vNFS: Maximizing NFS Performance with Compounds and Vectorized I/O

Ming Chen, Dean Hildebrand,[×] Henry Nelson,[⅄]
Jasmit Saluja, Ashok Sankar Harihara Subramony, and Erez Zadok

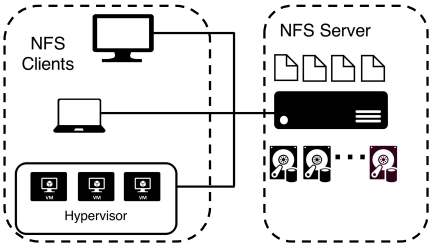
Stony Brook University, [×]IBM Research – Almaden, [⅄]Ward Melville High School

https://github.com/sbu-fsl/txn-compound

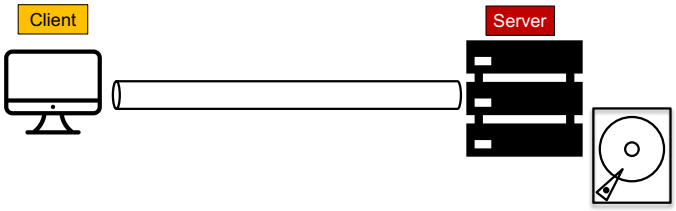
© 2017 Stony Brook University

Network File System (NFS)

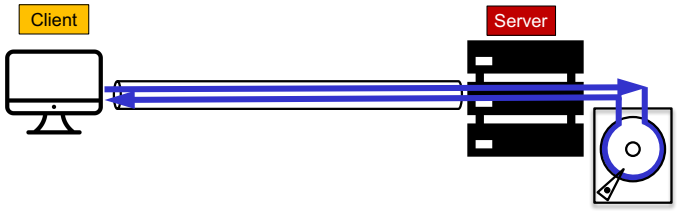
- An IETF standardized storage protocol
- Provides transparent remote file access
- Needs to overcome the high network latency



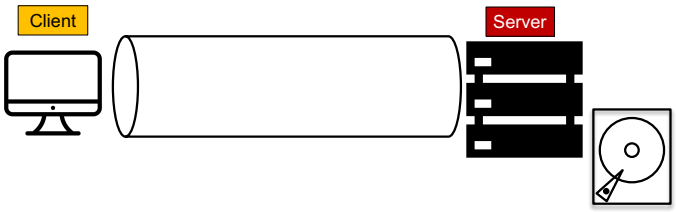
NFSv4 Compound Procedures



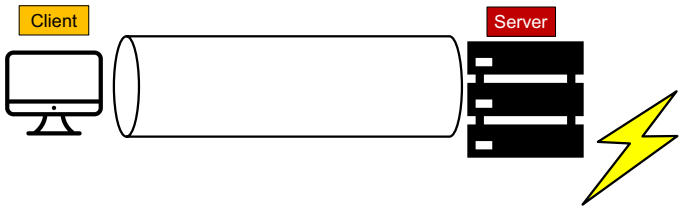
NFSv4 Compound Procedures



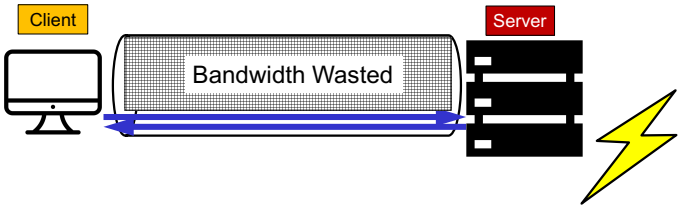
NFSv4 Compound Procedures



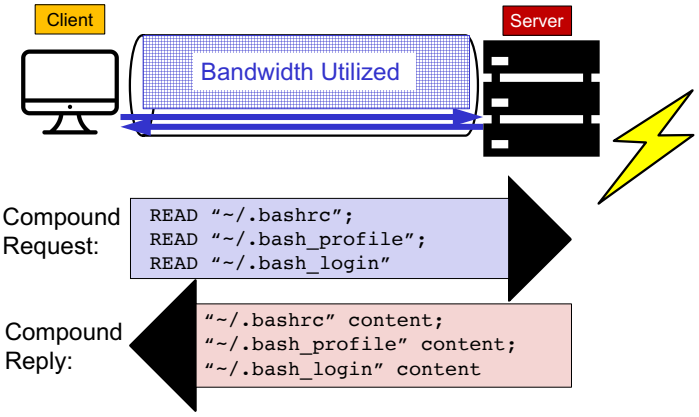
NFSv4 Compound Procedures



NFSv4 Compound Procedures



NFSv4 Compound Procedures



Compound Request:

```
READ "~/.bashrc";
READ "~/.bash_profile";
READ "~/.bash_login"
```

Compound Reply:

```
"~/.bashrc" content;
"~/.bash_profile" content;
"~/.bash_login" content
```


Compounds Are Underused!

- Limited by the synchronous POSIX file-system API
  - Example: reading "/home/Bob/.bashrc"

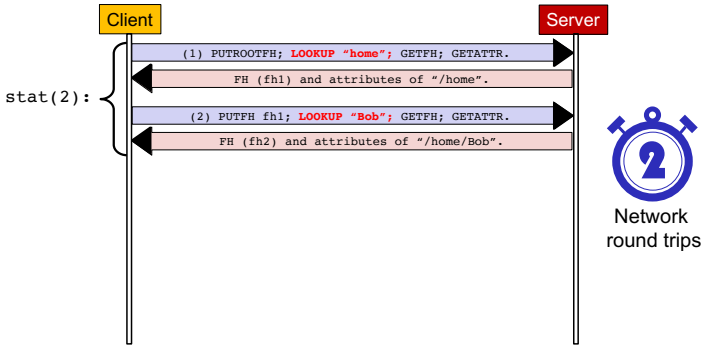
Compounds Are Underused!

- Limited by the synchronous POSIX file-system API
  - Example: reading "/home/Bob/.bashrc"



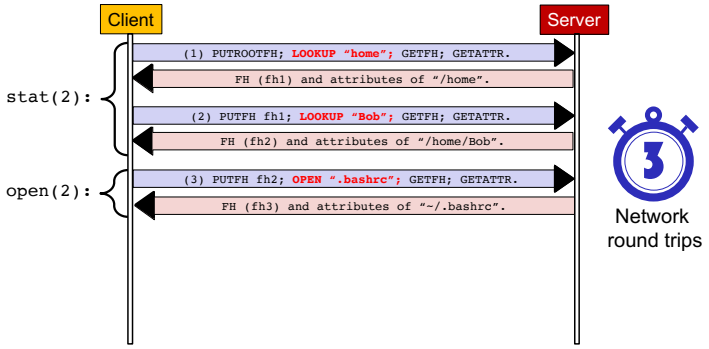
Compounds Are Underused!

