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

Robert Escriva, Emin Gün Sirer

Computer Science Department, Cornell University

Presented By

Alex Garcia

Common Trends in Distributed Filesystems

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

- Weak guarantees: (Google File System)
  - Eventual consistency
- Narrow interfaces: (Hadoop Distributed File System)
  - Writes must be sequential
  - Concurrent writes prohibited
- Unscalable design: (FLAT Data Center Storage)
  - Full-bisection bandwidth
  - Large "master" server

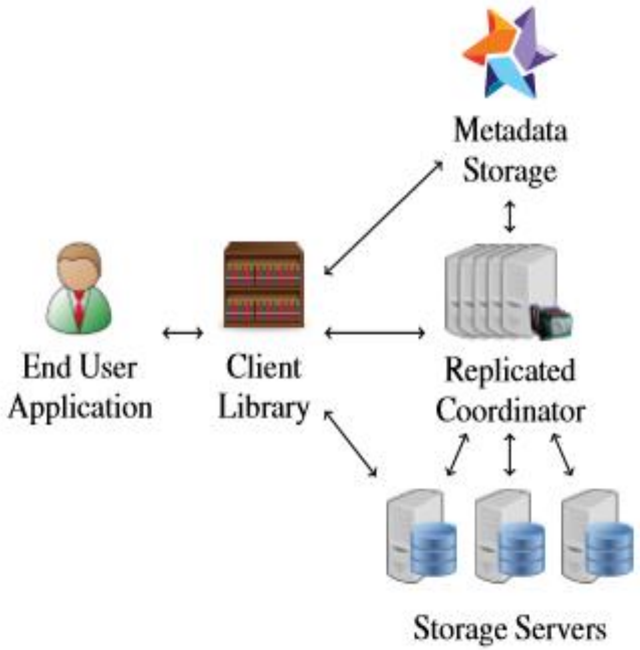
• Warp Transactional Filesystem (WTF)

- 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

WTF Architecture



- Client Library – Contains the majority of the functionality of the system where it combines the Metadata and data into a coherent file system and provides transactional guarantees to the end user.

- Metadata Storage – Provides transactional operations over metadata using HyperDex Warp. (No-SQL, Key-Value Store, Fault Tolerant and Strong Consistency, ACID Transactions)

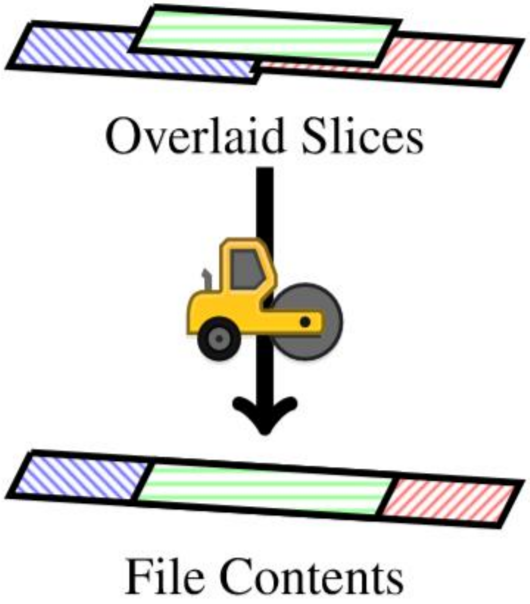
- Storage Servers – Hold the file system data and handle most of the I/O

- Replicated Coordinator – Serves as a rendezvous point for all the components of the system and maintains a list of the storage servers.

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 (No copying of file content needed)
  - 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

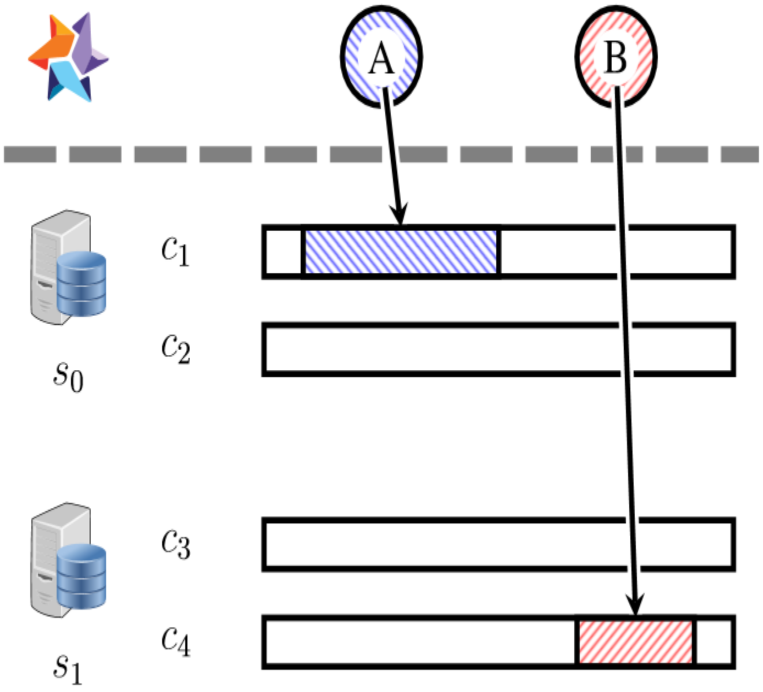
File Slicing Abstraction



- 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.
- Metadata is a sequence of slices.

