The Design and Implementation of the **W**arp **T**ransactional **F**ilesystem

Robert Escriva, Emin Gün Sirer

Cornell University

Symposium on Networked Systems Design and Implementation
March 18, 2016

Common Trends in Distributed Filesystems

Compromises or limitations are often introduced in search of higher performance:

- ✗ Weak guarantees:
  - Eventual consistency
  - “Consistent, but undefined”
- ✗ Narrow interfaces:
  - Writes must be sequential
  - Concurrent writes prohibited
- ✗ Unscalable design:
  - Full-bisection bandwidth
  - Large “master” server

Warp Transactional Filesystem (WTF)

WTF represents a new design point in the space of distributed filesystems

WTF employs the *file slicing abstraction* to provide applications with strong guarantees and zero-copy filesystem interfaces

✓ Strong guarantees: transactionally access and modify the filesystem

✓ Expanded interface: traditional POSIX APIs and new zero-copy APIs

✓ Scalable Design: avoids centralized master or expensive network bottlenecks

Zero-Copy File Slicing APIs

- Traditional APIs transfer bytes back and forth through the filesystem interface
- File-slicing APIs deal in *references* to data already in the filesystem

`yank` Obtain references to data in the filesystem

- Analogous to `read`

`paste` Write referenced data back to the filesystem

- Analogous to `write`

`append` Append referenced data to the end of a file

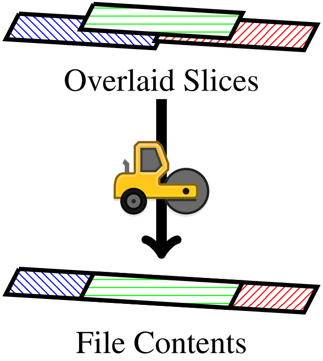
- Optimized for concurrency

`concat` Merge one or more files to create a new file

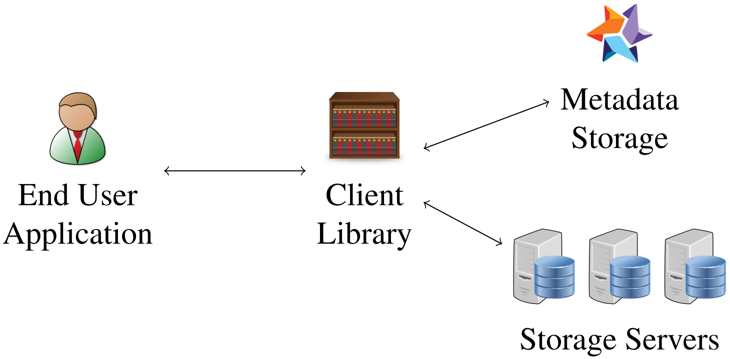
- Does not read or write data from the input files

The File Slicing Abstraction

- The central abstraction is a *slice*: an immutable, byte-addressable, arbitrarily sized sequence of bytes
- A file is represented by a sequence of slices that, when overlaid, comprise the file's contents