- Limited by the synchronous POSIX file-system API
  - Example: reading "/home/Bob/.bashrc"



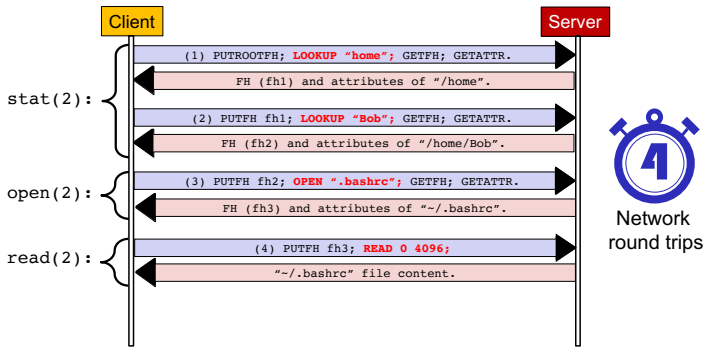
Compounds Are Underused!

- Limited by the synchronous POSIX file-system API
  - Example: reading "/home/Bob/.bashrc"



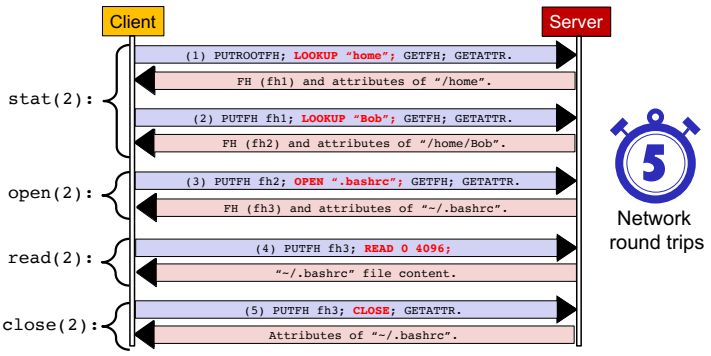
Compounds Are Underused!

- Limited by the synchronous POSIX file-system API
  - Example: reading "/home/Bob/.bashrc"



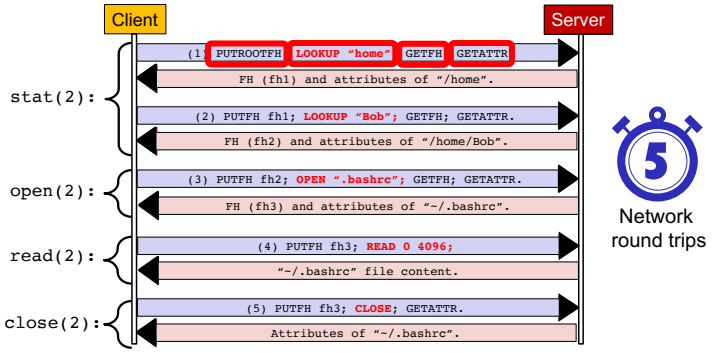
Compounds Are Underused!

- Limited by the synchronous POSIX file-system API
  - Example: reading "/home/Bob/.bashrc"



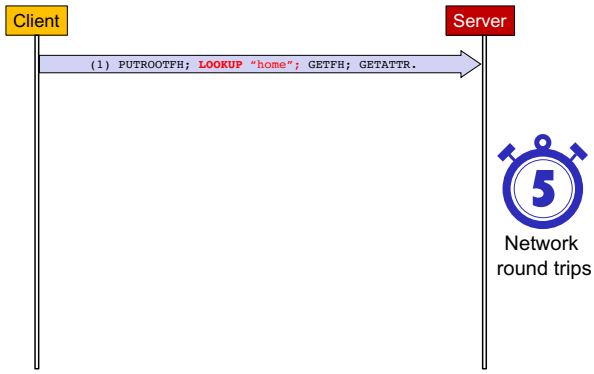
Compounds Are Underused!

- Limited by the synchronous POSIX file-system API
  - Example: reading "/home/Bob/.bashrc"



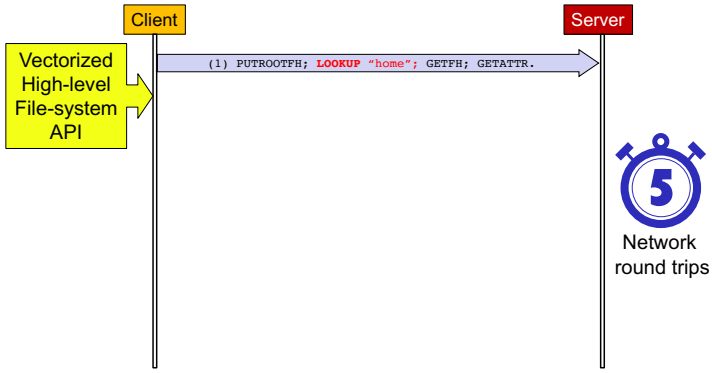
Need a Batching File-System API

- Target: open, read, and close multiple files in one RPC



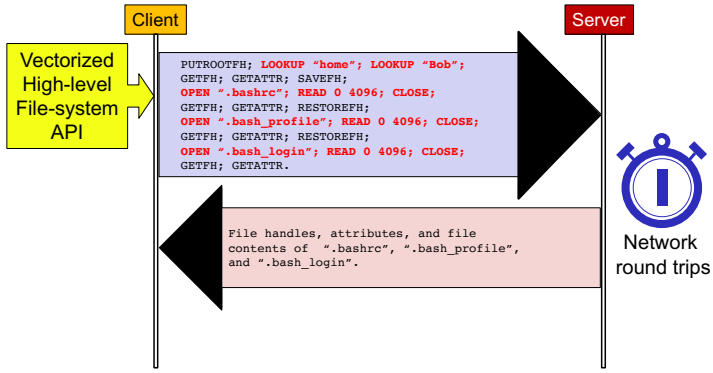
Need a Batching File-System API

- Target: open, read, and close multiple files in one RPC



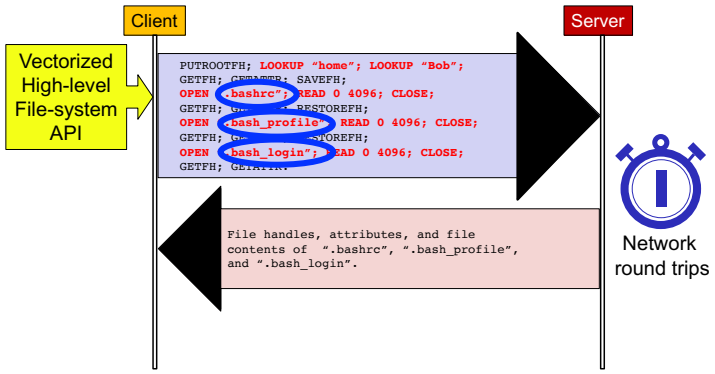
Need a Batching File-System API

- Target: open, read, and close multiple files in one RPC



Need a Batching File-System API

- Target: open, read, and close multiple files in one RPC



vread/vwrite

- Read/write many files
  - Unlike `readv/writev(2)`: many (non-contiguous) offsets of many files
  - Append to file when write offset is `UINT64_MAX`
- Automatic file opening/closing
  - Pass states using "*current filehandle*" and "*current stateid*"

```
struct vio {
    struct vfile vfile; // [IN]: a file identified by path or descriptor
    uint64_t offset;    // [IN]: offset of read/write, UINT64_MAX means append
    uint64_t length;    // [IN]: bytes to read/write; [OUT]: bytes read/written
    const char *data;   // [IN]: data to write; [OUT]: buffer for read
    uint32_t flags;     // [IN] flags: is_creation, is_write_stable
};                      // [OUT] flags: is_eof, is_write_stable
struct vres vread(struct vio *ios, int n);
struct vres vwrite(struct vio *ios, int n);
```


