

# CS 518

## Cluster Serverless Systems: Zebra, xFS, GMS

Randy Wang  
Fall 2002  
Princeton University

## Outline

- Introduction
- Networked RAID
- Cooperative client caching
- Global memory
- Distributed everything in xFS
- Thoughts for future

CS518

1

Randy Wang

## Outline

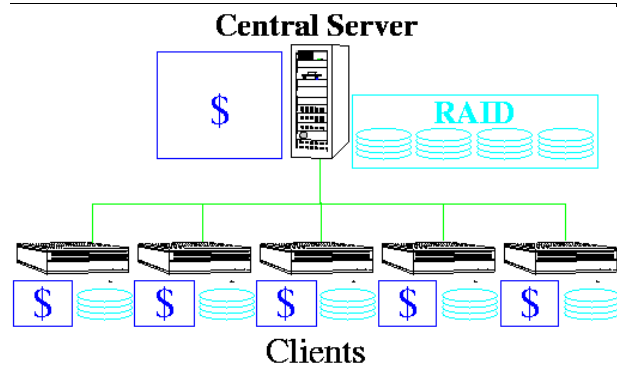
- **Introduction**
- Networked RAID
- Cooperative client caching
- Global memory
- Distributed everything in xFS
- Thoughts for future

CS518

2

Randy Wang

## Client/Server Architecture Problems



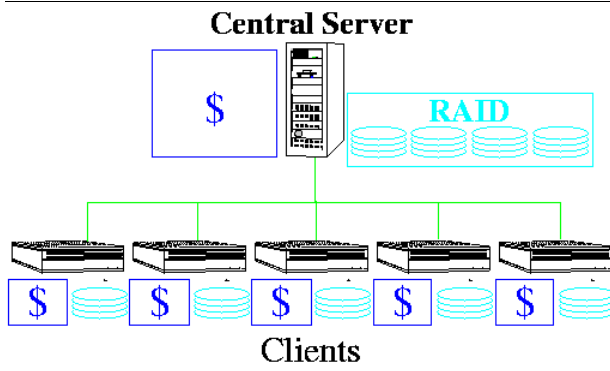
- Poor scalability
  - More bottleneck components as you add clients
- Poor performance
  - Resource and technology limitations on the server
- Poor availability
  - Server is single point of failure
- Expensive
  - Typically can't take advantage of commodity parts

CS518

3

Randy Wang

## Client/Server Architecture Problems



- Traditionally, scaling done with static partitioning of name space by multiple servers
- Static, and manual
- So potentially non-even load, and not responsive to changes
- Server replication can improve performance at the cost of maintaining consistency

CS518

4

Randy Wang

## Technology Opportunities

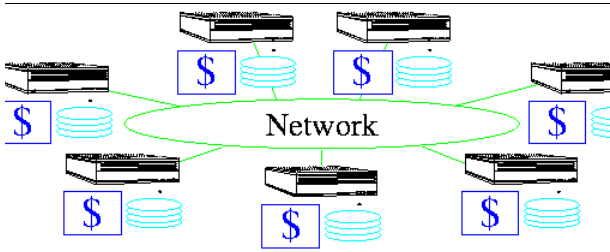
- Workstations
  - Powerful and cheap
- Networks
  - Scalable switched architecture
  - Low latency communication software
- Advances in file system technology
  - LFS, RAID, networked RAID
  - Networked memory

CS518

5

Randy Wang

## Serverless Architecture



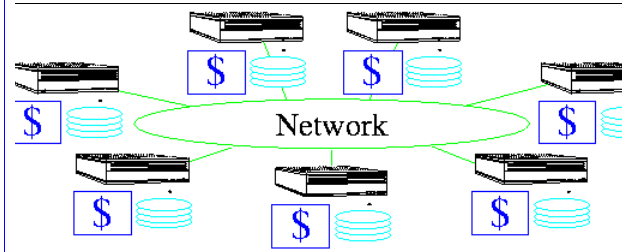
- Philosophy: "anything, anywhere"
  - Distributes disk storage, memory file cache, and metadata management
  - Location independent (meaning dynamic placement)
  - Any node can take over for any other node, for load balancing or availability
  - Different from LOCUS: harvest **aggregate** resource

