Distributed Storage Systems 2

Juncheng Yang

Feb 23

Harvard John A. Paulson
School of Engineering
and Applied Sciences

HARVARD UNIVERSITY
MADSYS
Measurement and Design of Systems Lab

Agenda

- Cluster file system
  - MooseFS, GlusterFS, BeeGFS
  - DeepSeek 3FS
- Distributed block storage
- (Distributed) object storage
- Evolution of distributed storage systems
- Distributed data structures

We do not go into details on consistency in this course, more details will be discussed in a distributed system class.

Key Questions to Think About After Class

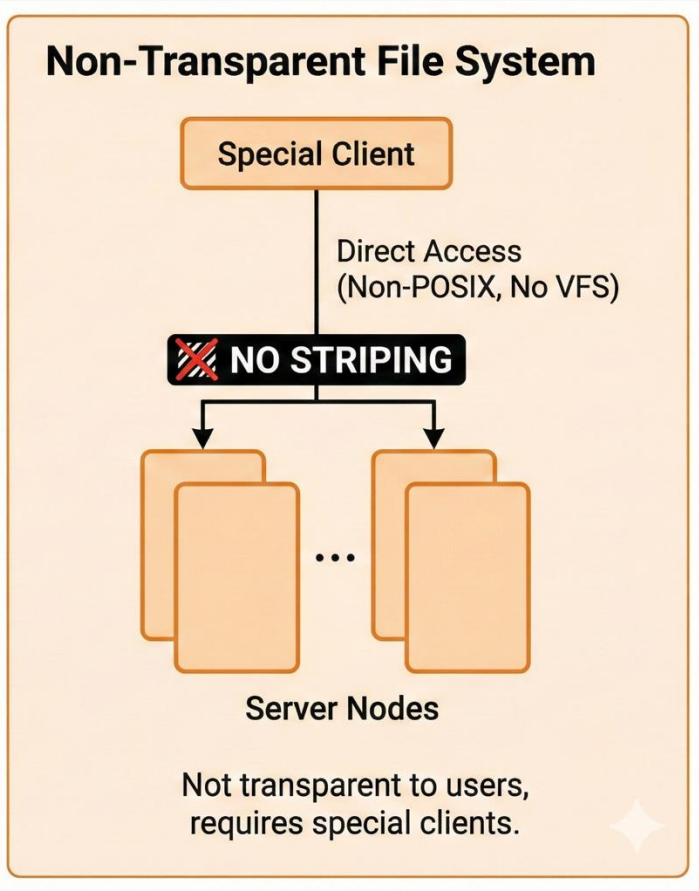
- What are the common design components and patterns of a distributed file system?
- Can you compare distributed block storage with distributed file systems?
- How is object storage different from distributed file system? Why is it more scalable? What are the limitations?

Cluster File Systems

- MooseFS
- GlusterFS
- BeeGFS

GFS and HDFS

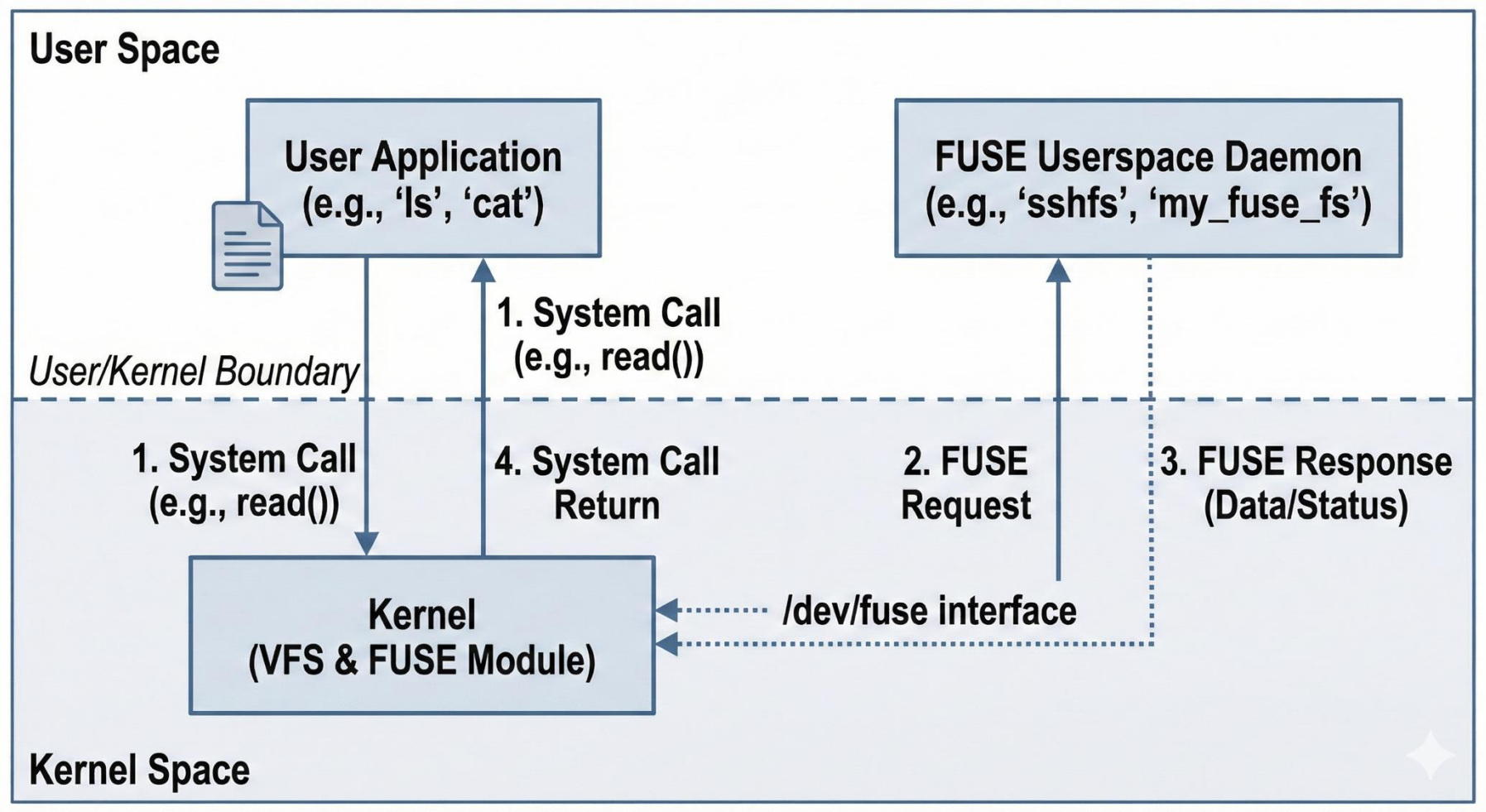
- Not a file system transparent to users
  - requires special clients
  - not hooked into VFS, no POSIX-compliance
  - does not stripe across servers