WTF Architecture



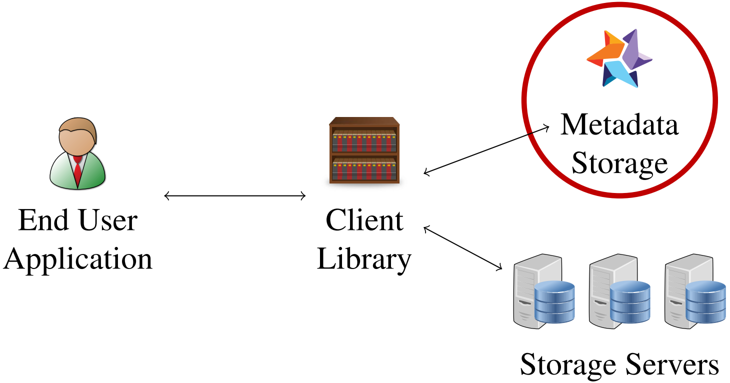
Metadata Storage

End User Application

Client Library

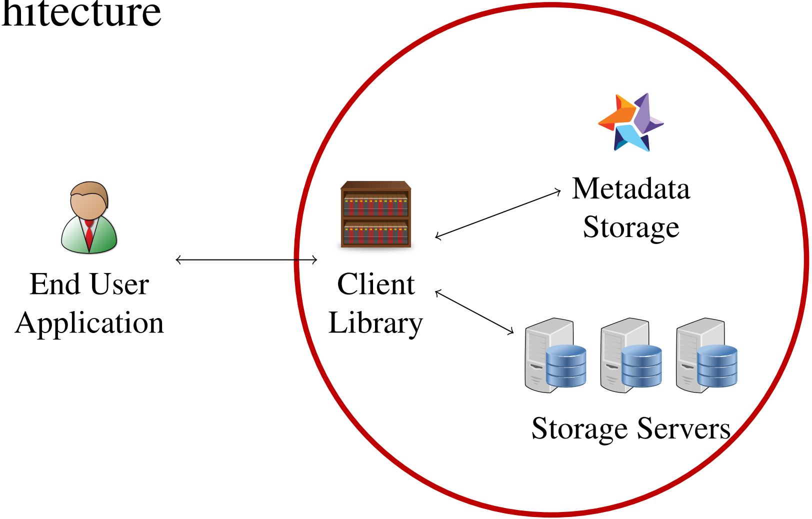
Storage Servers

WTF Architecture



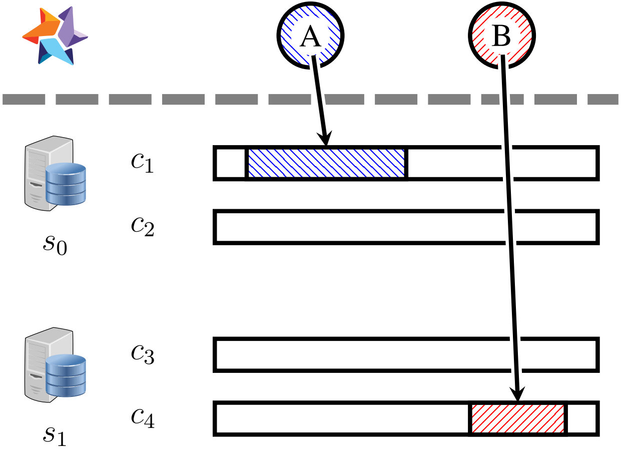
The metadata storage provides transactional operations over the metadata

WTF Architecture



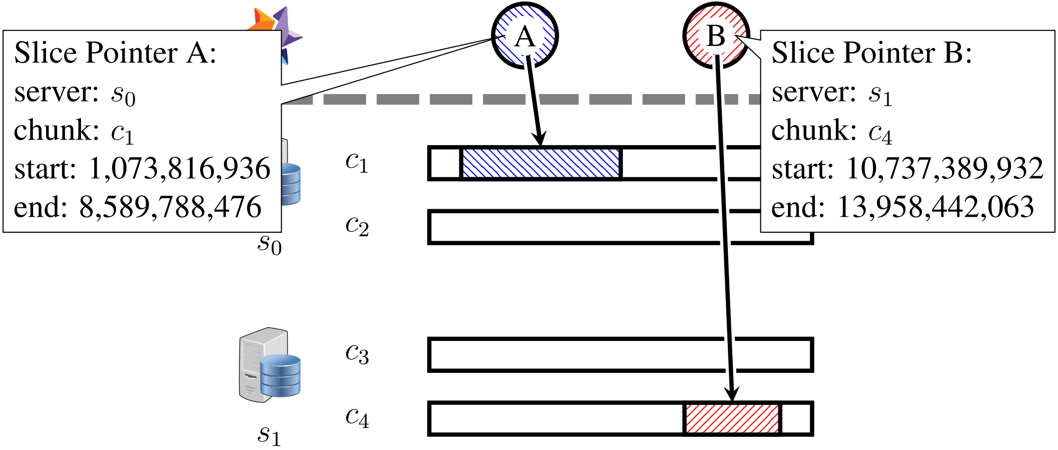
The client library extends these transactional guarantees to the end user

Slices and Slice Pointers



Slices reside on storage servers, while pointers to slices reside in HyperDex

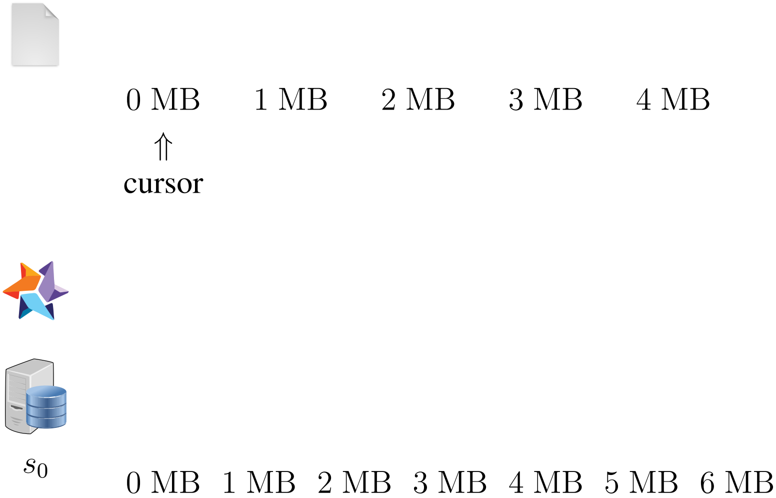
Slices and Slice Pointers



Slice Pointer A:
server: s_0
chunk: c_1
start: 1,073,816,936
end: 8,589,788,476

Slice Pointer B:
server: s_1
chunk: c_4
start: 10,737,389,932
end: 13,958,442,063