CS518

6

Randy Wang

## Serverless Architecture



- Performance
  - exploit aggregate resource: disks, memory, CPUs
- Scalability
  - easy incremental scaling
- Availability
  - redundancy and interchangeable components
- Cost effective

CS518

7

Randy Wang

## Main Pieces

- Disks
  - Networked RAID (Zebra, Berkeley, 93)
- Memory
  - Cooperative client caching (xFS, Berkeley, 94)
  - Global memory system (GMS, UW, 95)
- Metadata
  - Distributed metadata (GMS)
  - Distributed secondary storage metadata and distributed cache coherence control (xFS)

CS518

8

Randy Wang

## Outline

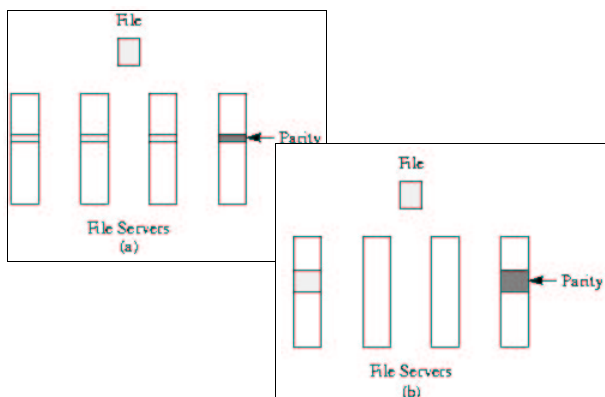
- Introduction
- **Networked RAID**
- Cooperative client caching
- Global memory
- Distributed everything in xFS
- Thoughts for future

CS518

9

Randy Wang

## How to do Networked RAID



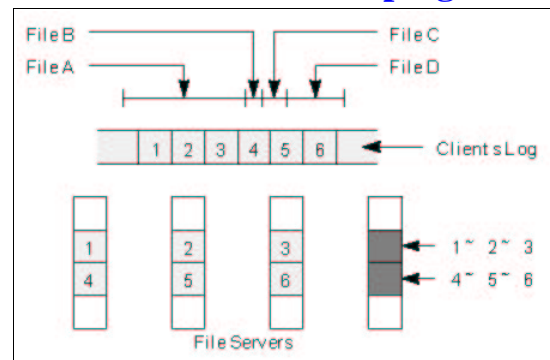
- Old idea: per-file striping
- Problem: small files
  - Depends on the granularity of striping, either
  - Large performance overhead, or
  - Large storage overhead
  - Also, potential imbalance of disks

CS518

10

Randy Wang

## Key Zebra Contribution: “Per-client” Striping



- Network version of LFS+RAID
- Stripe and compute parity for per-client log

CS518

11

Randy Wang

## Per-client Striping Advantages and Questions

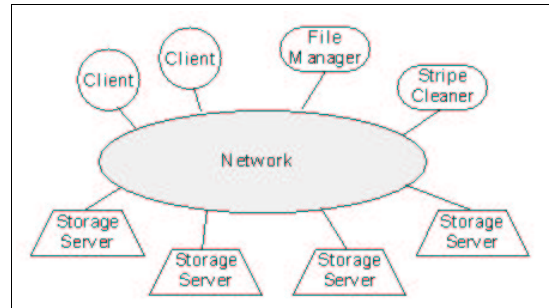
- Advantages
  - Nice solution for small files: good performance and no special handling
  - Nice load balancing
  - Efficient and simple parity calculation
- Questions
  - Why is the same “segment” unit used for both networks and disks?
  - A justification (sort of): networks were slow for them
  - Death of new data on the clients
  - But also inherits problems from LFS
    - + Cleaning (garbage collection) cost
    - + Potential lack of spatial locality
    - + “fsync problem”, and hence problem with transaction workload