vread/vwrite

- Read/write many files
  - Unlike `readv/writev(2)`: many (non-contiguous) offsets of many files
  - Append to file when write offset is `UINT64_MAX`
- Automatic file opening/closing
  - Pass states using "*current filehandle*" and "*current stateid*"

```
struct vio {
    struct vfile vfile; // [IN]: a file identified by path or descriptor
    uint64_t offset;    // [IN]: offset of read/write, UINT64_MAX means append
    uint64_t length;    // [IN]: bytes to read/write; [OUT]: bytes read/written
    const char *data;   // [IN]: data to write; [OUT]: buffer for read
    uint32_t flags;     // [IN] flags: is_creation, is_write_stable
};                      // [OUT] flags: is_eof, is_write_stable
struct vres vread(struct vio *ios, int n);
struct vres vwrite(struct vio *ios, int n);
```


vread/vwrite

- Read/write many files
  - Unlike `readv/writev(2)`: many (non-contiguous) offsets of many files
  - Append to file when write offset is `UINT64_MAX`
- Automatic file opening/closing
  - Pass states using "*current filehandle*" and "*current stateid*"

```
struct vio {
    struct vfile vfile; // [IN]: a file identified by path or descriptor
    uint64_t offset;    // [IN]: offset of read/write, UINT64_MAX means append
    uint64_t length;    // [IN]: bytes to read/write; [OUT]: bytes read/written
    const char *data;   // [IN]: data to write; [OUT]: buffer for read
    uint32_t flags;     // [IN] flags: is_creation, is_write_stable
};                      // [OUT] flags: is_eof, is_write_stable
struct vres vread(struct vio *ios, int n);
struct vres vwrite(struct vio *ios, int n);
```


vread/vwrite

- Read/write many files
  - Unlike `readv/writev(2)`: many (non-contiguous) offsets of many files
  - Append to file when write offset is `UINT64_MAX`
- Automatic file opening/closing
  - Pass states using "*current filehandle*" and "*current stateid*"

```
struct vio {
    struct vfile vfile; // [IN]: a file identified by path or descriptor
    uint64_t offset;    // [IN]: offset of read/write, UINT64_MAX means append
    uint64_t length;    // [IN]: bytes to read/write; [OUT]: bytes read/written
    const char *data;   // [IN]: data to write; [OUT]: buffer for read
    uint32_t flags;     // [IN] flags: is_creation, is_write_stable
};                      // [OUT] flags: is_eof, is_write_stable
struct vres vread(struct vio *ios, int n);
struct vres vwrite(struct vio *ios, int n);
```


vgetattrs/vsetattrs

- `vgetattrs`: get attributes of files
- `vsetattrs`: set attributes of files
  - Useful for copying attributes of files, e.g., `tar` and `rsync`
  - Combines `chmod`, `chown`, `utimes`, `truncate`
  - Linux kernel uses `inode_operations->setattr`
  - NFSv4 uses the `SETATTR` operation

vgetattrs/vsetattrs

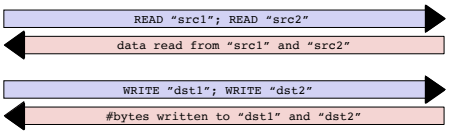
- vgetattrs: get attributes of files
- vsetattrs: set attributes of files
  - Useful for copying attributes of files, e.g., tar and rsync
  - Combines chmod, chown, utimes, truncate
  - Linux kernel uses inode_operations->setattr
  - NFSv4 uses the SETATTR operation

vgetattrs/vsetattrs

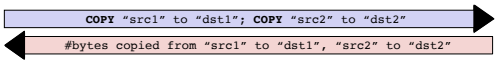
- vgetattrs: get attributes of files
- vsetattrs: set attributes of files
  - Useful for copying attributes of files, e.g., tar and rsync
  - Combines chmod, chown, utimes, truncate
  - Linux kernel uses inode_operations->setattr
  - NFSv4 uses the SETATTR operation

vcopy/vsscopy

- Copy many files partly or entirely
  - Create destination files if necessary
- vcopy

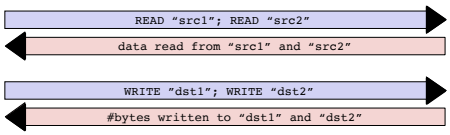


- vsscopy (Server Side Copy in NFSv4.2)

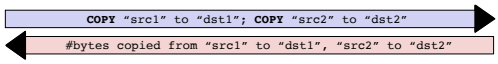


vcopy/vsscopy

- Copy many files partly or entirely
  - Create destination files if necessary
- vcopy

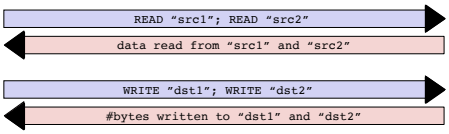


- vsscopy (Server Side Copy in NFSv4.2)

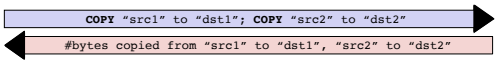


vcopy/vsscopy

- Copy many files partly or entirely
  - Create destination files if necessary
- vcopy

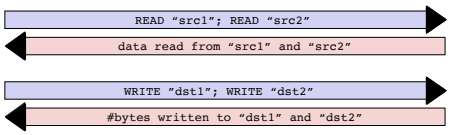


- vsscopy (Server Side Copy in NFSv4.2)

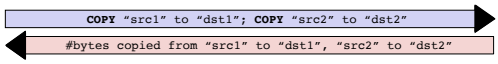


vcopy/vsscopy

- Copy many files partly or entirely
  - Create destination files if necessary
- vcopy

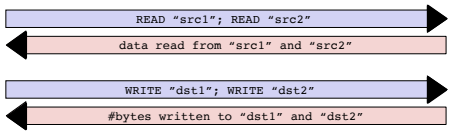


- vsscopy (Server Side Copy in NFSv4.2)

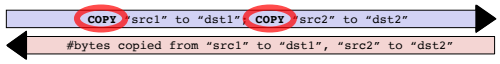


vcopy/vsscopy

- Copy many files partly or entirely
  - Create destination files if necessary
- vcopy



- vsscopy (Server Side Copy in NFSv4.2)