Slices and Slice Pointers

Slices pointers to slices reside in HyperDex.
Slice pointers directly indicate a slice's location in the system

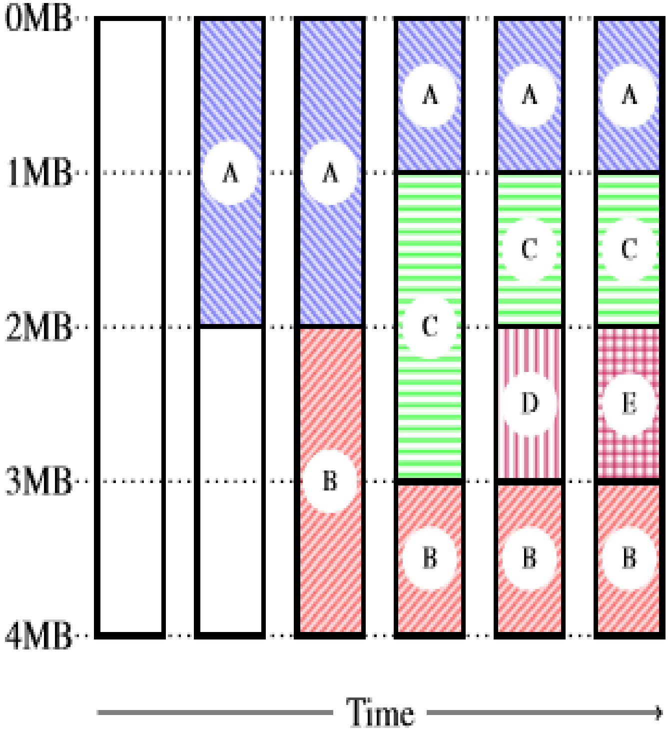


Slices reside on storage servers

Slice Pointer Tuple:

- Unique Identifier for the storage server holding the slice
- Local Filename containing the slice on that storage server
- File offset of the slice within the file
- Length of the slice
- Integer offset where the slice is to be overlaid.