CS518

12

Randy Wang

## Zebra Components



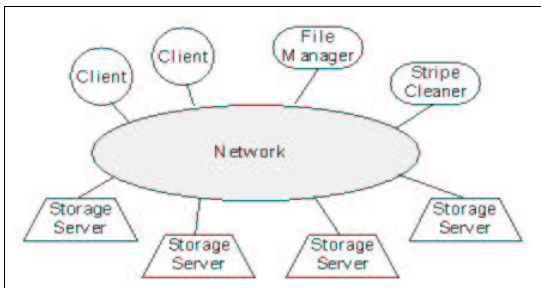
- Clients:
  - Aware of distributed storage
  - Increased intelligence (complexity)
- Storage servers
  - Like network disks
  - A “fragment” interface, simple and dumb

CS518

13

Randy Wang

## Zebra Components



- File managers:
  - Stores attributes and directories in a local LFS
  - Simple to implement, but
  - A bottleneck for performance and reliability
  - Nice proposal: store metadata on the storage servers
- Cleaner:
  - Another bottleneck and much room for improvement

CS518

14

Randy Wang

## Cool Idea: Communication via “Deltas”

- What are deltas?
  - Deltas: change records embedded in the logs
  - Written by clients, the file manager, and the cleaner
  - Read by the file manager to update its metadata
  - Read by the cleaner to update its utilization info and to identify live blocks
- Why is this a cool communication mechanism?
  - Reliable
  - Ordered
  - Good for asynchronous communication
  - Good for crash recovery
- Complications
  - Conflicting updates by clients and the cleaner
  - Ordering records from different client logs

CS518

15

Randy Wang

## Crash Recovery

- Two powerful techniques in general
  - Logging, and
  - Checkpointing
- Accomplish two tasks with the same mechanism
  - Consistency of data structures on different machines during normal operation
  - Recovery of data structures after crash
- This applies to
  - File manager metadata
  - Cleaner state

CS518

16

Randy Wang

## Outline

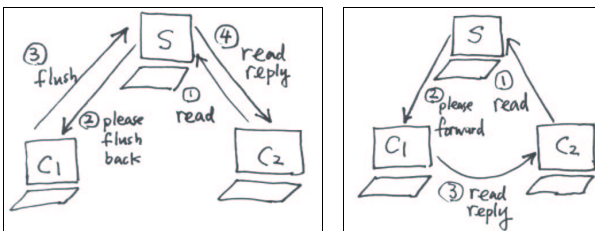
- Introduction
- Networked RAID
- **Cooperative client caching**
- Global memory
- Distributed everything in xFS
- Thoughts for future

CS518

17

Randy Wang

## Coop Caching Background



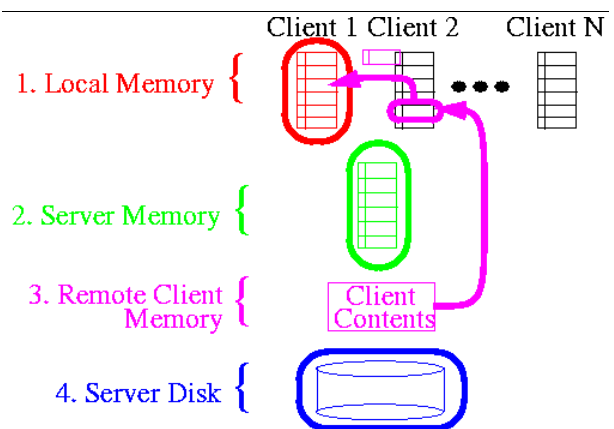
- Sprite lessons
  - Remote memory is fast compared to local disk
  - Cache hit rate surprisingly low
  - What's the role of the server?
    - Do you really want clients to flush back to server first?
    - The real server role is to keep track of who has what
    - Not just performance, but scalability implications, ...
- Main ideas of cooperative caching
  - Use peer client caches
  - Better coordination of client caches
  - Client to client transfers

CS518

18

Randy Wang

## Exploiting Peer Client Memory



- Introduce (NFS) server state to keep track of copies
- Issues: degree of replication, amount of central coordination, tradeoff local performance vs. global good

CS518

19

Randy Wang

## Coop Caching Algorithms

- “Prehistoric” -- base
  - No one helps or exploits anyone else
- “Slavery” -- direct client cooperation
  - Active clients enslave passive clients’ memory
- “Free-for-all capitalism” -- greedy forwarding
  - Read from other clients’ memory (exploitation without sacrifice)
- “Socialism” -- centrally coordinated caching
  - Each client gives up a portion of its memory that’s run by the “government”
- “Socialistic capitalism” -- N-chance forwarding
  - Clients read from others, but also accept “singlet” discards by others (exploitation with sacrifice)
- “Utopian communism” -- “best” case
  - Unrealistic case where each client gives up nothing but still gets free handout from “government”