MooseFS

- An open-source implementation of GFS
  - metadata in master DRAM (with a log on disk)
  - single master (can have an active standby)
- Meta-logger
  - asynchronous replicate the metadata changes
  - can be promoted to master manually
- POSIX-compliant using a FUSE client
  - FUSE: **F**ilesystem in **Use**rspace
  - FUSE kernel module calls userspace filesystem code

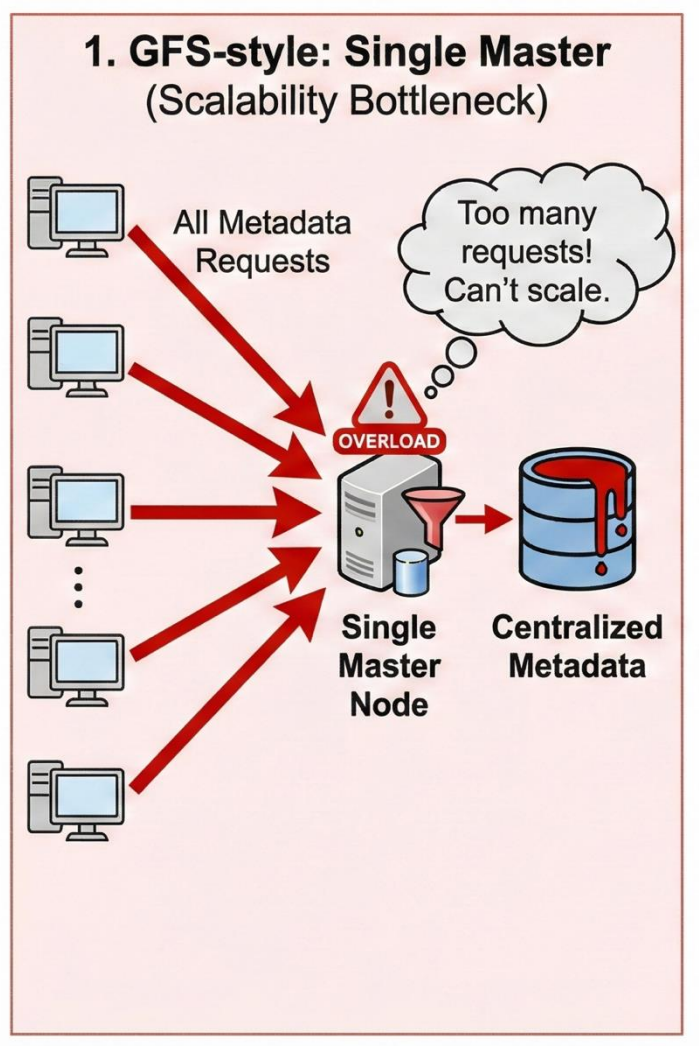
FUSE



- simpler, safer, easier to implement
- performance overhead: context switch

GlusterFS

- GFS: single-master scalability bottleneck
- Replace metadata server with distributed hash table (DHT)
  - calculate the hash of a path and find the server id
  - benefits: simple, fast
- **Amazing idea, why hasn't others used it?**
- What problems does DHT cause?
  - rename a file will need to move data
    - solution: linkto file that redirects the requests
  - directory listing requires broadcast
    - limits cluster size



GlusterFS

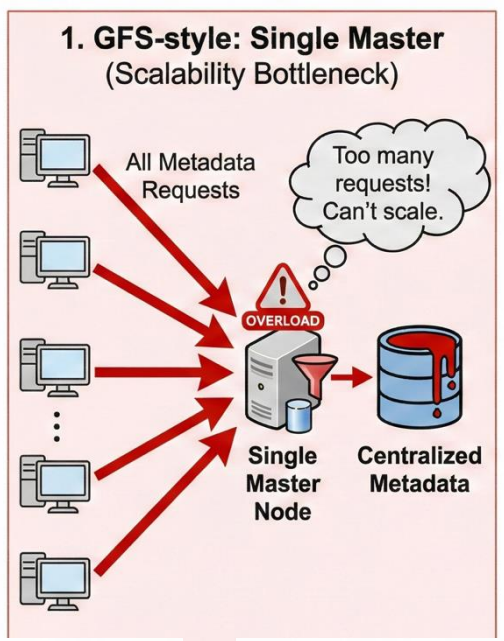
- Translator Stack Architecture
  - similar to device mapper, but in file system
  - request flows through multiple transformation layers
    - FUSE interface
    - DHT: routing
    - AFR (automatic file replication): for replicated volume
    - performance translator: caches
    - protocol: TCP or RDMA