Other Operations

- vopen/vclose
  - Large files
  - Maintain close-to-open consistency
- vsymlink/vreadlink/vhardlink
  - Example: create a symlink tree: "cp -sr"
- vmkdir/vlistdir
  - Example: create "/a", "/a/b", and "/a/b/c"
  - vlistdir: list multiple directories (recursively)
- vremove
- vrename

Other Operations

- vopen/vclose
  - Large files
  - Maintain close-to-open consistency
- vsymlink/vreadlink/vhardlink
  - Example: create a symlink tree: "cp -sr"
- vmkdir/vlistdir
  - Example: create "/a", "/a/b", and "/a/b/c"
  - vlistdir: list multiple directories (recursively)
- vremove
- vrename

Other Operations

- vopen/vclose
  - Large files
  - Maintain close-to-open consistency
- vsymlink/vreadlink/vhardlink
  - Example: create a symlink tree: "cp -sr"
- vmkdir/vlistdir
  - Example: create "/a", "/a/b", and "/a/b/c"
  - vlistdir: list multiple directories (recursively)
- vremove
- vrename

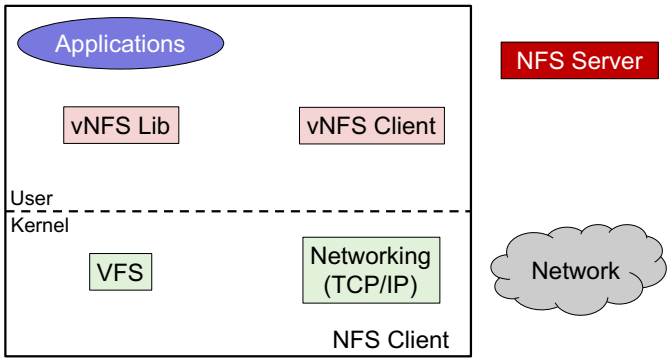
Other Operations

- vopen/vclose
  - Large files
  - Maintain close-to-open consistency
- vsymlink/vreadlink/vhardlink
  - Example: create a symlink tree: "cp -sr"
- vmkdir/vlistdir
  - Example: create "/a", "/a/b", and "/a/b/c"
  - vlistdir: list multiple directories (recursively)
- vremove
- vrename

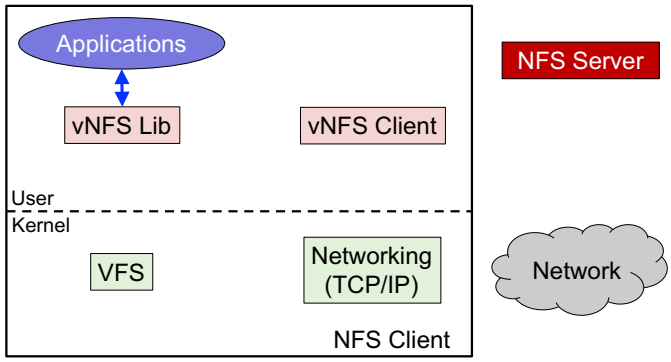
Other Operations

- vopen/vclose
  - Large files
  - Maintain close-to-open consistency
- vsymlink/vreadlink/vhardlink
  - Example: create a symlink tree: "cp -sr"
- vmkdir/vlistdir
  - Example: create "/a", "/a/b", and "/a/b/c"
  - vlistdir: list multiple directories (recursively)
- vremove
- vrename

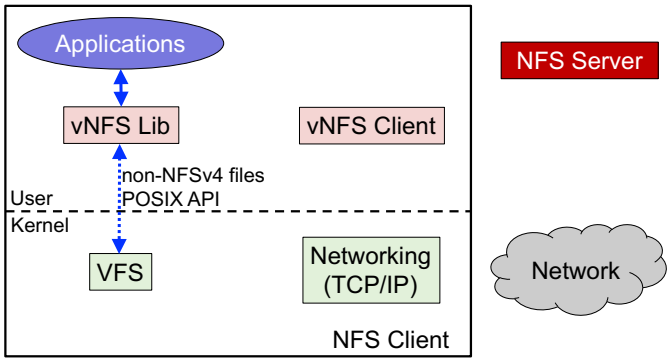
Architecture



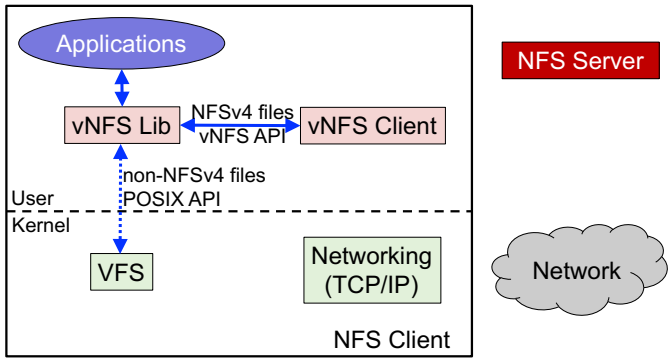
Architecture



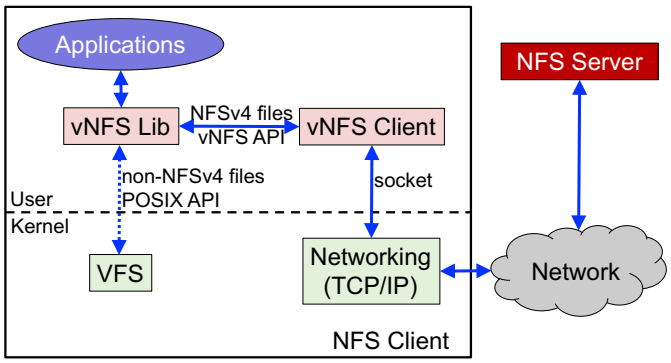
Architecture



Architecture



Architecture



Implementation

- NFS-Ganesha
  - An open-source user-space NFS server
  - File-System Abstraction Layer (FSAL) that is similar to VFS
- Client side
  - vNFS client based on NFS-Ganesha Proxy FSAL (NFSv4.1)
  - vNFS library
  - No client-side cache yet
- Server side
  - NFS-Ganesha VFS FSAL
  - Server Side Copy & atomic file appending
- Code
  - C/C++: added 10,632 to NFS-Ganesha; deleted 1,407
  - https://github.com/sbu-fsl/txn-compound

Implementation

- NFS-Ganesha
  - An open-source user-space NFS server
  - File-System Abstraction Layer (FSAL) that is similar to VFS
- Client side
  - vNFS client based on NFS-Ganesha Proxy FSAL (NFSv4.1)
  - vNFS library
  - No client-side cache yet
- Server side
  - NFS-Ganesha VFS FSAL
  - Server Side Copy & atomic file appending
- Code
  - C/C++: added 10,632 to NFS-Ganesha; deleted 1,407
  - https://github.com/sbu-fsl/txn-compound