## Coop Caching Conclusions

- Algorithm evaluations
  - N-chance as good as “best” and pretty simple
  - Centrally coordinated as good but depends more on the fast network due to poor local hits
  - Greedy is simplest and provides modest gain for “free”
  - Direct cooperation not very useful and hard to get right
- Verdict on cooperative caching
  - Improves read response time by as much as 73%
- Why not put everything on server?
  - Local memory is still a lot faster than remote memory
  - Don’t want to overload server
  - More flexible use of memory on clients
  - Cheaper to put small amounts of memory on some (or even many) clients

## Outline

- Introduction
- Networked RAID
- Cooperative client caching
- **Global memory**
- Distributed everything in xFS
- Thoughts for future

## Global Memory System (GMS)

- Benefits both VM and FS
- Central coordination not hard
- Central coordination necessary for corner cases
- De-centralized metadata management

## How to Obtain Approximate Global LRU Info

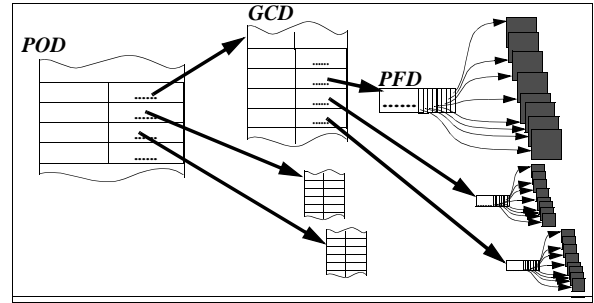
- Tax year
  - Define an “epoch”
- File a tax return
  - To start an epoch, each node reports local age info to an “initiator”
- Progressive tax rate
  - The initiator assigns a heavier “weight” to the node that has more older pages
- Rich cats pay more
  - During each epoch, heavy-weight nodes receive more page-outs
- Need to vary the length of epoch if nodes’ fortune change rapidly

CS518

24

Randy Wang

## How to Find Pages



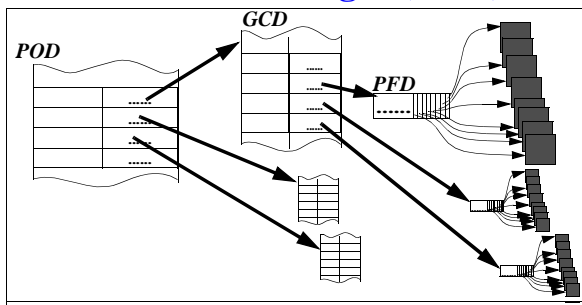
- PFD: page frame directory
  - local node data structure
  - maps logical pages to physical memory frames
- GCD: global cache directory
  - global data structure, distributed
  - maps logical pages to nodes that are currently storing the page
- POD: page ownership directory
  - global data structure, replicated
  - maps logical pages to GCD's

CS518

25

Randy Wang

## How to Find Pages (cont.)



- To find a page: POD -> GCD -> PFD
- To move a page: update GCD and PFD
- Add or remove nodes: change POD and redistribute GCD
- Try to keep most of the lookups local

CS518

26

Randy Wang

## Why the Trouble?

- Avoid centralized performance bottleneck
- Take advantage of locality
- Allow flexible (dynamic) reconfiguration to load balance and circumvent failures

CS518

27

Randy Wang

## Outline

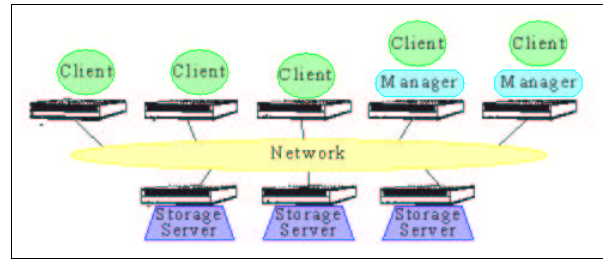
- Introduction
- Networked RAID
- Cooperative client caching
- Global memory
- **Distributed everything in xFS**
- Thoughts for future