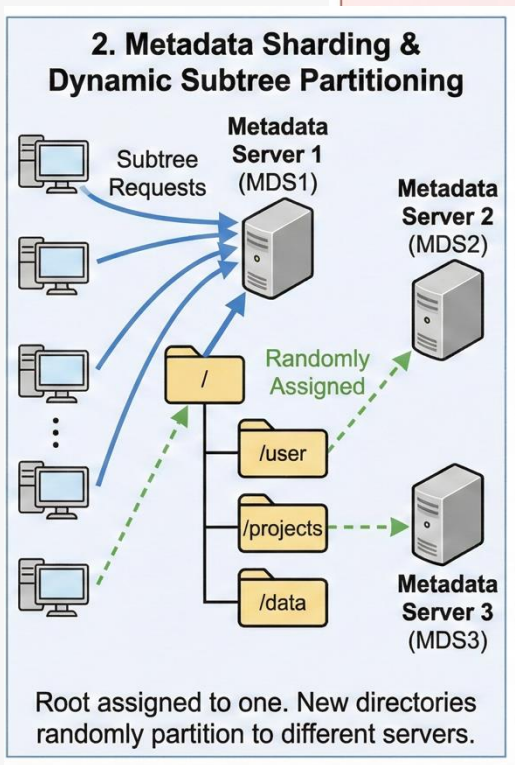
BeeGFS

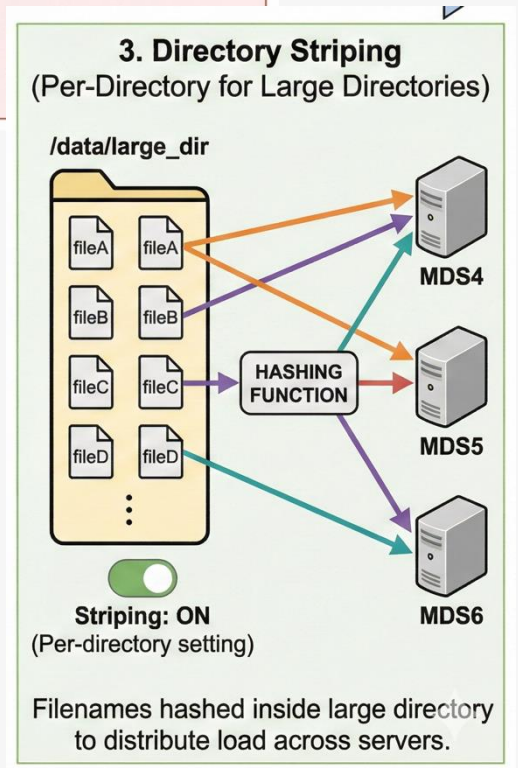
- Parallel file system for HPC/AI workloads
  - design goal: maximum bandwidth
- Four components
  - management service
    - DNS, maintain the node list, not involved in I/O operations
  - metadata service
  - storage service
  - client

BeeGFS: Distributed Metadata

- dynamic subtree partitioning
  - root (/) is assigned to one metadata server
  - each new directory is (randomly) assigned to a different server
- directory striping
  - solve the problem when a directory has too many files
  - hash the filenames and assign to different servers
  - can be turned on per directory







BeeGFS: Performance-Oriented Designs

- Data Striping
  - split files into chunks (e.g., 1MB) and distribute across servers
  - dynamic striping
    - per directory or per file
    - small files (e.g., source code): target=1
    - large files (e.g., model checkpoint): target=100
- Heavily optimized for RDMA
  - RDMA: remote direct memory access
  - zero copy, server directly write data into client's DRAM
    - fast, save CPU cycles
  - two-sided RDMA for control messages, one-sided RDMA for data movement

BeeGFS: Buddy Mirroring

- Buddy group
  - paired nodes (normally 2)
  - storage buddy, metadata buddy
- Buddy mirroring for high availability (HA)
  - synchronous replication
  - if primary is unreachable, the secondary takes over
  - can enable per directory

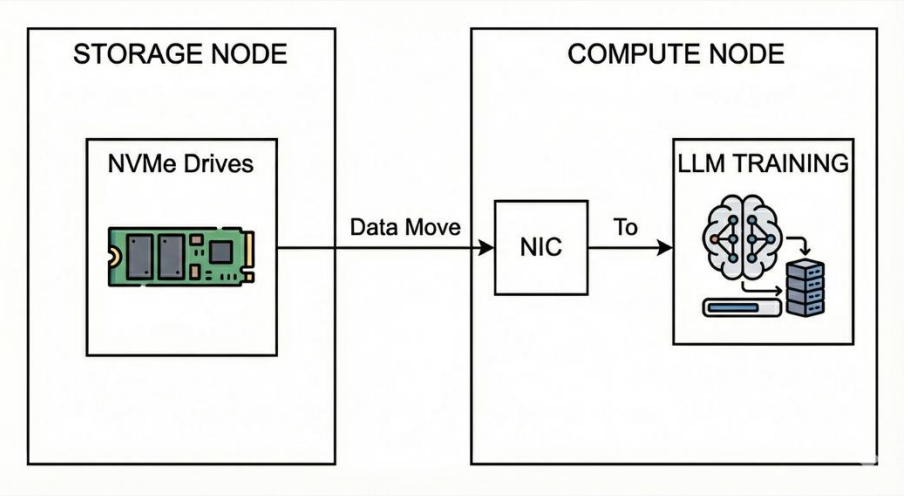
BeeGFS: Request Flow

- Namespace lookup/open
  - client contacts the metadata node responsible for the directory/file metadata
- Layout discovery
  - metadata informs the client of the file’s stripe pattern (chunk size, number of targets, pool, mirroring mode, etc.)
- Data transfer
  - client sends reads/writes directly to the storage servers holding the relevant chunks, often to multiple servers concurrently

Modern Distributed File Systems

- DeepSeek 3FS

Deepseek 3FS: Why Is It Fast?