Slice pointers directly indicate a slice’s location in the system



0 MB 1 MB 2 MB 3 MB 4 MB

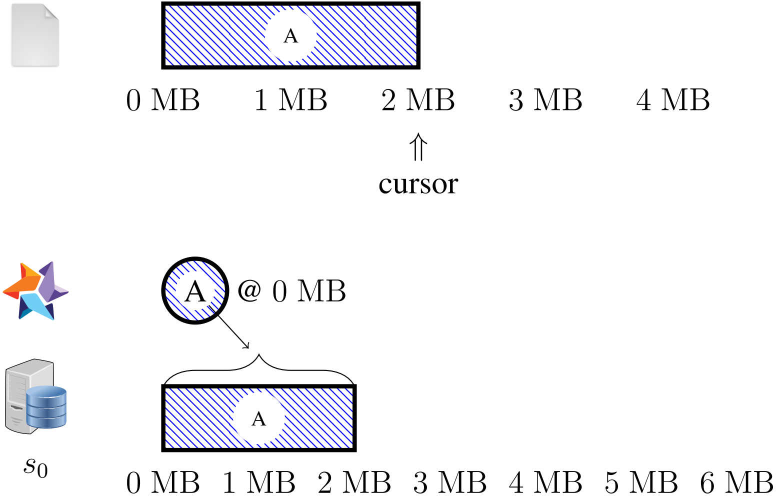
⇑

cursor

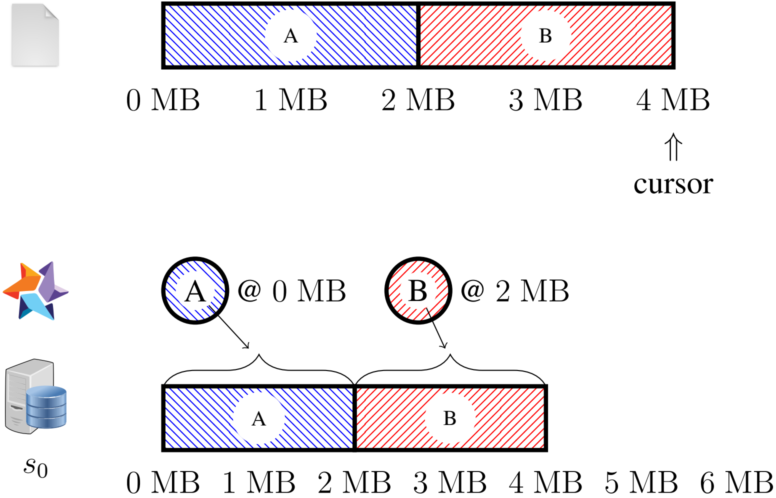
s_0

0 MB 1 MB 2 MB 3 MB 4 MB 5 MB 6 MB

An empty file has no metadata and occupies no space on storage servers



A 2 MB write writes to the storage servers and metadata



A

B

0 MB 1 MB 2 MB 3 MB 4 MB

⇑

cursor

A @ 0 MB

B @ 2 MB

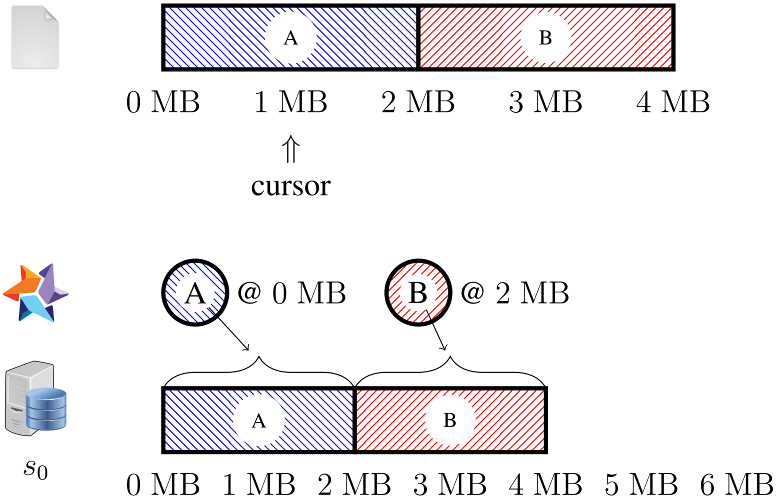
A

B

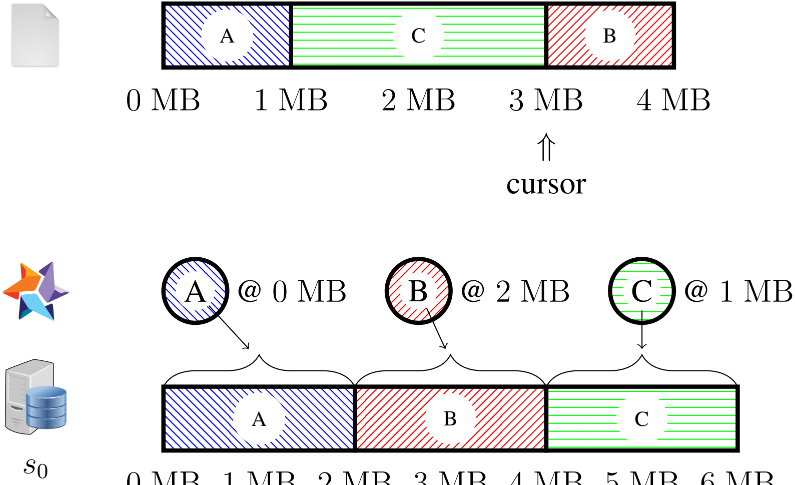
s_0

0 MB 1 MB 2 MB 3 MB 4 MB 5 MB 6 MB

Another 2 MB write



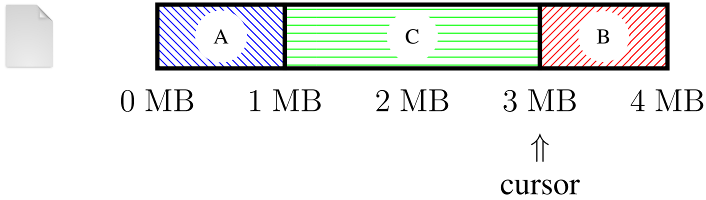
WTF supports writes at arbitrary offsets within files

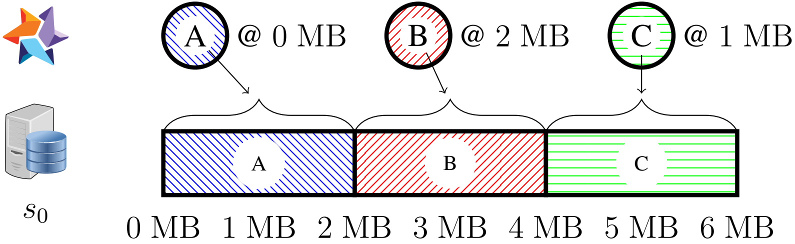


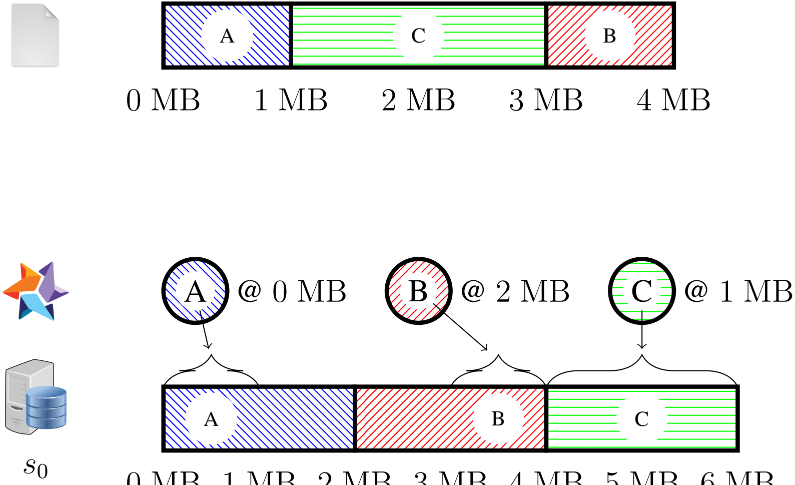
A 2 MB write that overwrites part of both prior writes

Metadata Compaction

