6.5810: Flat Storage

Adam Belay <abelay@mit.edu>

Logistics

- No class Wednesday
- Reminder: Work on project report draft (Due this Friday)

Motivation

- Conventional approach: Storage should expose specific disks
- Flat storage:
  - Applications are multiplexed across *every* disk instead
  - Enabled by fast networks with full bisection bandwidth
  - Goal: Much higher storage throughput
- Other services like S3 also provide roughly flat storage, but with much higher overhead than FDS

Why disk locality has downsides

- Forces developers to think about moving code to data
- The application itself may not have locality (e.g., sort, matrix multiply)
- Waiting to get local access to a disk can cause stragglers
- Locality can cause Imbalances between disk and CPU, leaving resources idle

Quick aside: Balanced systems

- Every app needs a different ratio of memory, compute, and storage
- Each machine has a fixed ratio of memory, compute, and storage
- From the paper: “FDS is more efficient for many workloads because every job can use the cluster’s I/O bandwidth and CPU resources in exactly the ratio required.”
- Balance is key for good performance!

Flat storage abstractions

- **Blobs**: Units of data storage, with a 128-bit GUID
  - Can be any length, up to the storage capacity
  - Similar to files in UNIX, but no path name
- **Tracts**: The granularity of reads and writes from blobs
  - Larger than a typical disk block (e.g., 8 MB)
  - Goal: Sequential and random access performs the same
- **TractServers**: Runs on each machine with a disk
  - Exposes the raw disk, dividing it into tracts; No Linux filesystem

FDS Client API

Getting access to a blob
`CreateBlob(UINT128 blobGuid)`
`OpenBlob(UINT128 blobGuid)`
`CloseBlob(UINT128 blobGuid)`
`DeleteBlob(UINT128 blobGuid)`

Interacting with a blob
`GetBlobSize()`
`ExtendBlobSize(UINT64 numberOfTracts)`
`WriteTract(UINT64 tractNumber, BYTE *buf)`
`ReadTract(UINT64 tractNumber, BYTE *buf)`
`GetSimultaneousLimit()`

Locating tracts

- FDS metadata server only tracks active tract servers (TLT)
- Deterministic function determines the location
- Server = (Hash(GUID) + tract_num) mod TLT_Length
- Then give Server the blob’s GUID + tract_num to get the tract

Q: Why does Hash(GUID + tract_num) not work?

Q: Why is the TLT made of multiple random permutations of tract servers?

Q: How does this design compare to GFS?

What about blob metadata?

- FDS stores it in tract -1
- New blobs start at length 0 tracts
- Client must extend the blob before writing past the end if it
- Extensions are atomic (i.e., safe with concurrent access from other clients)
- Why? Allows clients to add a range of tracks without risk of conflict
  - i.e. atomic append
- Actual tracts are allocated *lazily* on the tract servers (i.e., on first write)

Allocating work to compute nodes

- FDS makes it possible to ignore locality
- Instead, give each worker a new task when it finishes an old one
- This is a “closed queueing system”

Replication

- FDS allows tracts to be n-way replicated across servers
- Each entry in the TLT contains n servers instead of 1 server
- Writes must be sent to every server in the TLT entry
- Reads select a single server at random
- Implication: Relaxed consistency model

Q: What about blob metadata?

- E.g., CreateBlob, ExtendBlobSize, and DeleteBlob.

Q: What about blob metadata?

- E.g., CreateBlob, ExtendBlobSize, and DeleteBlob.
- TLT entry marks one replica as the “primary”
- Primary runs two-phase commit on other replicas

Variable replication

- User can control how much replication is applied to each blob
- More replication improves read throughput, less makes write more efficient
- Maximum level of replication is determined by n-way in TLT

Failure recovery

Imagine a single tract server becomes unreachable

1. Invalidate the TLT by incrementing the version number (in each row the failed server appears)
2. Pick random tractserver to fill in the empty spaces
3. Send updated TLT assignment to every server affected
4. Wait for ack from the tract servers
5. Tractservers contact replicas and copy lost tracts

If a client uses a stale TLT entry version number, request is rejected

Networking assumptions

- Storage node’s network bandwidth >= disk bandwidth
- Full bisection bandwidth, to prevent bottlenecks
- Compute node’s network bandwidth >= I/O bandwidth