- Optimize for a specific workload
  - read training data (random read)
  - write checkpoints
- Although it comes with a FUSE client, high bandwidth requires native client
  - sacrifice some file system features
  - no read caching and prefetching (random reads)
  - kernel bypass: no data copy
    - set up a shared memory region between application and NIC
    - data move from storage node NVMe drives to compute node NIC to application

Deepseek 3FS: Why Is It Fast?

- CRAQ (Chain Replication with Apportioned Queries)
  - write path
    - flow linearly down a chain of nodes (three nodes)
    - strong consistency: acknowledge only when the last node commits
  - read path
    - high IOPS/bandwidth: read from any of the node
  - trade capacity for performance
- Comparison with the chained replication in GFS
  - GFS: client sends data to the closest server, primary serialize reads, reads always go to primary
  - 3FS: client writes to a pre-defined list of server, read from any server

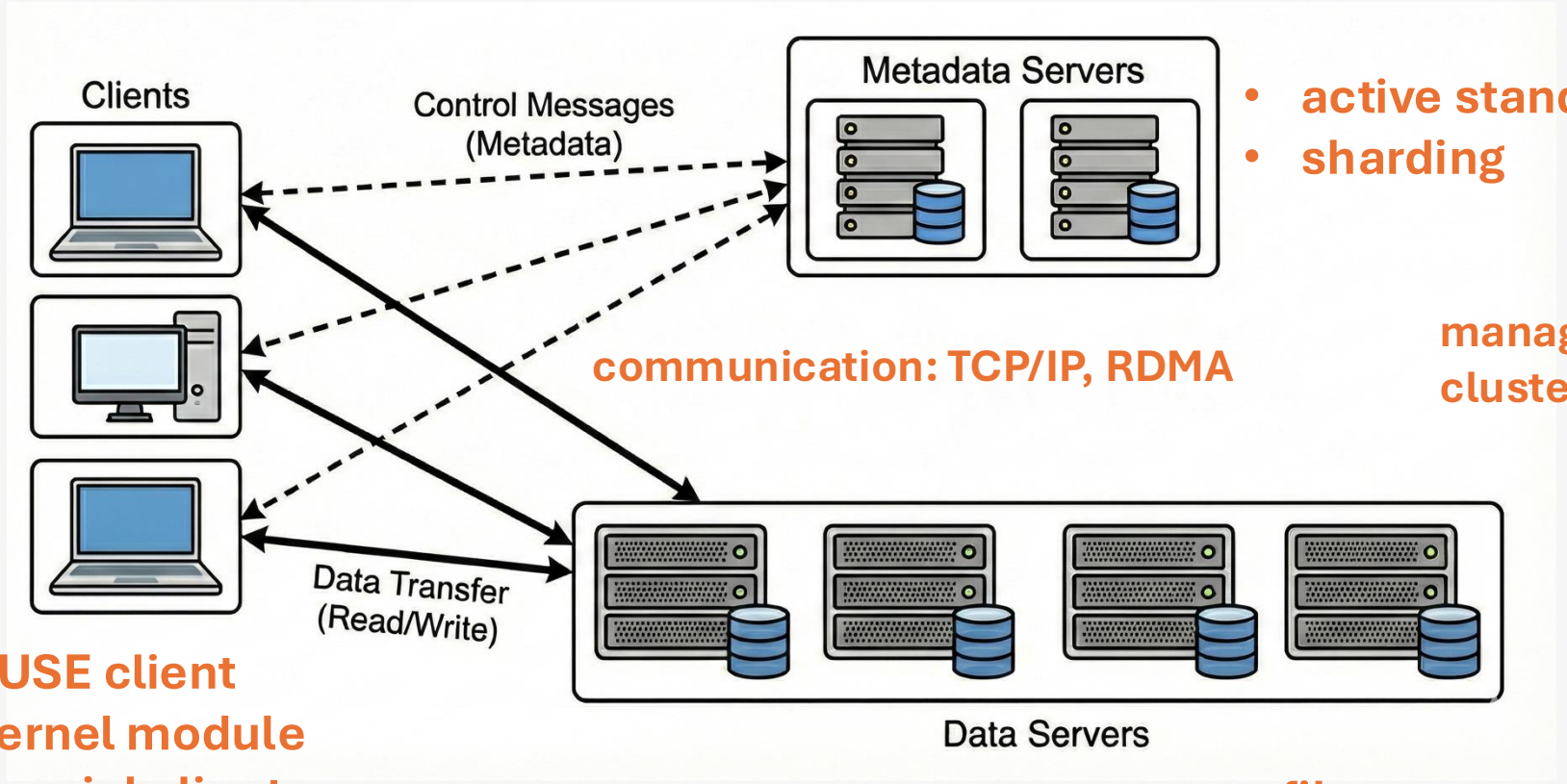
Deepseek 3FS: Why Is It Fast?