- *Compaction* reduces the size of the metadata list by removing references to unused portions of slices
- Because slice pointers directly reference the location of files, they can be modified in the metadata list using local computation
- Consequently, compaction occurs entirely at the metadata level


A
C
B
0 MB
1 MB
2 MB
3 MB
4 MB
⇑
cursor

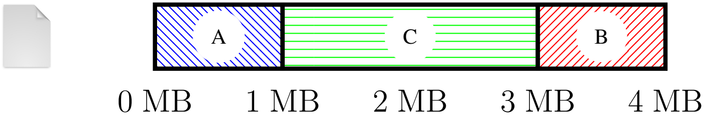

A @ 0 MB
B @ 2 MB
C @ 1 MB
A
B
C
s_0
0 MB
1 MB
2 MB
3 MB
4 MB
5 MB
6 MB



Compaction eliminates references to overwritten or erased data

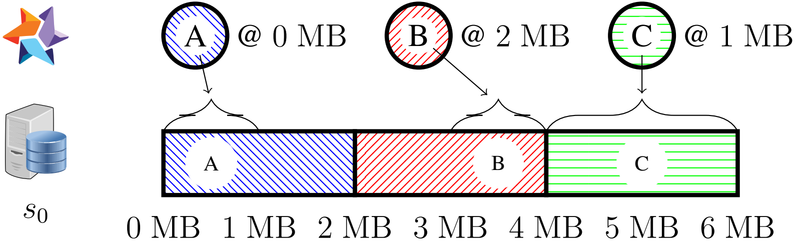
Garbage Collection

- Garbage collection cleans up the slices no longer referenced by any slice pointer
- WTF periodically scans the filesystem and collects all slice pointers
- Storage servers use the scan, along with their local data, to determine which data is garbage