Very good assumptions for today’s datacenter racks

Single disk performance

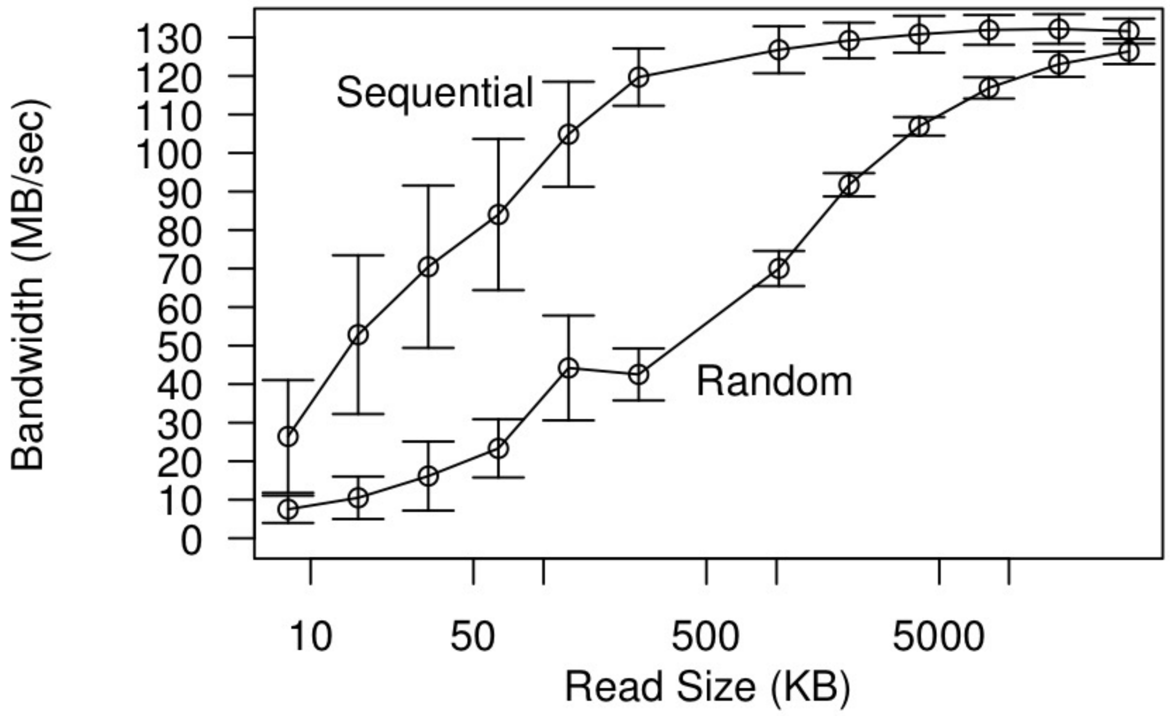
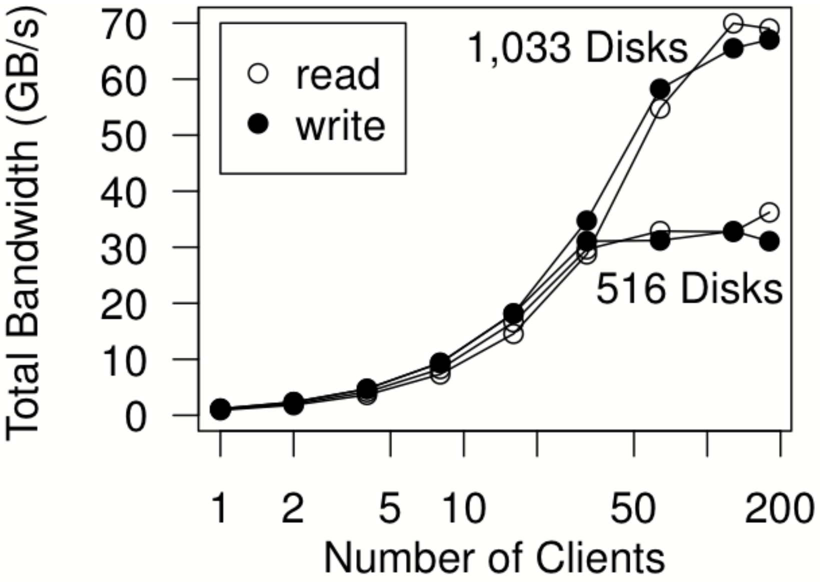
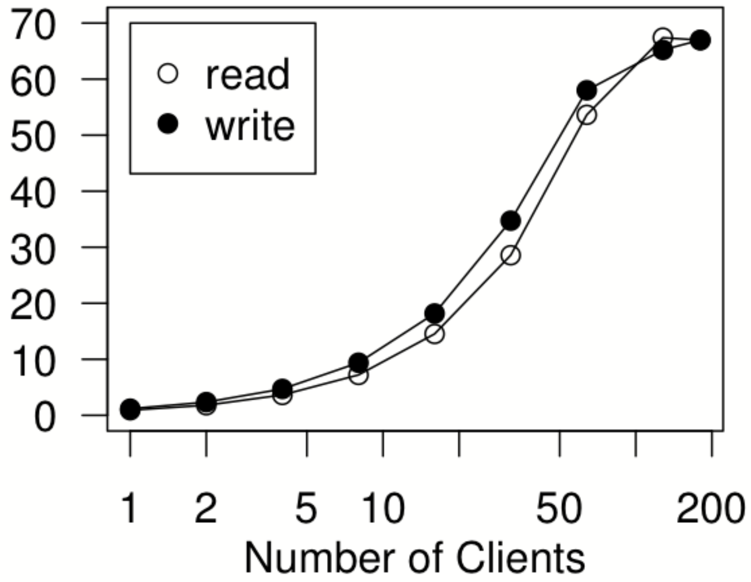