- Metadata
  - stored in FoundationDB (in a separate cluster)
    - transaction / atomic support, e.g., create, mv, rename
    - stateless metadata server that can be scaled up easily
  - flat namespace with directory
    - file Inode is stored as a key-value entry
    - each directory has an Inode stored as a key-value pair
    - O(1) path resolution
    - fast directory list (`ls`) using range scans

Summary

- NFS, AFS:
  - a remote file system where one server is responsible for both data and metadata
- GFS, HDFS:
  - separate metadata and data and store metadata in DRAM
  - require special client and not compatible with VFS
- MooseFS: GFS with a POSIX-compliant file system
- BeeGFS:
  - separate management from metadata server
  - distributed metadata server
- GlusterFS:
  - removing metadata server bottleneck using DHT, but introduce broadcast storm
- Deepseek 3FS: kernel bypass, flat namespace with directory, using database for metadata

Designing Distributed File Systems



- active standby
- sharding

manager:
cluster topology

- FUSE client
- kernel module
- special client

- file system
- raw block device

Design for your use case, workload and hardware

Where We Are and Where We Will Go Next

- Goals:
  - capacity: **solved**
  - durability: **solved**

  } **replication, erasure coding**
  - reliability: **solved (distributed metadata, backup)**
  - performance: **partially solved**
    - latency: **worse**
    - bandwidth, IOPS: **improved (limited)**
    - metadata performance: **vary**
  - sharing: **partially solved**

trade POSIX-compliance for it

stronger consistency guarantee

Distributed Block Storage

- Typical use cases: VM disk
- EBS

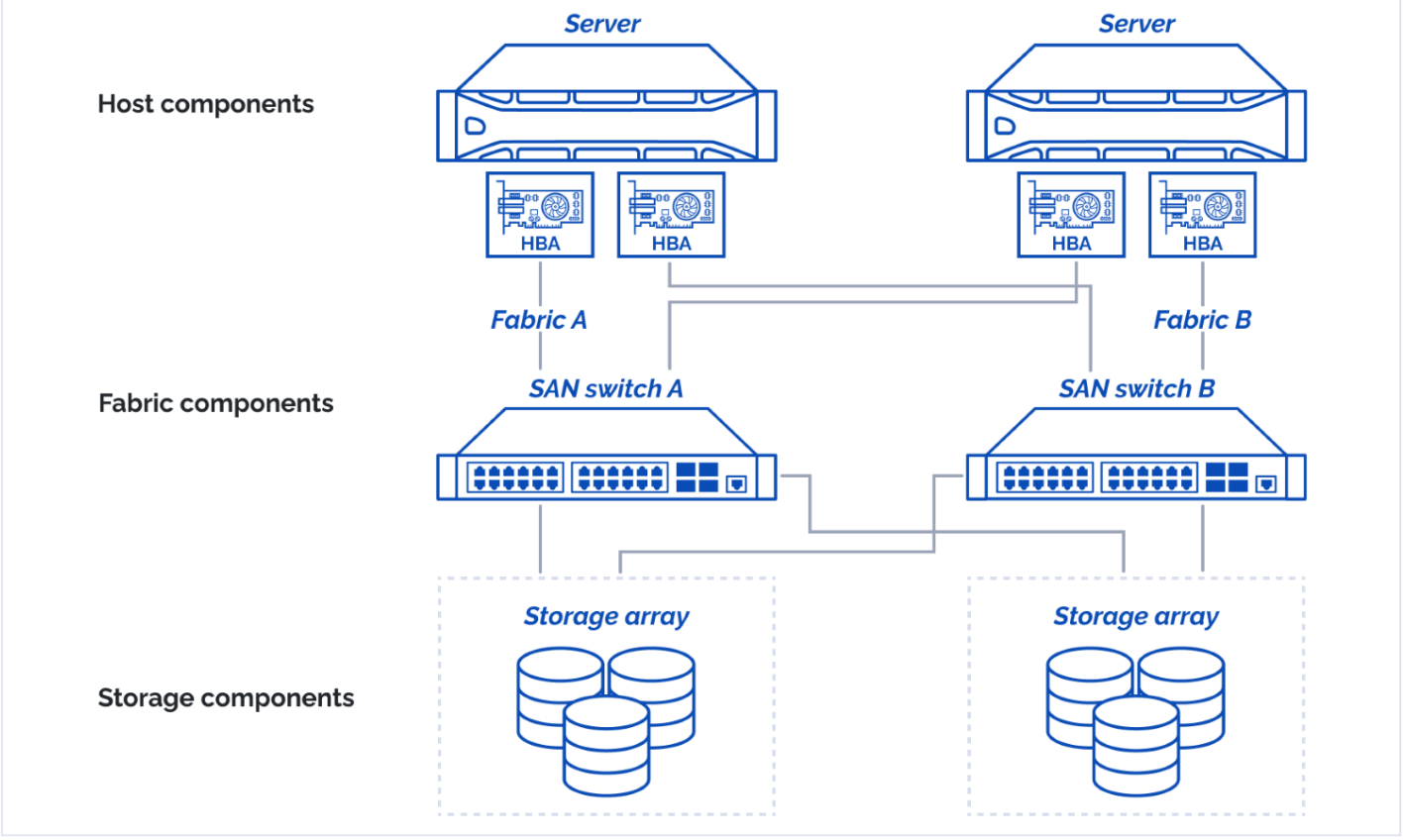
Distributed Block Storage vs. Distributed File Systems

- Exposed as a block device to users
- It is actually simpler
  - simple mapping from (Volume, LBA) to a physical location
  - no complex operations (rename, move…)
  - minimal metadata (no permission, timestamps, references…)
  - assume single writer (or a higher-level coordination layer)
- Common challenges
  - data placement and load balancing
- Key unique challenges for distributed block storage
  - random read/write performance
  - durability barrier, write ordering (if copies are stored on different servers)

Storage Area Network (SAN)