CS518

28

Randy Wang

## Sample Configuration



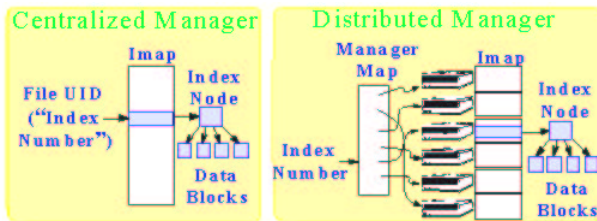
- Clients
- Storage servers
- Managers
- Cleaners
- (How's this different from Zebra?)

CS518

29

Randy Wang

## Distributed Metadata



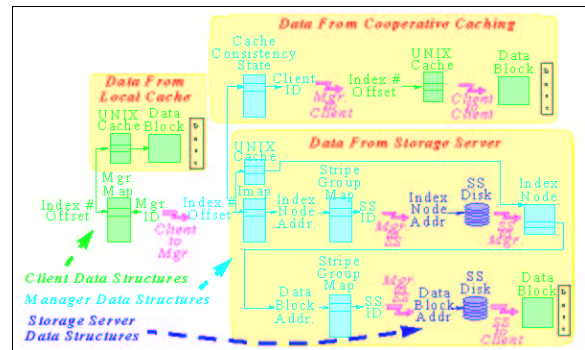
- Manager map (globally replicated)
  - Maps file "index number" to manager
  - Changed (infrequently) for load balancing or cluster reconfiguration
  - Like GMS's POD
- Imap (split across managers)
  - Maps index number to inode SS address
  - Like GMS's GCD
- Storage servers (local data structure)
  - Maps SS address to physical disk address
  - Like GMS's PFD
- This is for disks, similar story for memory cache

CS518

30

Randy Wang

## Example: Read



- (Yikes!)
- Three main paths
- Common case: avoid disks, and few net hops

CS518

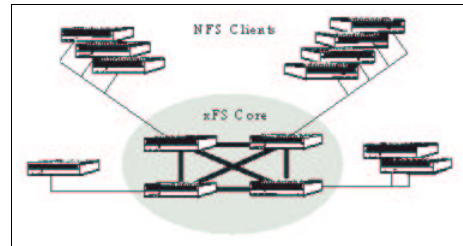
31

Randy Wang

## Recovery

- A combination of Zebra and Sprite
  - Logging
  - Checkpointing
  - Asking clients to reconstruct lost state
- Weaknesses
  - In general, potentially require “ $O(N^2)$ ” messages
  - Client-driven (or server-pull) reconstruction is hard to get right
  - In general, too complex
  - What’s the impact on a large scale system? Failure operation is the norm?

## Security

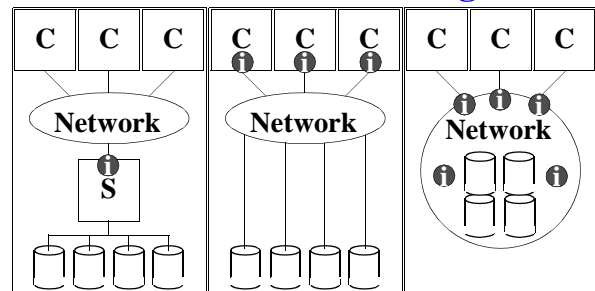


- Using xFS as a scalable server is a cool and potentially more practical idea
- Precursor of “NASD”
  - Two different networks
  - One secure, and the other not
  - Storage servers provide dumb block interface
  - The rest of the core implements a particular file system
  - If the core machine that takes the request is different from the core machine that supplies the reply, what do we have?

## Outline

- Introduction
- Networked RAID
- Cooperative client caching
- Global memory
- Distributed everything in xFS
- Thoughts for future

## Where to Place the Intelligence?



- Client/server (intelligence in the server)
  - Performance, scalability, and availability bottleneck
- Serverless systems (intelligence on clients)
  - Hard to deal with client complexity, heterogeneity, configuration and reconfiguration, info collection from other clients, and dynamic adaptation (sort of like source routing)
- Moving intelligence into the net