Implementation

- NFS-Ganesha
  - An open-source user-space NFS server
  - File-System Abstraction Layer (FSAL) that is similar to VFS
- Client side
  - vNFS client based on NFS-Ganesha Proxy FSAL (NFSv4.1)
  - vNFS library
  - No client-side cache yet
- Server side
  - NFS-Ganesha VFS FSAL
  - Server Side Copy & atomic file appending
- Code
  - C/C++: added 10,632 to NFS-Ganesha; deleted 1,407
  - https://github.com/sbu-fsl/txn-compound

Implementation

- NFS-Ganesha
  - An open-source user-space NFS server
  - File-System Abstraction Layer (FSAL) that is similar to VFS
- Client side
  - vNFS client based on NFS-Ganesha Proxy FSAL (NFSv4.1)
  - vNFS library
  - No client-side cache yet
- Server side
  - NFS-Ganesha VFS FSAL
  - Server Side Copy & atomic file appending
- Code
  - C/C++: added 10,632 to NFS-Ganesha; deleted 1,407
  - https://github.com/sbu-fsl/txn-compound

Evaluation

- Experimental setup
  - Two six-core machines with 64GB RAM and a 10GbE NIC
  - Running CentOS 7 with 3.14 kernel
  - Intel S3700 200GB SSD
  - Network latency of 0.2ms
  - Use `netem` to emulate different networks
  - Baseline: Linux in-kernel NFSv4.1 client
- Benchmarks & application porting
  - Micro-benchmarks
  - GNU Coreutils (cp, ls, rm)
  - Tar/Untar
  - Filebench
  - HTTP/2 server (nghttp2)

Evaluation

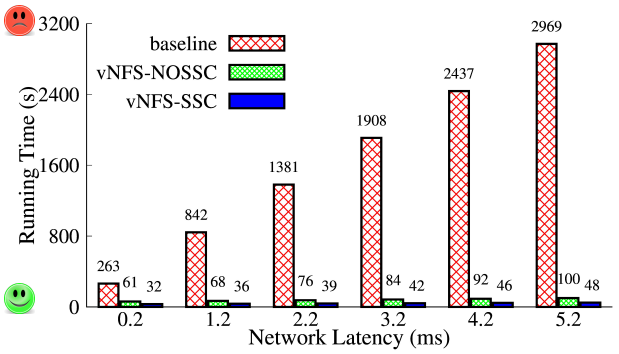
- Experimental setup
  - Two six-core machines with 64GB RAM and a 10GbE NIC
  - Running CentOS 7 with 3.14 kernel
  - Intel S3700 200GB SSD
  - Network latency of 0.2ms
  - Use `netem` to emulate different networks
  - Baseline: Linux in-kernel NFSv4.1 client
- Benchmarks & application porting
  - Micro-benchmarks
  - GNU Coreutils (cp, ls, rm)
  - Tar/Untar
  - Filebench
  - HTTP/2 server (nghttp2)

Evaluation

- Experimental setup
  - Two six-core machines with 64GB RAM and a 10GbE NIC
  - Running CentOS 7 with 3.14 kernel
  - Intel S3700 200GB SSD
  - Network latency of 0.2ms
  - Use `netem` to emulate different networks
  - Baseline: Linux in-kernel NFSv4.1 client
- Benchmarks & application porting
  - Micro-benchmarks
  - GNU Coreutils (cp, ls, rm)
  - Tar/Untar
  - Filebench
  - HTTP/2 server (nghttp2)

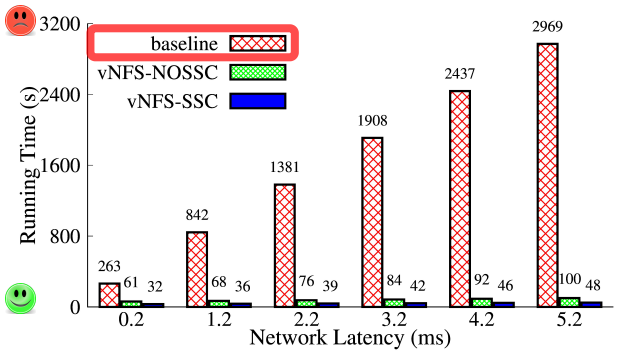
GNU Coreutils (cp)

- Copy the Linux source tree (`cp -r`)



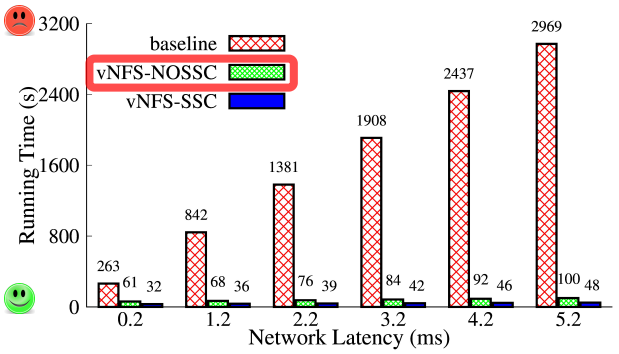
GNU Coreutils (cp)

- Copy the Linux source tree (`cp -r`)



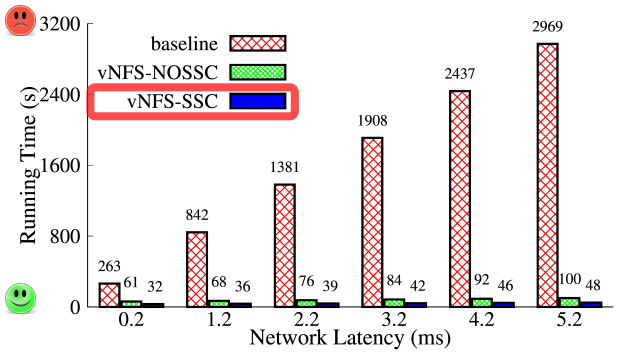
GNU Coreutils (cp)

- Copy the Linux source tree (`cp -r`)



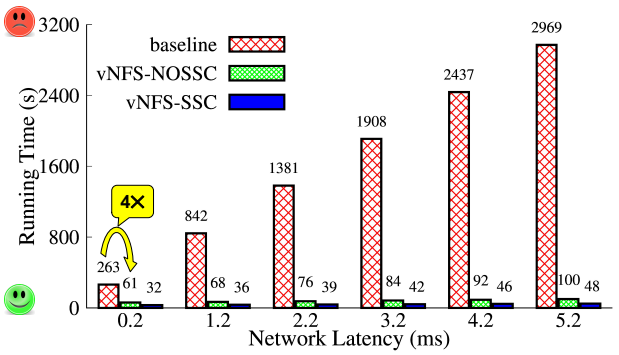
GNU Coreutils (cp)

- Copy the Linux source tree (`cp -r`)