- The parent of modern distributed block storage
- Three layers
  - **host** layer: servers equipped with host bus adapters (HBAs)
  - **fabric** layer: SAN switches that route data between hosts and storage
  - **storage** layer: storage arrays containing many disks (HDD/SSD)
- Network protocols
  - **fibre channel** (FC): old standard, dedicated fiber optics (or copper cable) for low latency
  - **iSCSI**: SCSI commands in Ethernet packets, cheaper but has higher CPU overhead
  - **NVMe over Fabrics** (NVMe-oF): new standard designed for NVMe

Storage Area Network (SAN)

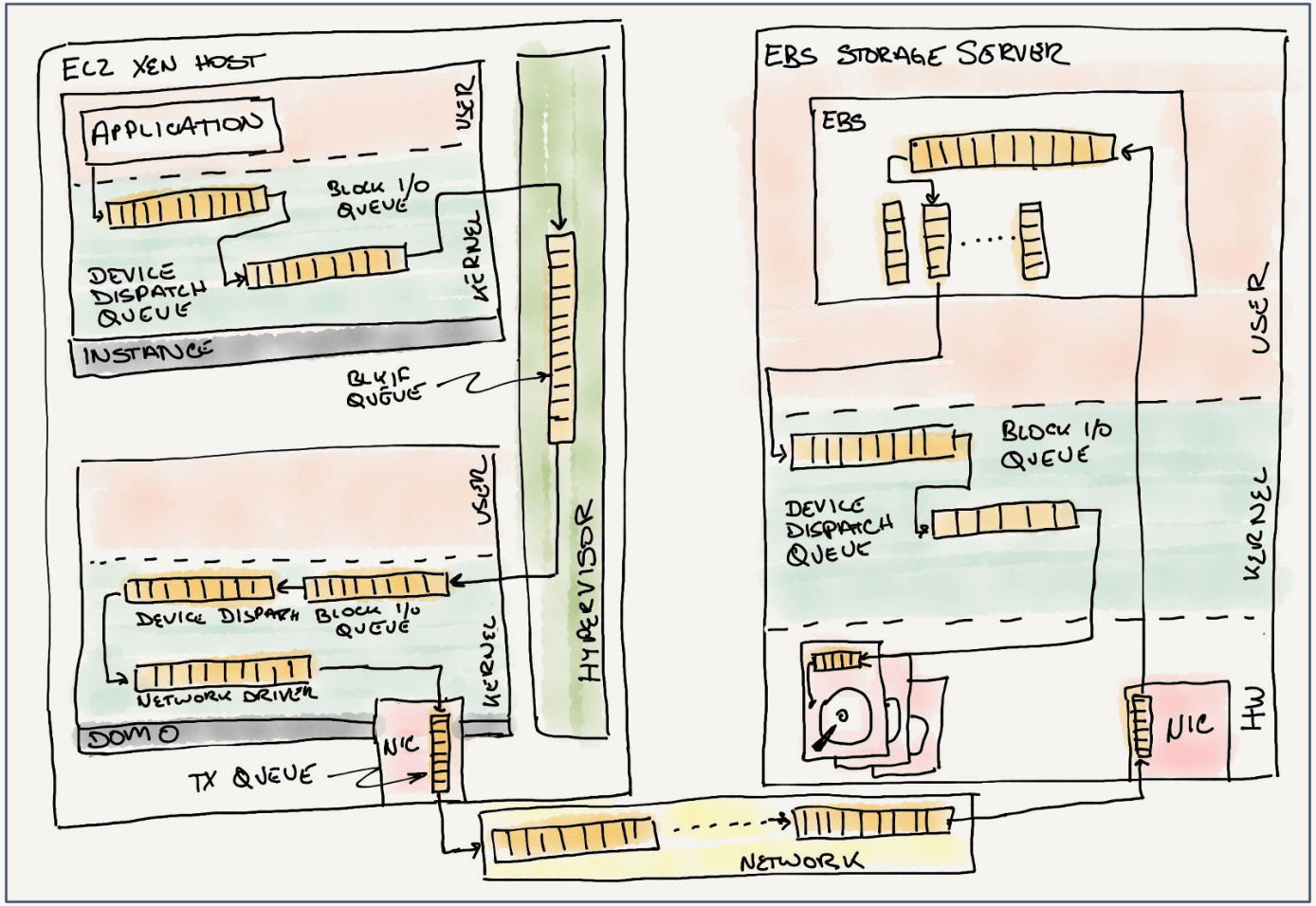


Core components of SAN

https://www.starwindsoftware.com/blog/what-is-san-storage-area-network-how-does-it-work

Distributed Block Storage

- Commodity hardware
  - server and network
- Managed using software
  - sharding
  - replication
  - load balancing
- Scale out



AWS EC2 EBS 2012

Continuous reinvention: A brief history of block storage at AWS

Distributed Block Storage (AWS EBS)

- Internal sharding & distribution
  - every volume is broken into smaller chunks and spread across many storage nodes
- Automatic I/O load balancing
  - reads/writes are dynamically routed and balanced to avoid hotspots
- Synchronous replication
  - every write goes to multiple replicas in real time
- Hardware offloading and new network protocol
  - nitro card: encryption, I/O processing, network encapsulation
  - Scalable Reliable Datagrams (SRD): multi-path, out-of-order delivery, fast retransmission

Physalia: Millions of Tiny Databases (NSDI 2020)