Slices

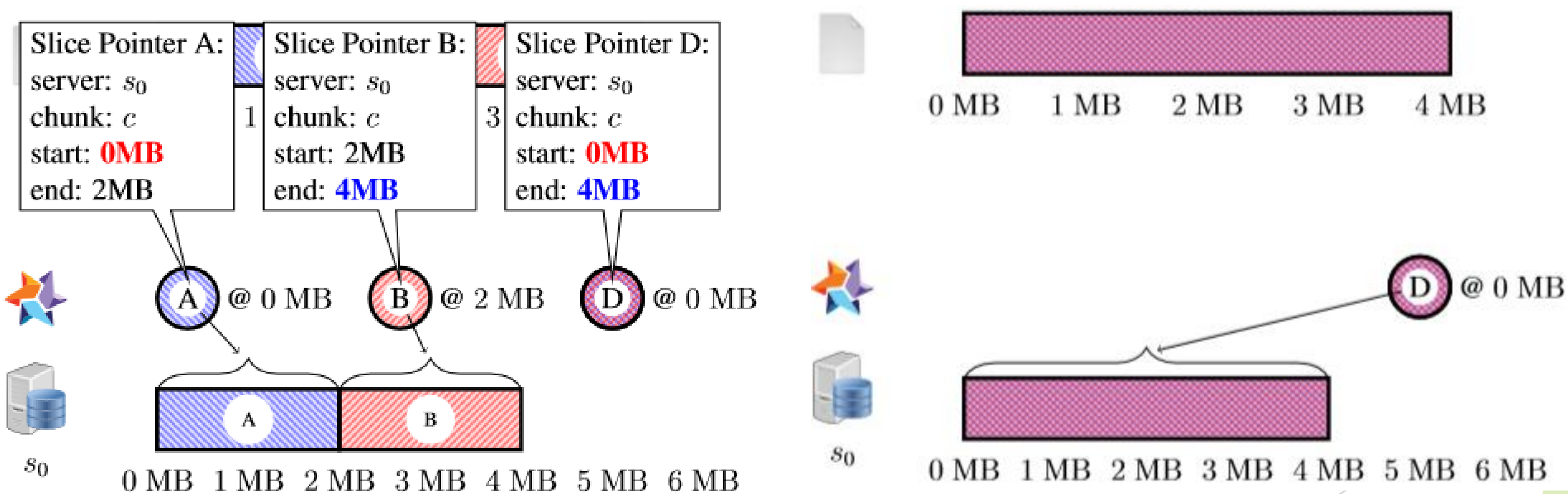


Final Metadata:
A@[0,2], B@[2,4], C@[1,3], D@[2,3], E@[2,3]
Compacted Final Metadata:
A@[0,1], C@[1,2], E@[2,3], B@[3,4]

- A Writer creates file slices on the storage servers.
- Overalys them at the appropriate positions within the file by appending their slice pointers to the metadata list.
- Readers retrieve the metadata list, compact it and determines which slice must be retrieved from the storage servers.
- The overlap, the latest takes precedence.

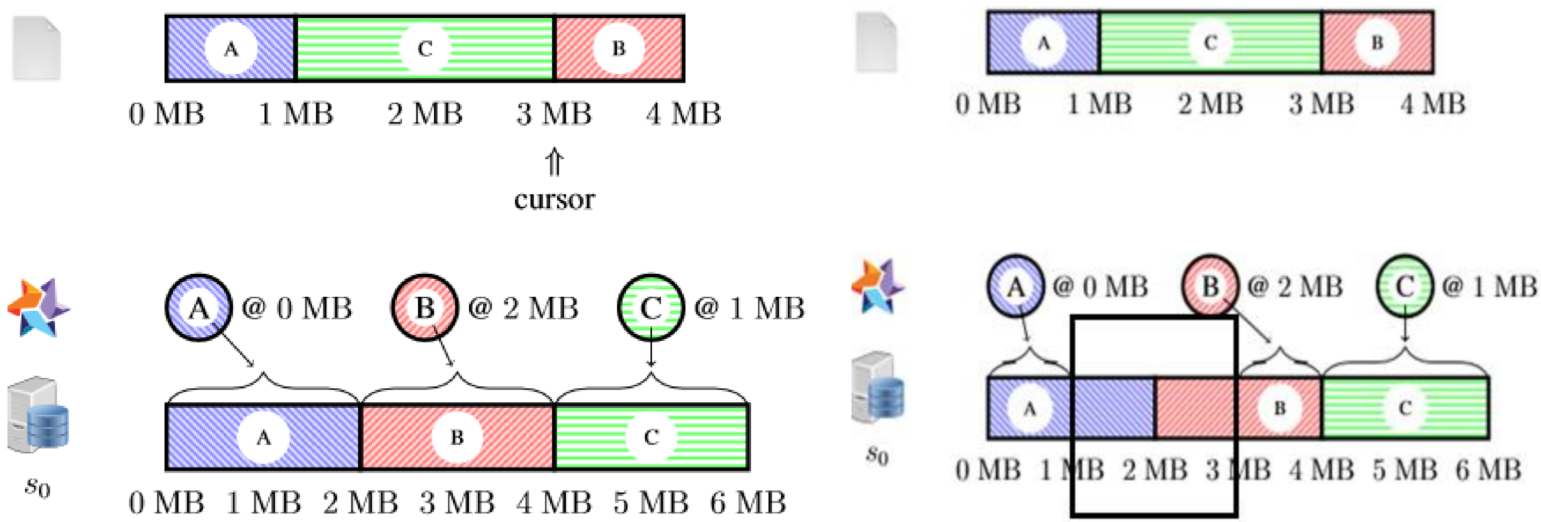
Locality-Aware Slice Placement

- Client library place slices contiguously to improve reads and metadata compaction.
- 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



