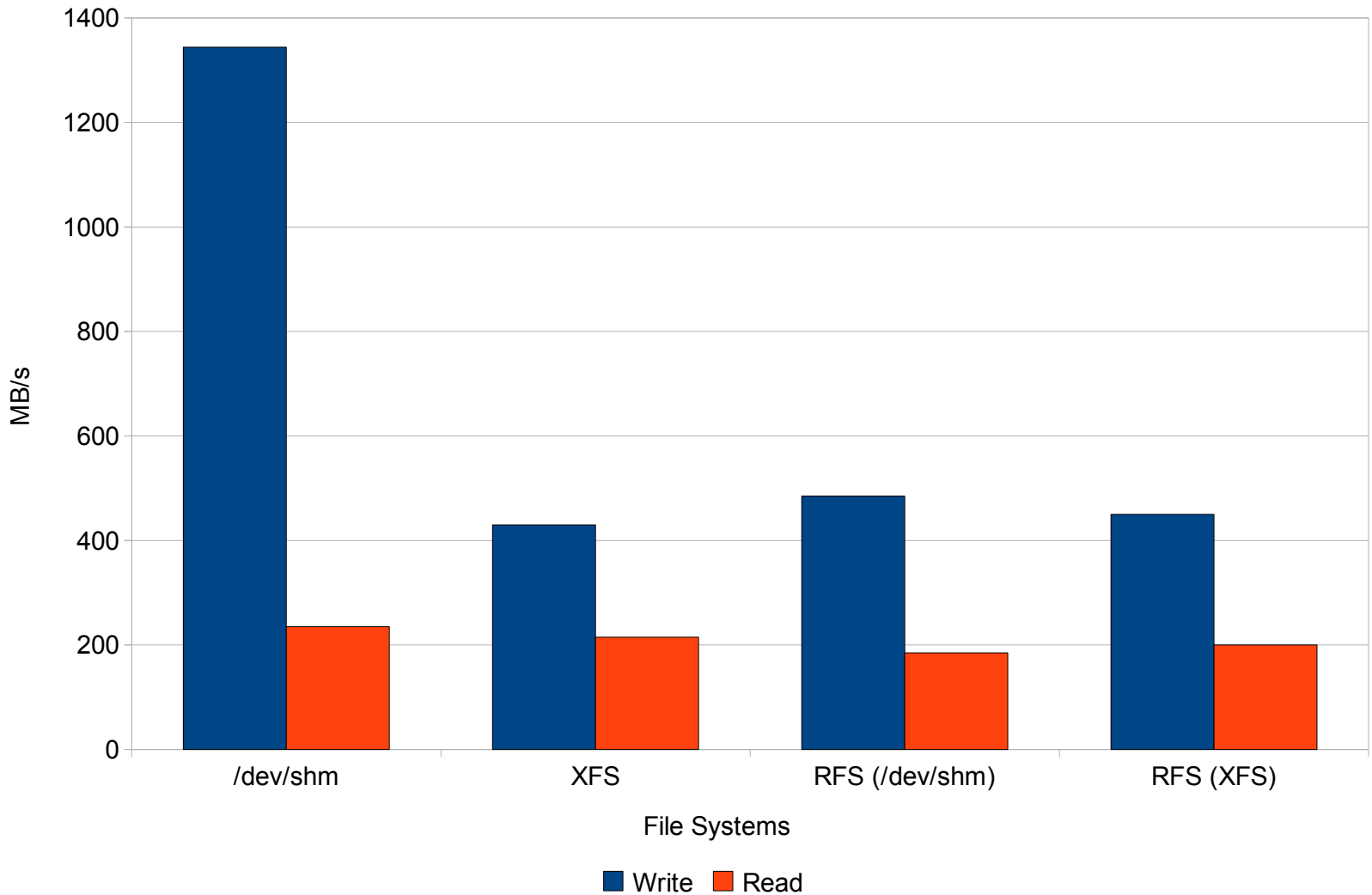
3 Remote File System Library

4 Global Remote File System

5 Evaluation
- Evaluation