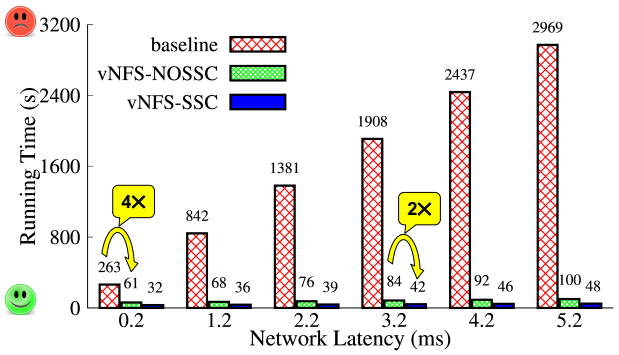
GNU Coreutils (cp)

- Copy the Linux source tree (`cp -r`)



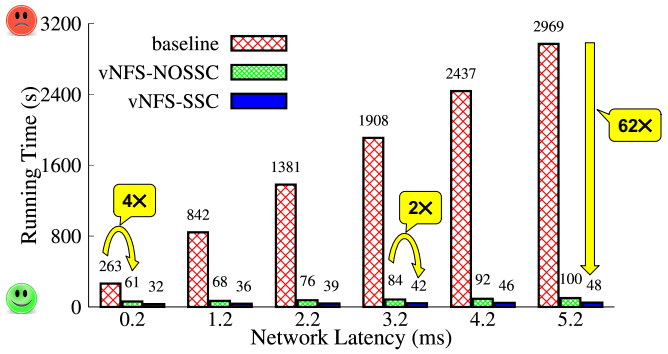
GNU Coreutils (cp)

- Copy the Linux source tree (`cp -r`)

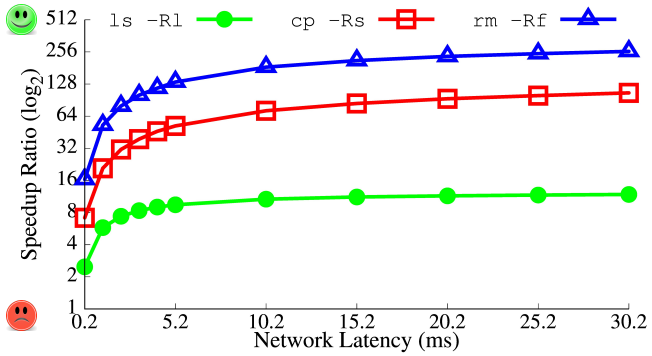


GNU Coreutils (cp)

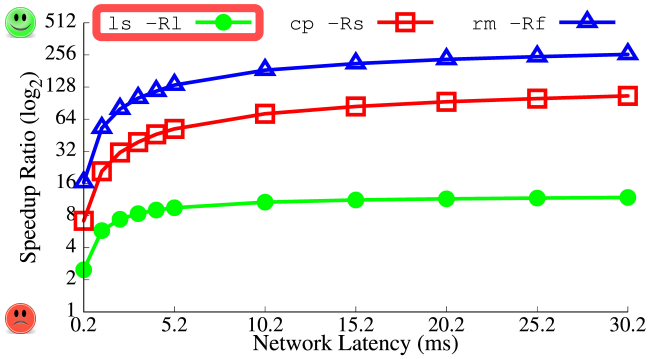
- Copy the Linux source tree (`cp -r`)



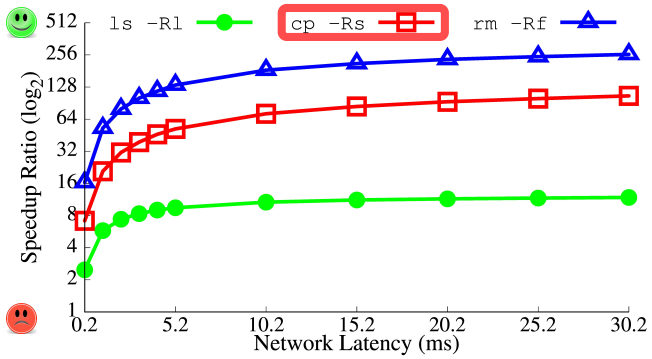
GNU Coreutils (ls, cp, rm)



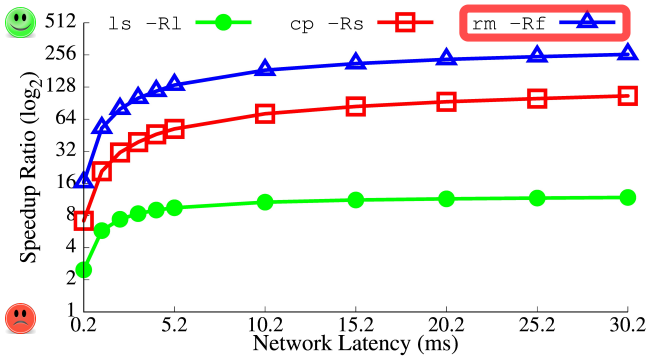
GNU Coreutils (ls, cp, rm)



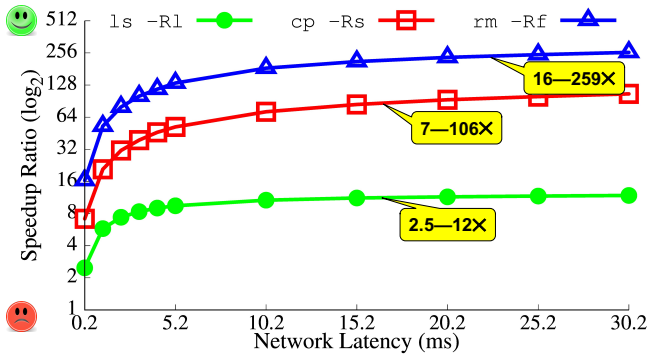
GNU Coreutils (ls, cp, rm)



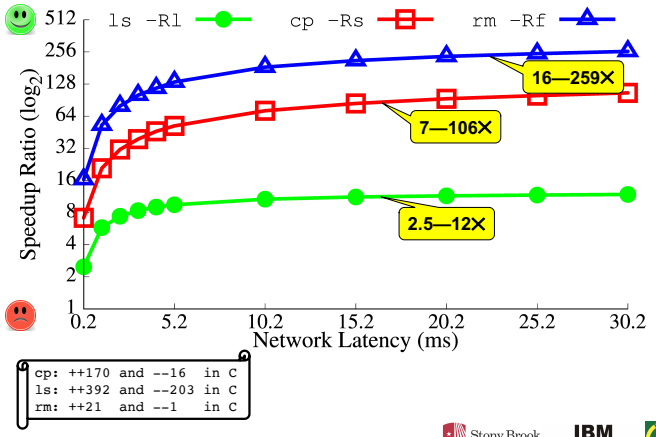
GNU Coreutils (ls, cp, rm)



GNU Coreutils (ls, cp, rm)