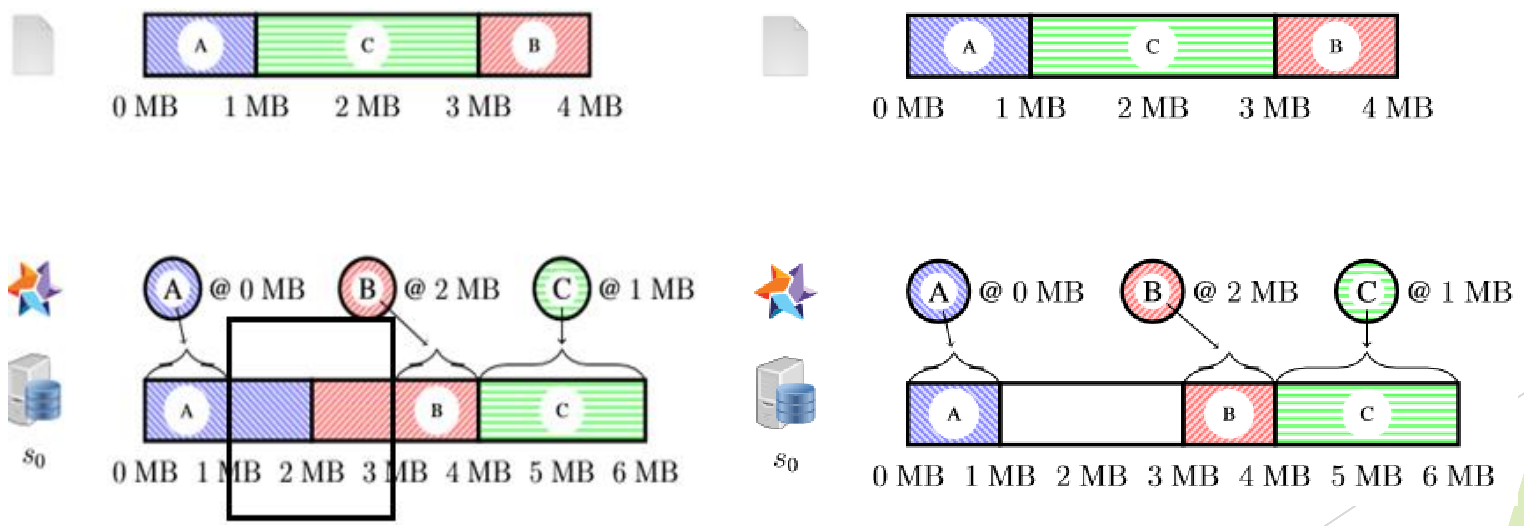
Metadata Compaction and Defragmentation

- Compaction reduces the size of the metadata list by removing references to unused/overwritten portions of slices.
- Fragmented data is rewritten within the region into a single slice and replaces the metadata with a single pointer to the slice



Garbage Collection

- Garbage collection cleans up the slices no longer referenced by any slice pointer from the results of metadata compaction.
- WTF periodically scans the filesystem and constructs a list of in-use slice pointers for each storage server.
- Storage servers use the scan, along with their local data, to determine which data is garbage.

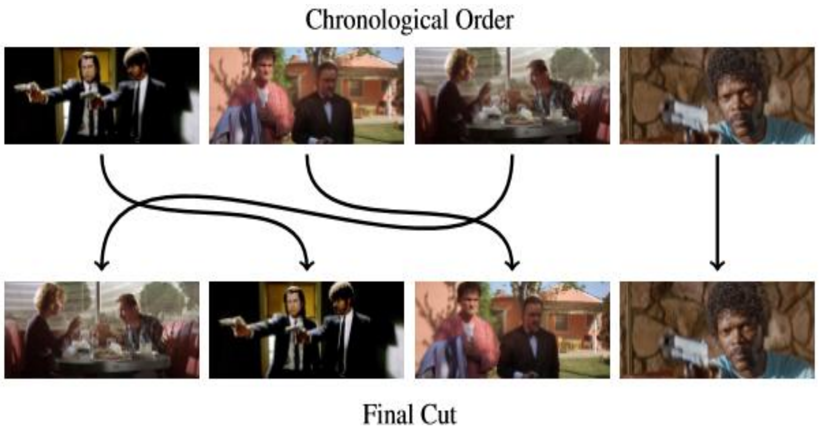


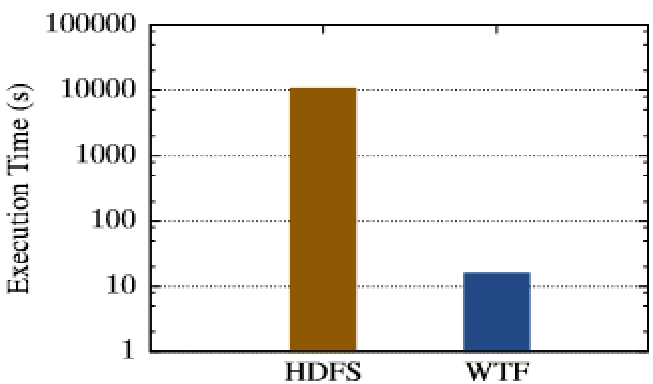
Fault Tolerance

- WTF uses replication to add fault tolerance to the system.
- Writers create multiple replicas slices on distinct servers and append their pointers atomically as one list entry.
- HyperDex uses value-dependent chaining to coordinate between the replicas and manage recovery from failures.

Applications & Evaluation

- **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





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

Related Works

- **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

QUESTIONS ?

THANK YOU!