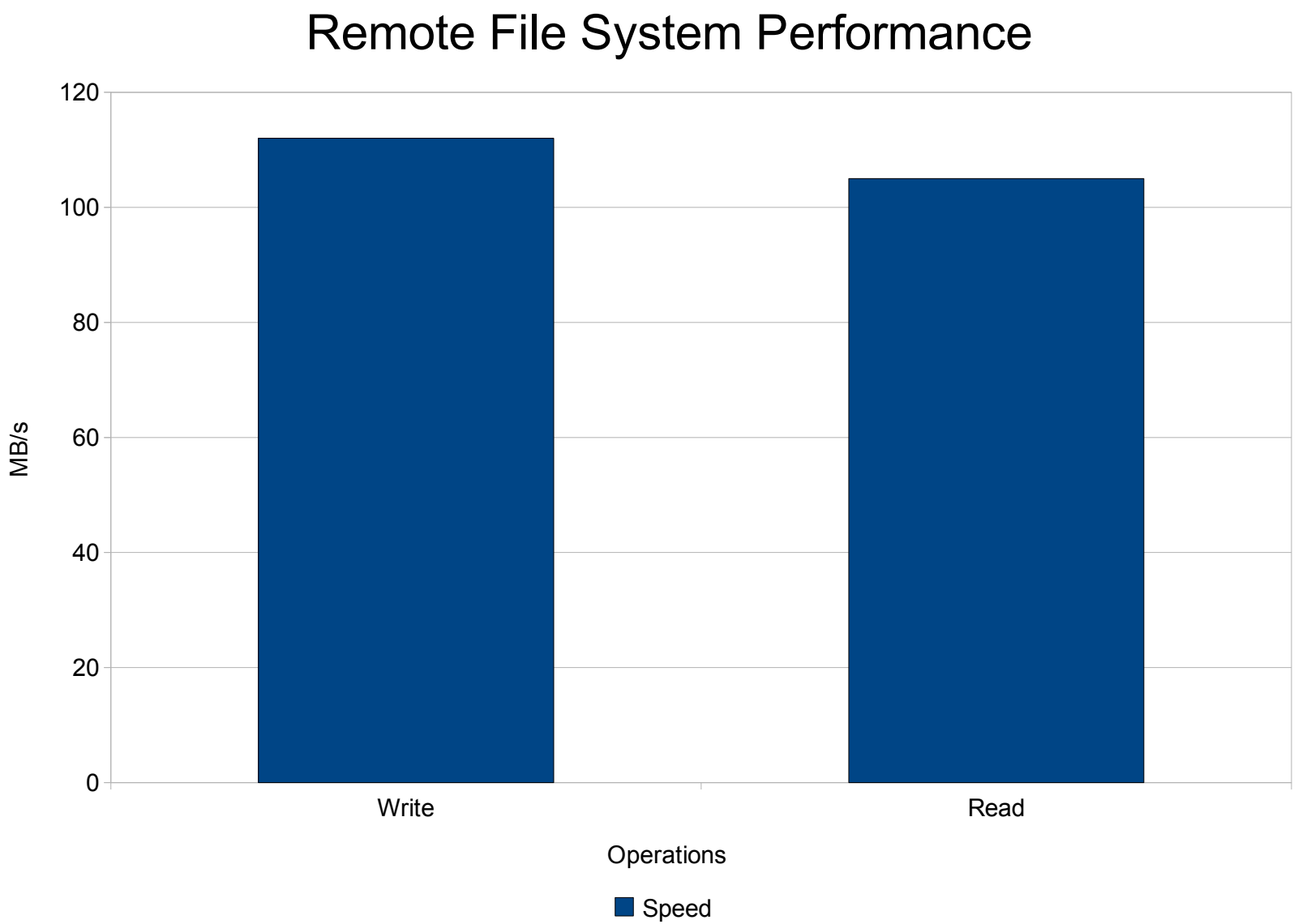
6 Conclusion and Future Work

- The next benchmark is local
  - That is, client and server were started on the same machine