Distributed Block Storage vs. SAN

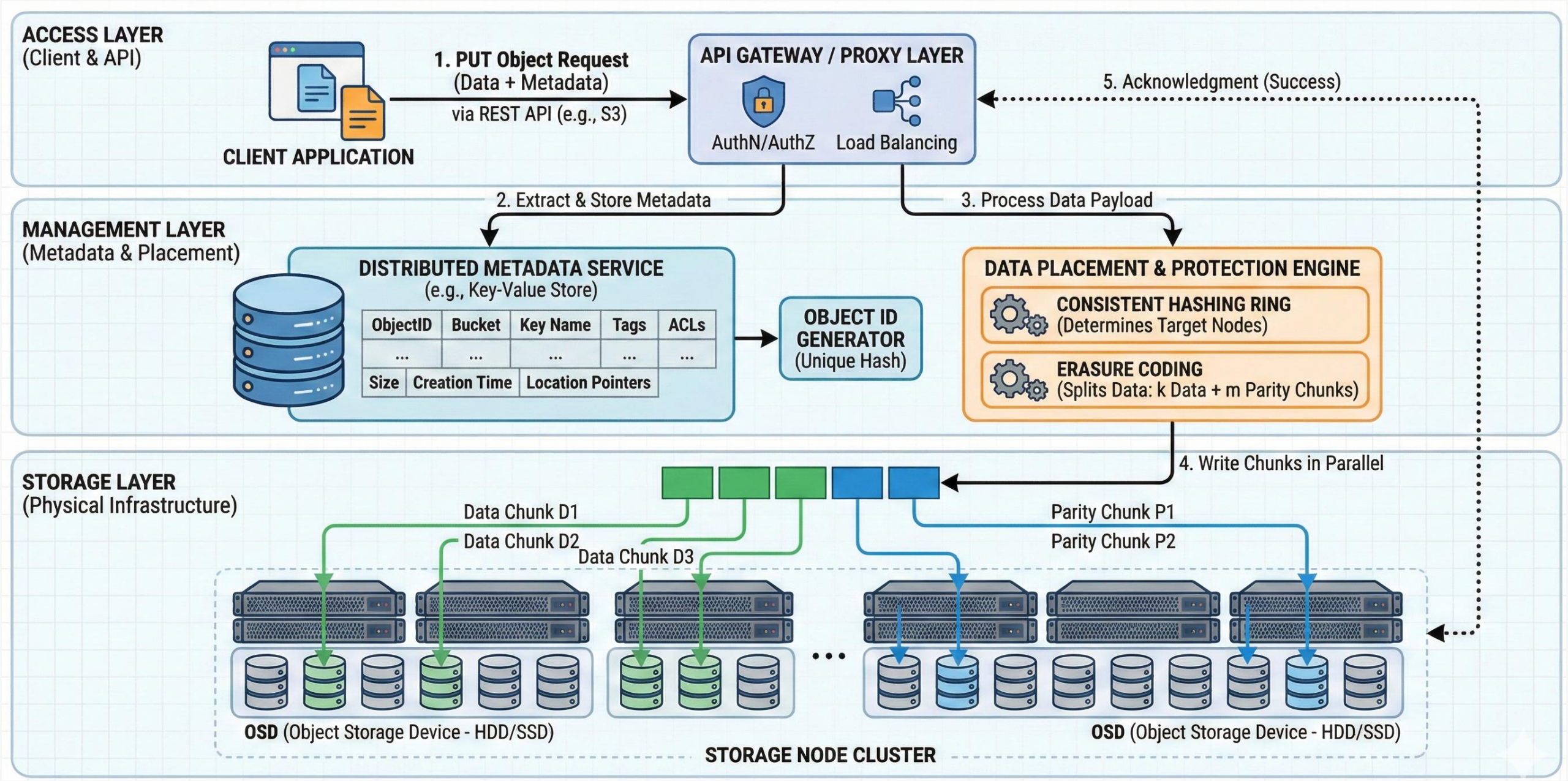
Feature	Traditional SAN	Distributed Block Storage
Scalability		
Hardware		
Fault Tolerance		
Latency		
Complexity		

(Distributed) Object Storage

- AWS S3

Object Storage

- Object: key + metadata + data
  - key: bucket + name, e.g., “s3://harvard/cs2640_lecture1_slides.pdf”
  - rich metadata: highly customizable set of tags describing the data
  - immutable: suitable for some content types
    - media, data lake, backup
    - not latency-sensitive, but rather bandwidth-hungry
- Flat namespace
  - no recursive directory lookup
- RESTful API
  - GET, PUT, DELETE, LIST


ACCESS LAYER
(Client & API)
CLIENT APPLICATION
1. PUT Object Request
(Data + Metadata)
via REST API (e.g., S3)
API GATEWAY / PROXY LAYER
AuthN/AuthZ
Load Balancing
5. Acknowledgment (Success)
2. Extract & Store Metadata
3. Process Data Payload
MANAGEMENT LAYER
(Metadata & Placement)
DISTRIBUTED METADATA SERVICE
(e.g., Key-Value Store)
ObjectID
Bucket
Key Name
Tags
ACLs
Size
Creation Time
Location Pointers
OBJECT ID GENERATOR
(Unique Hash)
DATA PLACEMENT & PROTECTION ENGINE
CONSISTENT HASHING RING
(Determines Target Nodes)
ERASURE CODING
(Splits Data: k Data + m Parity Chunks)
4. Write Chunks in Parallel
STORAGE LAYER
(Physical Infrastructure)
Data Chunk D1
Data Chunk D2
Data Chunk D3
Parity Chunk P1
Parity Chunk P2
OSD (Object Storage Device - HDD/SSD)
OSD (Object Storage Device - HDD/SSD)
STORAGE NODE CLUSTER

Object Storage: Benefits

- Infinite scalability
  - no centralized metadata server on read path: hash(key) => data location
  - scale-out by adding more servers
  - caveat: rename penalty
- Cost efficiency
  - most object stores use HDDs (focus on bandwidth)
- Rich metadata
  - directly query without a database

Object Storage: Use Cases

- Media storage and delivery
  - cheap, immutable
- Binary artifact (e.g., OS image)
  - cheap, immutable
- Data lake, analytics and AI/ML pipeline
  - massive bandwidth, cheap, metadata query
- Backup and long-term retention
  - durable, cheap, immutable
- User-generated content
  - infinite scalability

Comparing Different Distributed Storage Systems

Feature	Distributed block storage	Distributed file systems	Object storage
Organization			
Access Method			
Performance			
Scalability			
Best For			

Think: for the data you are working with, what is the best storage systems?