A C B

0 MB 1 MB 2 MB 3 MB 4 MB

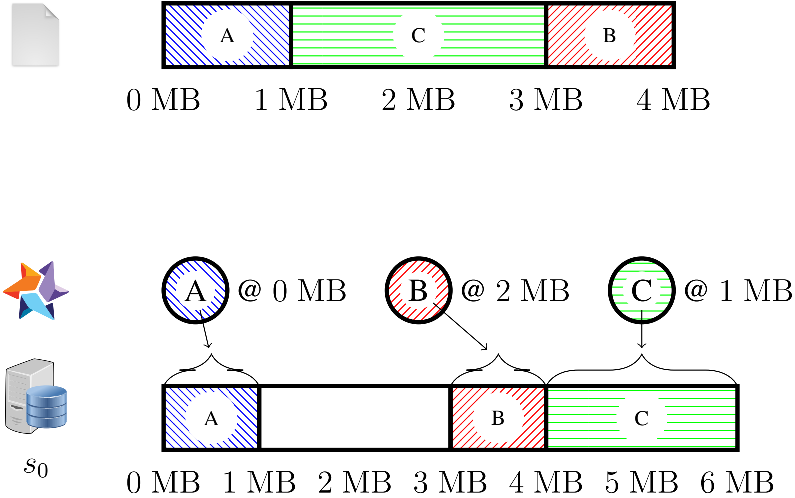


A @ 0 MB B @ 2 MB C @ 1 MB

A B C

s_0

0 MB 1 MB 2 MB 3 MB 4 MB 5 MB 6 MB

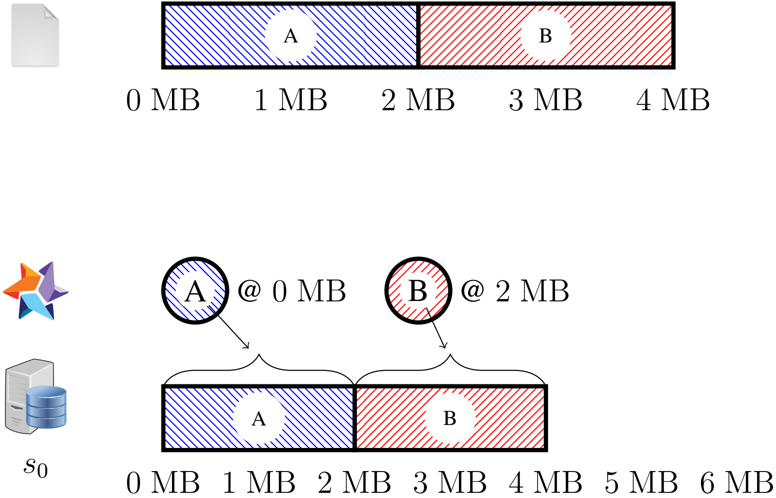


Garbage is freed from the underlying filesystem

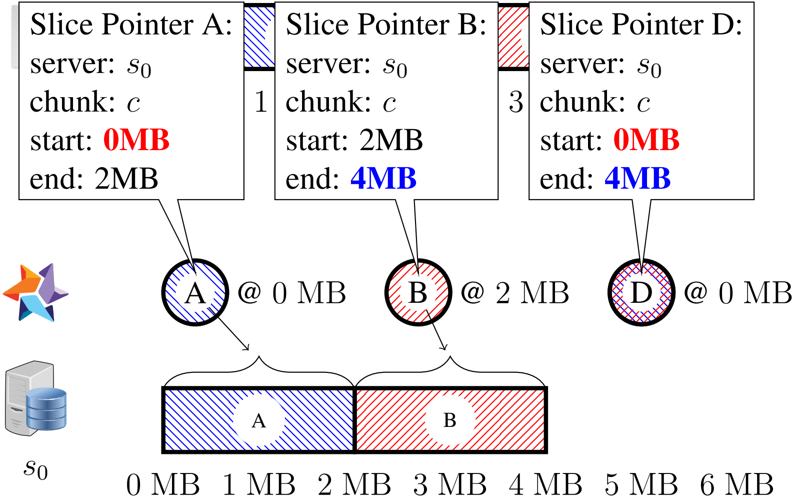
Locality-Aware Slice Placement

Locality-aware slice placement prevents fragmentation when writing sequentially

- Slices placed contiguously on storage servers improve locality when reading files
- Consistent hashing across storage servers in the system on a per-file basis increases probability that sequentially written slices are adjacent
- The metadata for adjacent slices may be represented in a more compact form