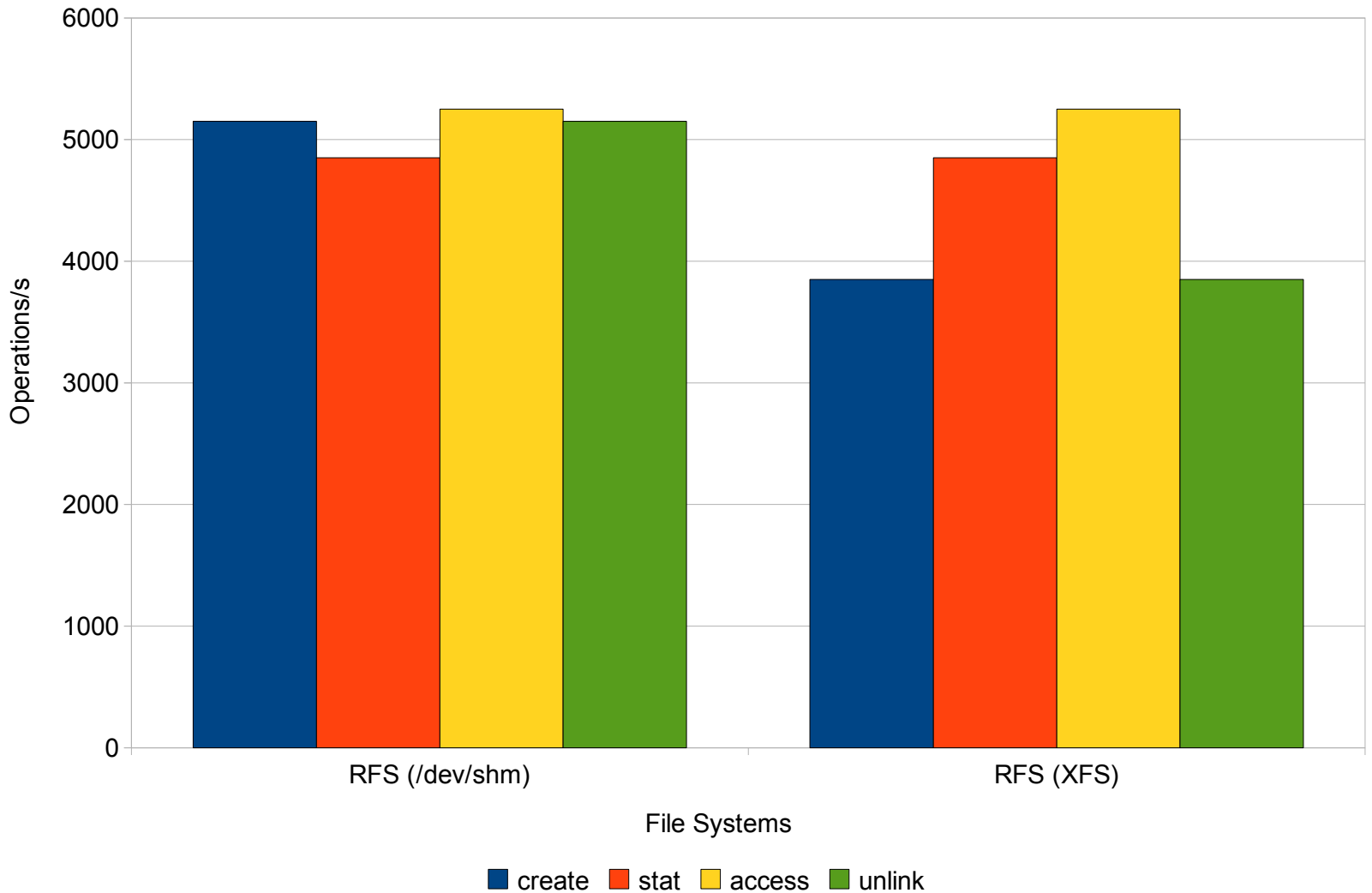


Read: Quite slow — Write: `RFS (XFS)` > `XFS` – `dd` vs. `rfsc`?

- The next two benchmarks are remote
  - Client and server were started on two separate machines
- We have a GBit network
  - About 119 MB/s maximum throughput