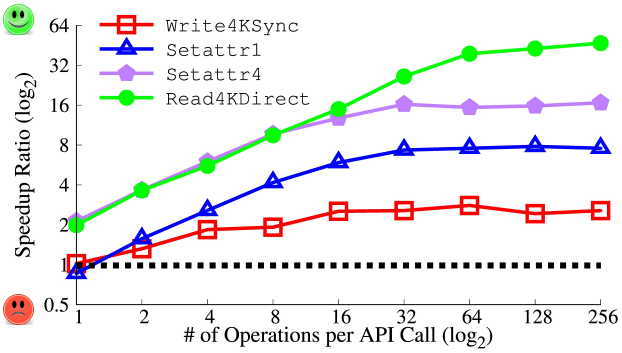


GNU Coreutils (ls, cp, rm)



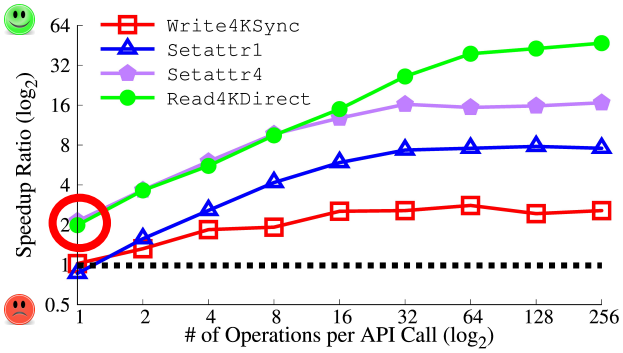

```
cp: ++170 and --16  in C
ls: ++392 and --203 in C
rm: ++21  and --1   in C
```