Locality-aware slice placement reduces fragmentation

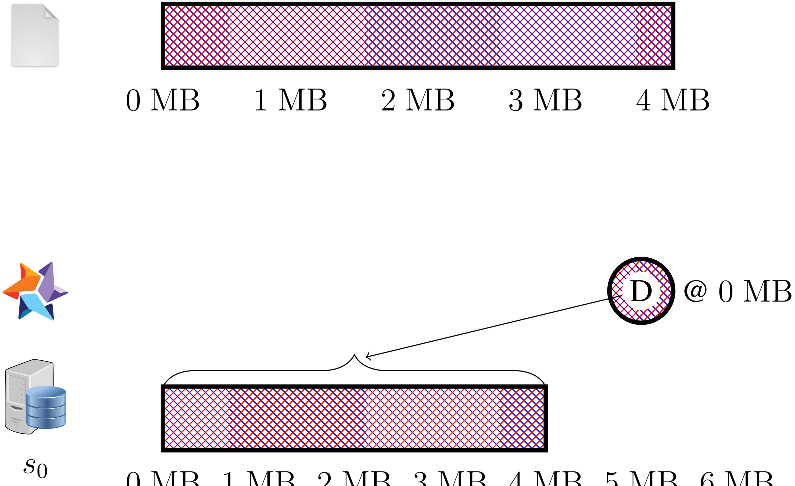


Slice Pointer A:
server: s_0
chunk: c
start: **0MB**
end: 2MB

Slice Pointer B:
server: s_0
chunk: c
start: 2MB
end: **4MB**

Slice Pointer D:
server: s_0
chunk: c
start: **0MB**
end: **4MB**

Adjacent slices may be represented by a new, merged slice pointer



The new slice pointer represents the contiguous range on the storage servers

WTF Applications

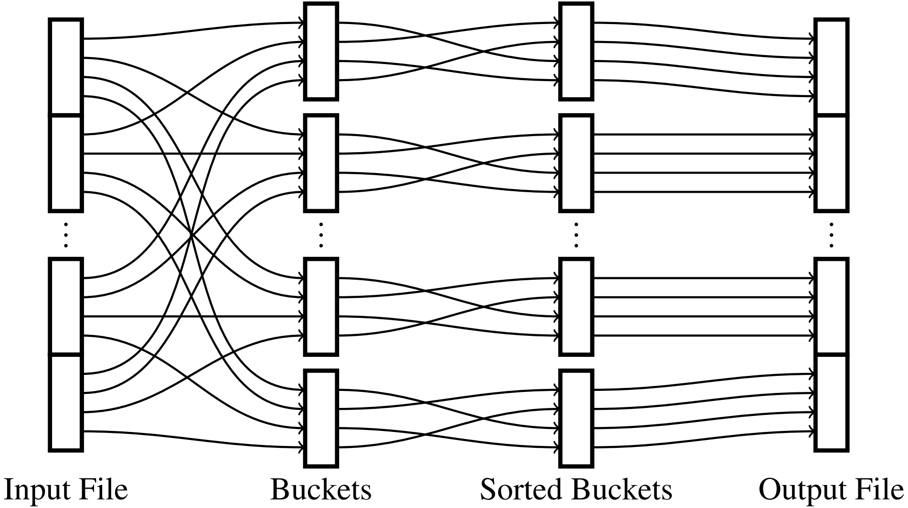
MapReduce Sort: `concat` enables an efficient bucket-based merge sort

Work Queue: `append` units of work are appended to the file; all contention happens in the metadata layer

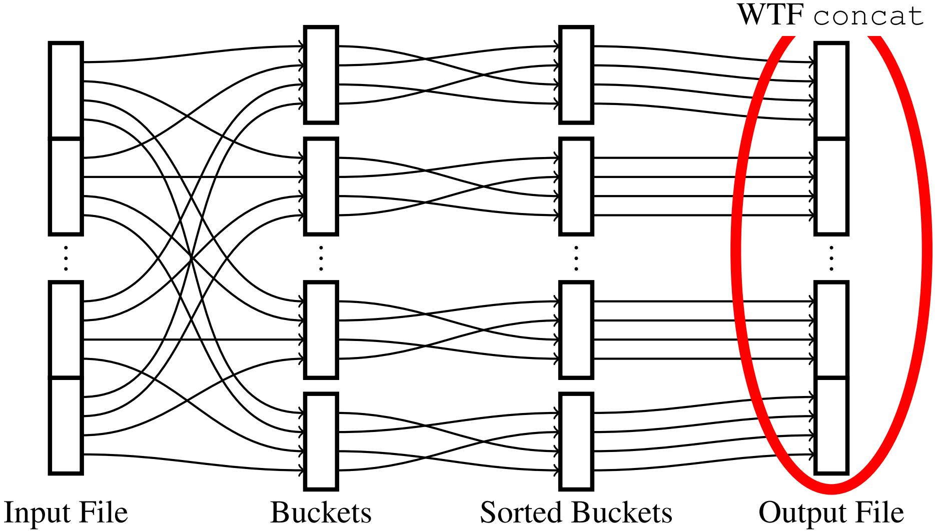
Video editor: `yank` and `paste` enable the editor to reorder scenes without rewriting the movie

Fuse Bindings: transactional behavior exposed to the user for easy data exploration