Figure 3: Performance of a single process reading to a single 10,000 RPM disk. Each point is the mean across 24 disks. Error bars show the standard deviation.

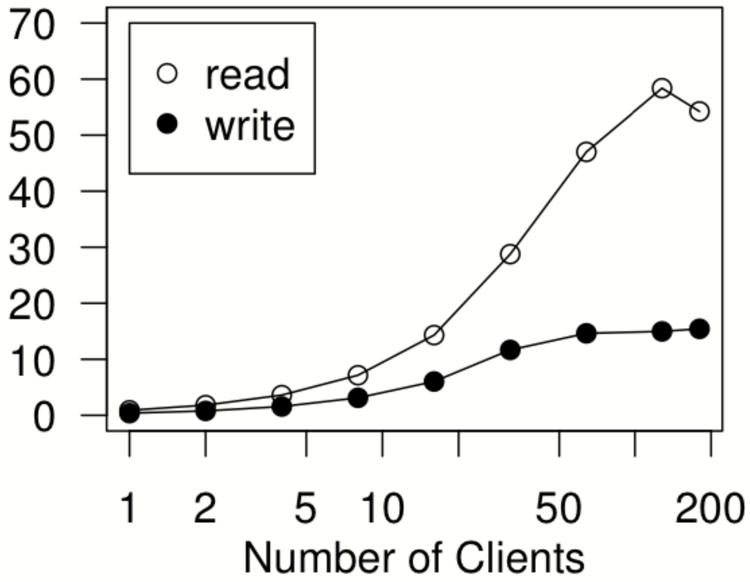
How well does FDS scale?



Sequential read + write



Random read + write



Sequential read + write
(triple replicated)

Can FDS improve sort performance?

System	Computers	Data Disks	Sort Size	Time	Implied Disk Throughput
MinuteSort—Daytona class (general purpose)					
FDS, 2012	256	1,033	1,401 GB	59 s	46 MB/s
Yahoo!, Hadoop, 2009 [25]	1,408	5,632	500 GB	59 s	3 MB/s
Yahoo!, Hadoop, 2009 [25] (unofficial 1 TB run)	1,408	5,632	1,000 GB	62 s	5.7 MB/s
MinuteSort—Indy class (benchmark-specific optimizations allowed)					
FDS, 2012	256	1,033	1,470 GB	59.4 s	47.9 MB/s
UCSD TritonSort, 2011 [27]	66	1,056	1,353 GB	59.2 s	43.3 MB/s

Actual disk throughput is ~100 MB/s

Why does FDS not saturate the disks?

Debate

- Should we ignore locality as networks become faster?
- Could a centralized metadata server perform better?
- Why is saturating storage so hard?
- FDS balances storage… How could we balance other resources like memory?