Compounding Degree



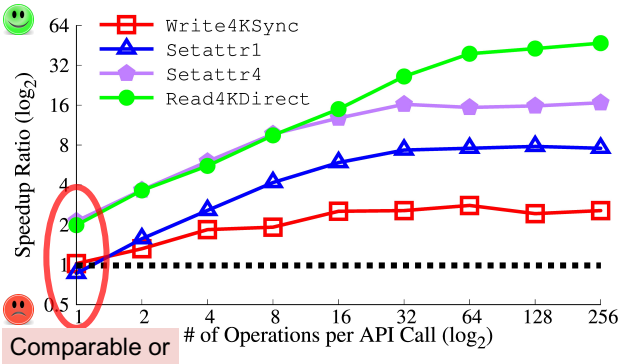
Network Latency: 0.2ms

Compounding Degree



Network Latency: 0.2ms

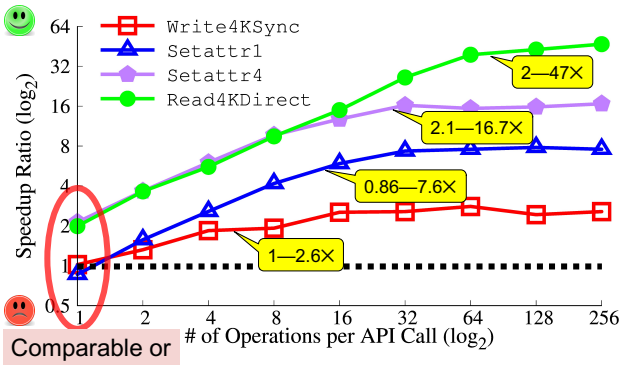
Compounding Degree



Comparable or Slightly better

Network Latency: 0.2ms

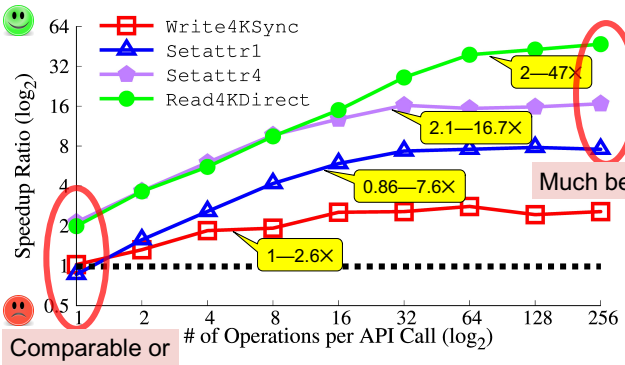
Compounding Degree



Comparable or Slightly better

Network Latency: 0.2ms

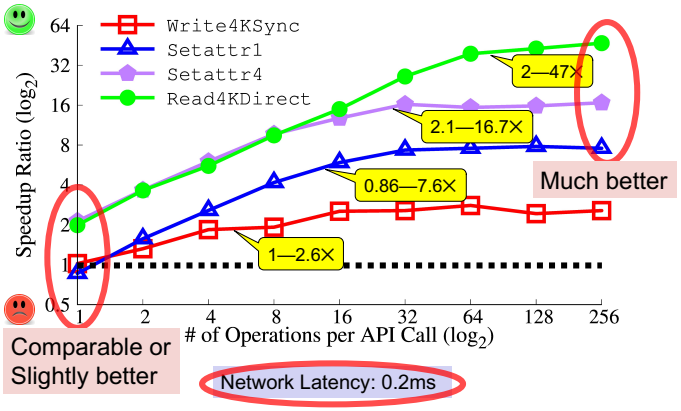
Compounding Degree



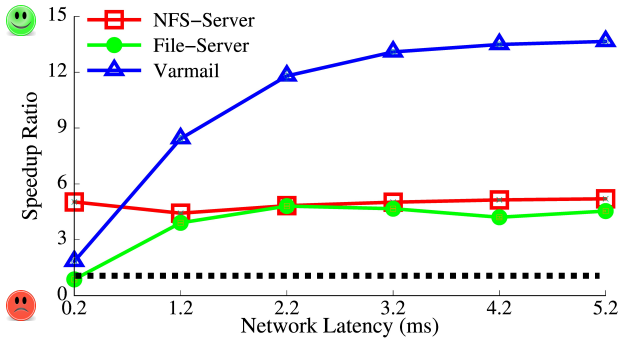
Comparable or Slightly better

Network Latency: 0.2ms

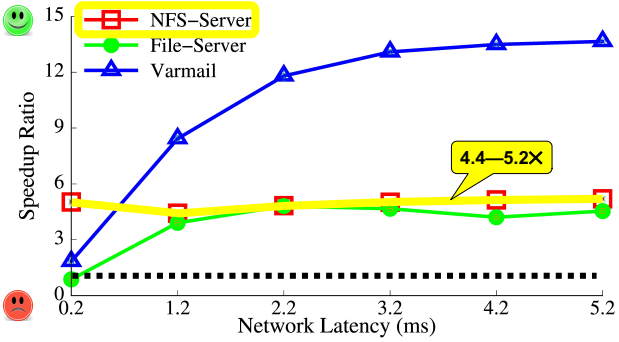
Compounding Degree



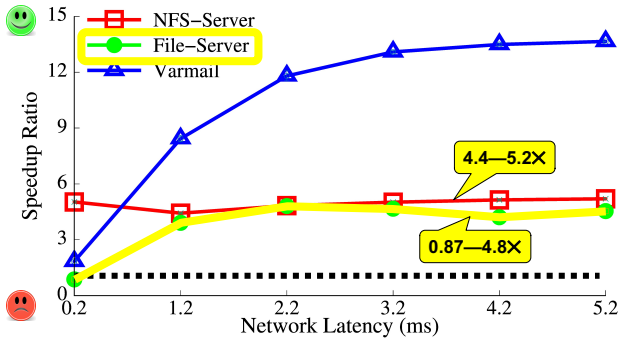
Filebench Workloads



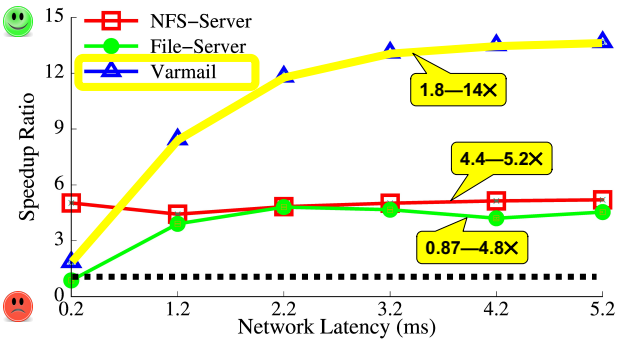
Filebench Workloads



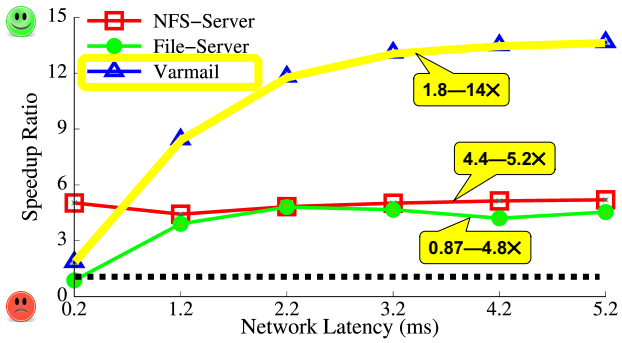
Filebench Workloads



Filebench Workloads

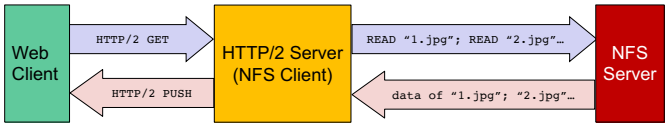


Filebench Workloads

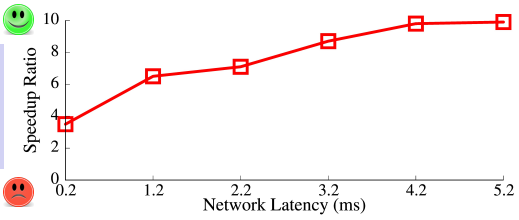


Filebench: ++759 and --37 in C

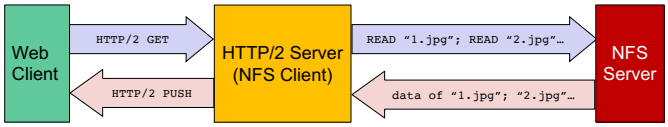
HTTP/2 Server



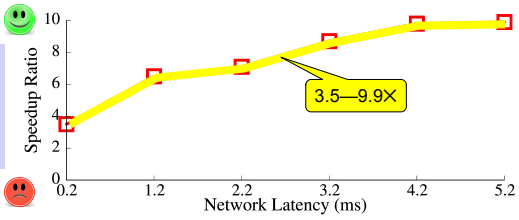
The HTTP/2 server reads and pushes a typical Web page that contains 96 objects (html, jpg, js, css) totaling round 2MB.



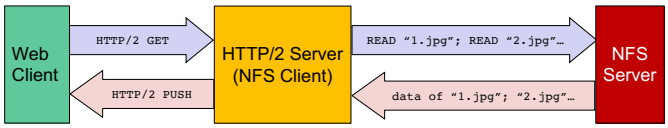
HTTP/2 Server



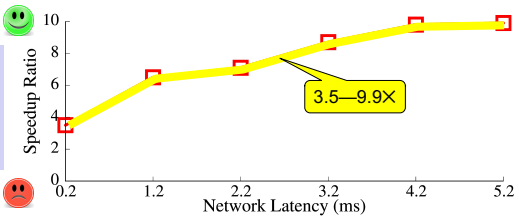
The HTTP/2 server reads and pushes a typical Web page that contains 96 objects (html, jpg, js, css) totaling round 2MB.



HTTP/2 Server



The HTTP/2 server reads and pushes a typical Web page that contains 96 objects (html, jpg, js, css) totaling round 2MB.




```
nghttp2: ++543 and --108 in C++
```


Conclusions

- A set of vectorized file-system API to take advantage of NFSv4 compound procedures without changing NFS protocols or servers
- Implemented vNFS in user-space
- Porting applications was generally easy
- Improved performance by up to 200×
- vNFS made NFS more usable in high-latency networks

Conclusions

- A set of vectorized file-system API to take advantage of NFSv4 compound procedures without changing NFS protocols or servers
- Implemented vNFS in user-space
- Porting applications was generally easy
- Improved performance by up to 200×
- vNFS made NFS more usable in high-latency networks

Conclusions

- A set of vectorized file-system API to take advantage of NFSv4 compound procedures without changing NFS protocols or servers
- Implemented vNFS in user-space
- Porting applications was generally easy
- Improved performance by up to 200×
- vNFS made NFS more usable in high-latency networks

Conclusions

- A set of vectorized file-system API to take advantage of NFSv4 compound procedures without changing NFS protocols or servers
- Implemented vNFS in user-space
- Porting applications was generally easy
- Improved performance by up to 200×
- vNFS made NFS more usable in high-latency networks

Conclusions

- A set of vectorized file-system API to take advantage of NFSv4 compound procedures without changing NFS protocols or servers
- Implemented vNFS in user-space
- Porting applications was generally easy
- Improved performance by up to 200×
- vNFS made NFS more usable in high-latency networks

Conclusions

- A set of vectorized file-system API to take advantage of NFSv4 compound procedures without changing NFS protocols or servers
- Implemented vNFS in user-space
- Porting applications was generally easy
- Improved performance by up to 200×
- vNFS made NFS more usable in high-latency networks

Limitations and Future Work

- Client-side caching (appending)
- Transactional compounds
- Parallel processing of operations

Limitations and Future Work

- Client-side caching (appending)
- Transactional compounds
- Parallel processing of operations

Limitations and Future Work

- Client-side caching (appending)
- Transactional compounds
- Parallel processing of operations

Limitations and Future Work

- Client-side caching (appending)
- Transactional compounds
- ☞ Parallel processing of operations

vNFS: Maximizing NFS Performance with Compounds and Vectorized I/O

Q&A

Ming Chen, Dean Hildebrand, Henry Nelson,
Jasmit Saluja, Ashok Sankar Harihara Subramony, and Erez Zadok

https://github.com/sbu-fsl/txn-compound

© 2017 Stony Brook University