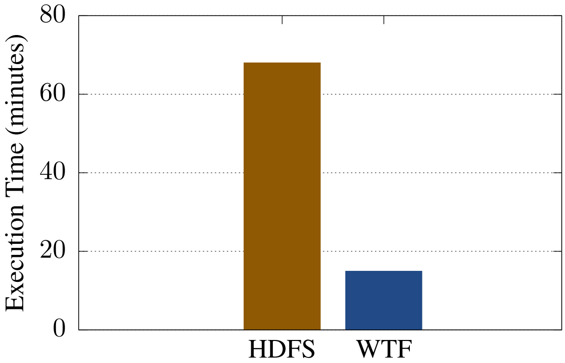
Application: MapReduce Sort



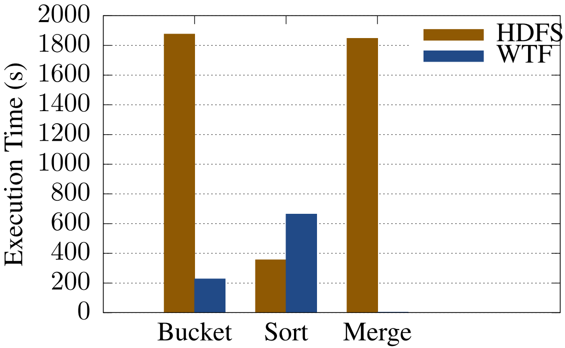
Application: MapReduce Sort



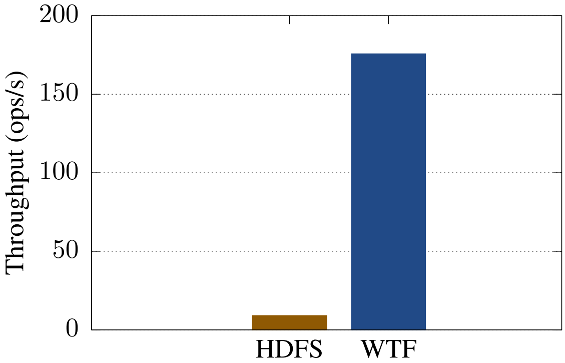
Application: MapReduce Sort



Application: MapReduce Sort

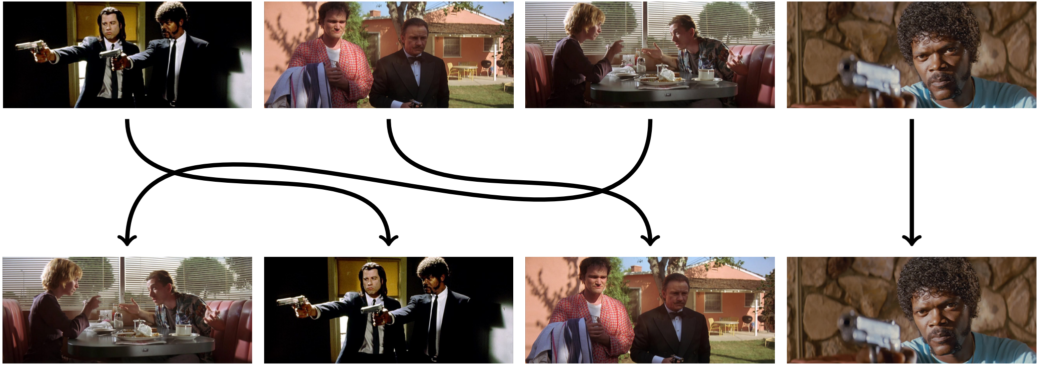


Application: Work Queue



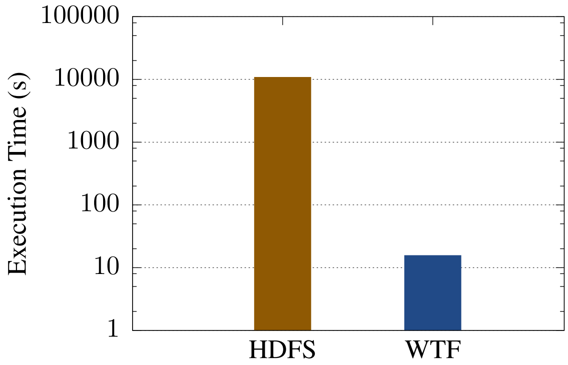
Application: Video Editor

Chronological Order



Final Cut

Application: Video Editor

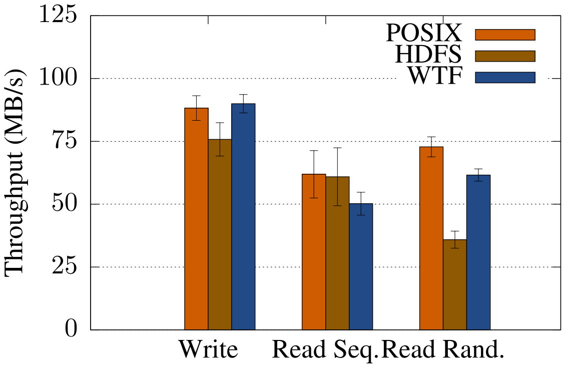


WTF can rewrite 377 GB of raw movie footage in 16 s using file slicing—effectively 23 GB/s, as opposed to rewriting the footage using traditional APIs, which requires approximately three hours

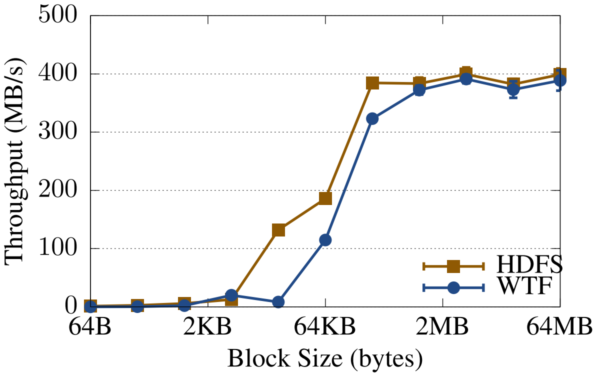
Application: Interactive Transactions

```
# wtf begin-transaction
# ls
./data.0000  ./data.0001
./data.0002  ./data.0003
....
# rm -rf *
# ls
# wtf abort-transaction
# ls
./data.0000  ./data.0001
./data.0002  ./data.0003
....
```