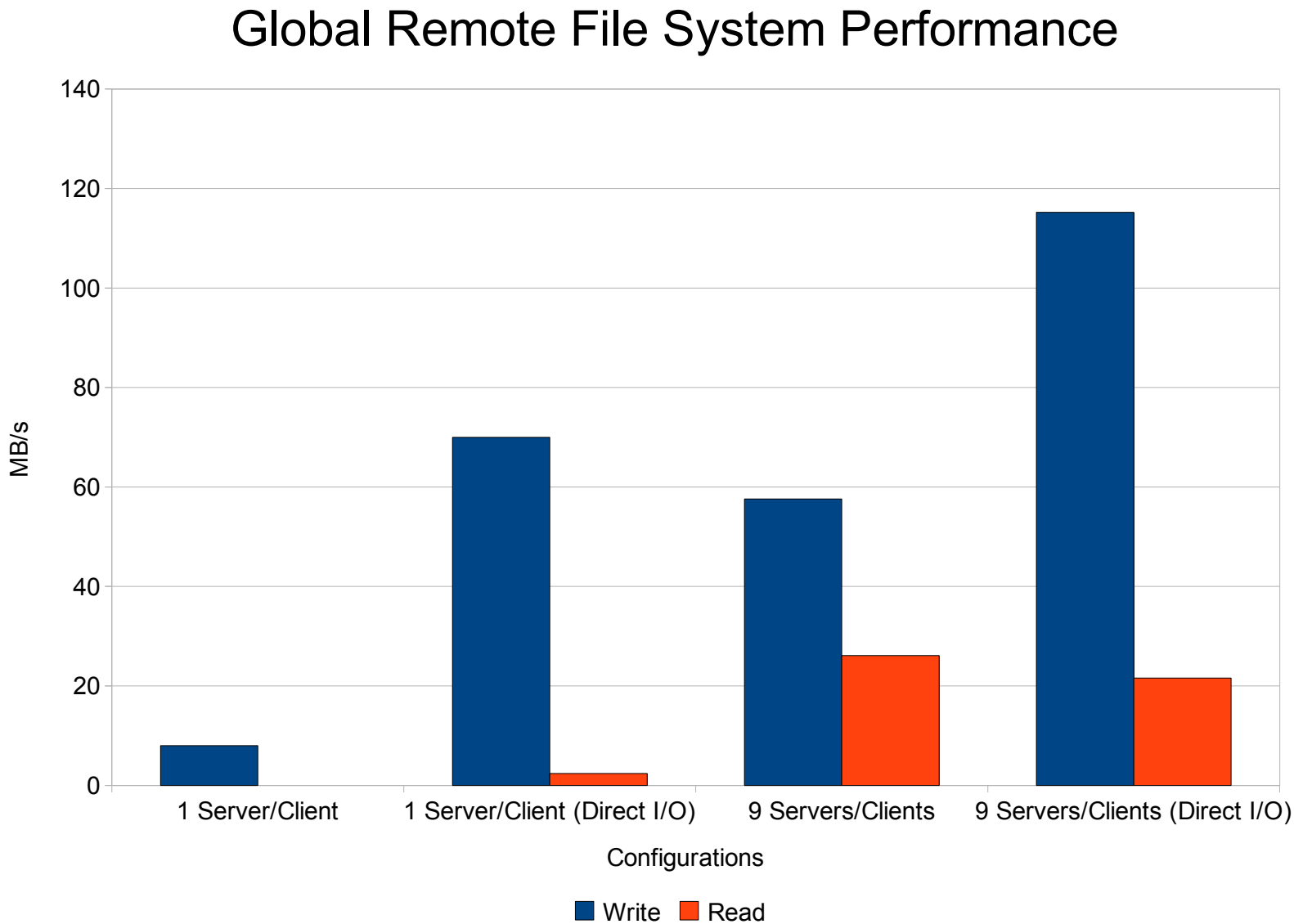


Write: Almost network maximum – Read: Slight overhead



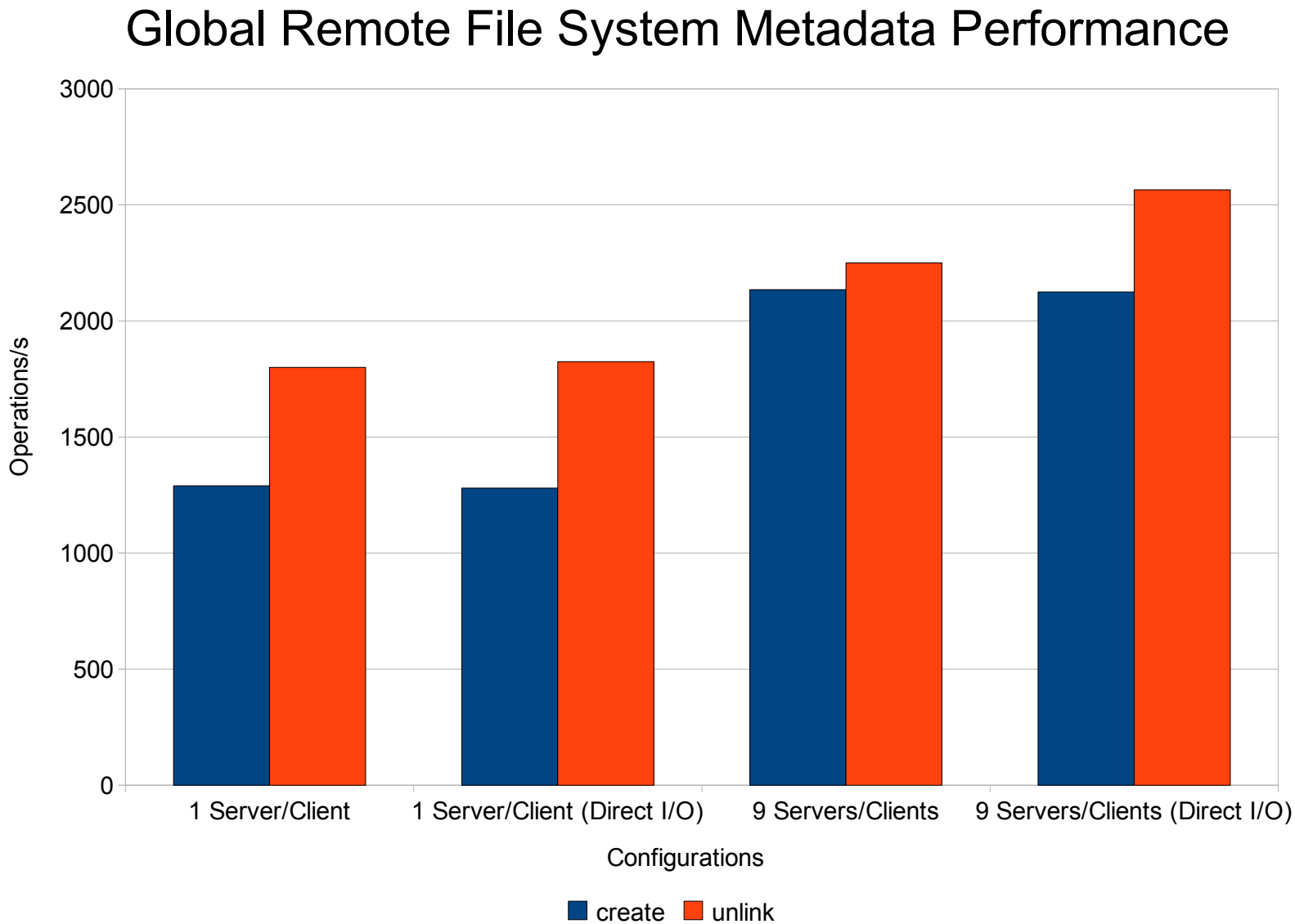
`stat` and `access`: Same on `/dev/shm` and `XFS` (no writes)

- The next two benchmarks are remote
  - Clients and servers were started on separate machines
  - All clients were started on the **same** machine
- We have a GBit network
  - About 119 MB/s maximum throughput



Read: 2 GB/s from kernel cache (screws up diagram)

- These numbers are quite bad . . .
- New performance ~~hack~~ improvement in `grfs`
  - No new diagrams, it was too late :-)
- 1 server/client
  - Normal: 56 MB/s (Write) – 2 GB/s (Read)
  - Direct I/O: 112 MB/s – 19 MB/s (Read)
- Remaining problems
  - Normal: Fix write
  - Direct I/O: Fix read
  - Increasing FUSE's buffer size would suffice
    - Too much overhead for 4 or 16 KB buffers
    - 128 KB work well
  - Does not work – FUSE bug?


Global Remote File System Metadata Performance
Operations/s
3000
2500
2000
1500
1000
500
0
1 Server/Client
1 Server/Client (Direct I/O)
9 Servers/Clients
9 Servers/Clients (Direct I/O)
Configurations
create
unlink

1 Introduction

2 Remote File System Daemon

3 Remote File System Library

4 Global Remote File System

5 Evaluation

6 Conclusion and Future Work
- Future Work

- Server-side replication ✓
  - The client – that is, the Global Remote File System – used to do this
- Support for High Availability ✓
  - Global Remote File System should continue working if servers go offline
  - Simply use remaining servers, providing a partial view of the global file system
  - Reconnect on `SIGHUP`
- Synchronize multiple Remote File System Daemons
  - Unique IDs for all modifying operations
- Make `grfs` usable for all users
  - Check FUSE's `allow_other` and `default_permissions`
- Optimize the Global Remote File System/FUSE
- Container support (?)