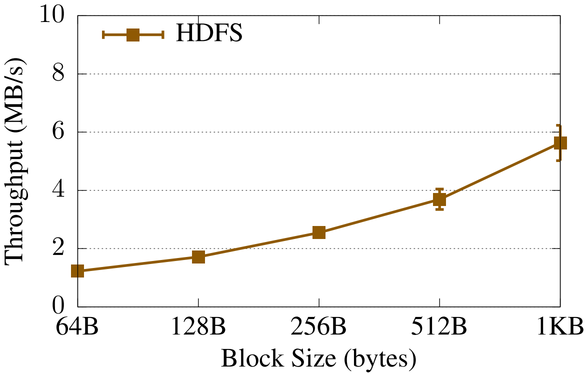
Microbenchmark: Baseline Performance



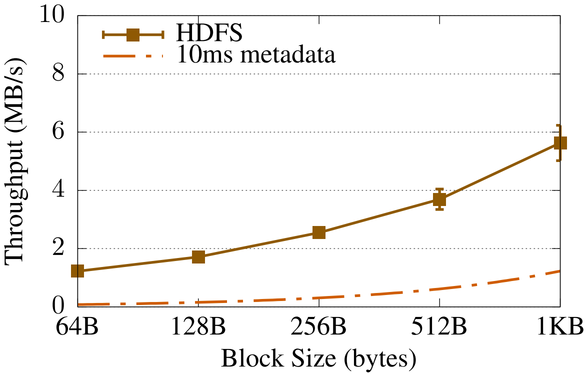
Microbenchmark: Write Sequential



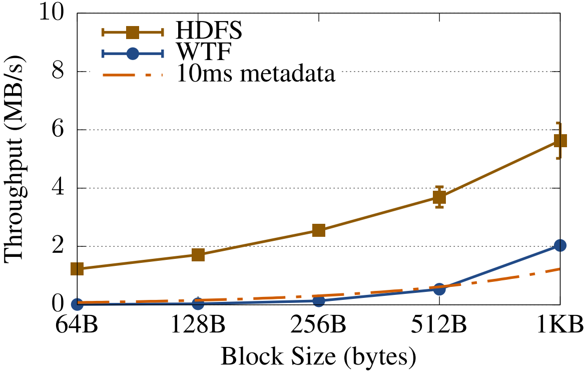
Microbenchmark: Write Sequential



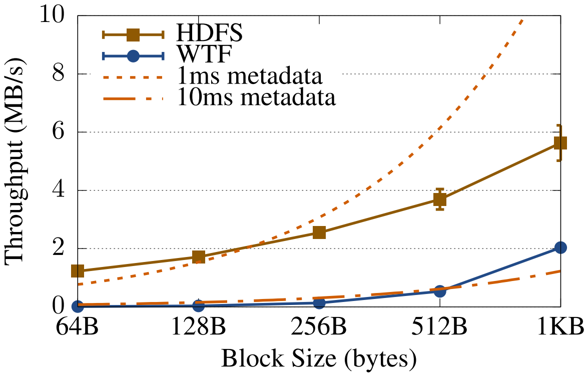
Microbenchmark: Write Sequential



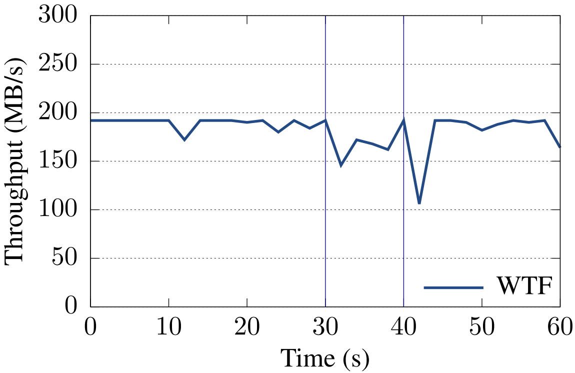
Microbenchmark: Write Sequential



Microbenchmark: Write Sequential



Microbenchmark: Fault Tolerance



Related Work

- Distributed Filesystems
    - Farsite, AFS, xFS, Swift, Petal, Frangipani, NASD, Panasas
- Data Center Filesystems
    - CalvinFS, GFS, HDFS, Salus, Flat Datacenter Storage, Blizzard, f4, Pelican
- Transactional Filesystems
    - QuickSilver, Transactional LFS, Valor, PerDis FS, KBDBFS, Inversion, Amino

Conclusion

WTF is a new design point in distributed filesystems that leverages the file slicing abstraction to provide:

- Transactional guarantees
- Expanded APIs
- Improved performance