Evolution of Distributed Storage Systems

Distributed Storage Evolution

- Move from "one big box" to millions of small boxes working as one
  - driven by data grew faster than the hardware improvement
- The Pre-Distributed Era (1960s–1980s): centralized
  - mainframe, direct-attached storage (DAS)
- The Networked Era: NAS and SAN (1980s–1990s)
  - motivation: many computers share storage
  - NFS, AFS, NAS (file-level), SAN (block-level)
  - single controller that does not scale

Distributed Storage Evolution

- The Big Data Era: GFS and HDFS (early 2000s)
  - motivation: shifting to commodity hardware (failure is the norm)
  - separating metadata into separate server with relaxed POSIX compliance
  - software redundancy for durability and availability
- The Cloud Era: Object Storage (Mid 2000s–2020s)
  - motivation: media-heavy, hierarchical folder is too slow
  - infinite *scalability*, cheap but high bandwidth, multi-tenancy

Distributed Storage Evolution

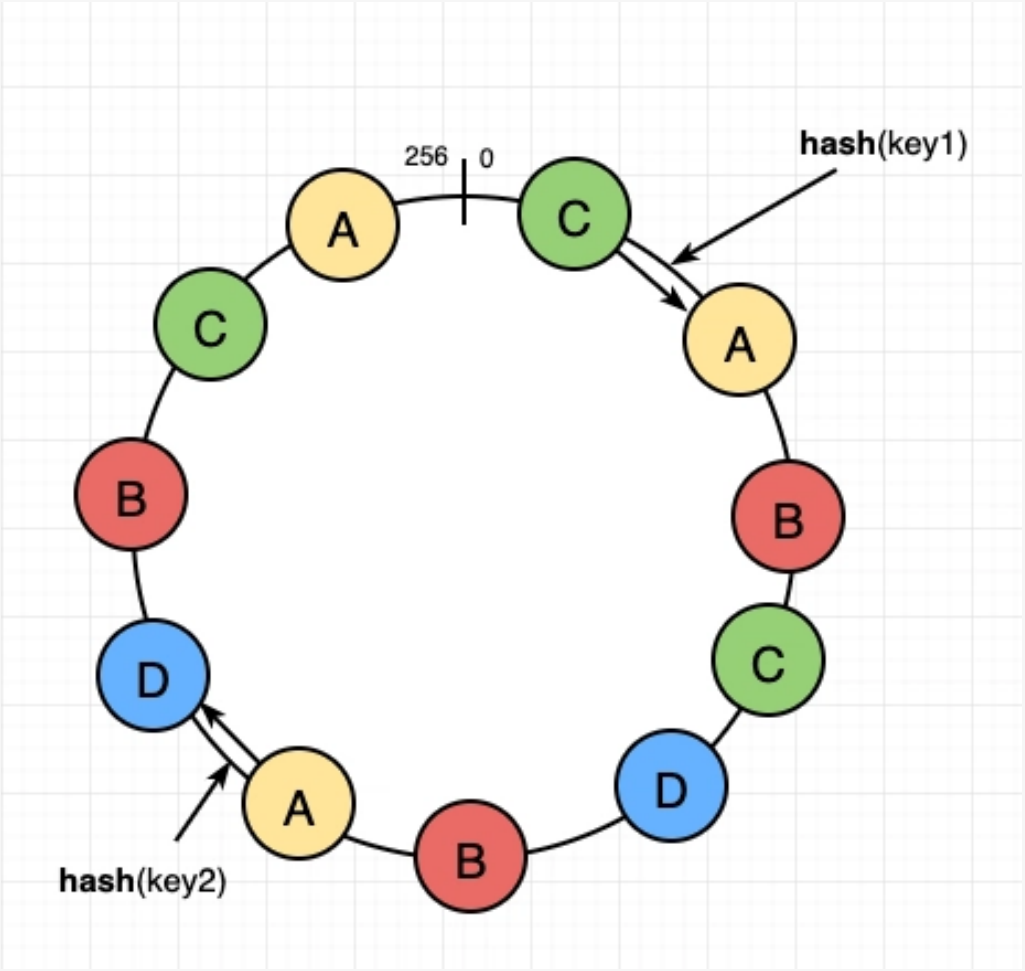
- The Modern Era: Disaggregated Storage (2010s–Present)
  - motivation: independent scaling of compute and storage
  - maintain high performance
    - high-bandwidth network, RDMA
    - user-space I/O, hardware offloading
  - reduce operational overhead
    - one substrate, many interfaces
  - reduce cost
    - from replication to erasure coding

Other Distributed Data Storage

- Distributed Hash Table (DHT)
- Distributed Message Queue
- Distributed Log
- Distributed Data Structures
- Conflict-free Replicated Data Types (CRDT)

Distributed Hash Table

- Hash table distributed on many nodes
- Consistent hashing
  - map each node to multiple positions
  - add nodes: keys are re-assigned evenly from other nodes
- Often used to build distributed key-value store
- Example: NoSQL database



Distributed Message Queue

- Allow services to communicate asynchronously
- Producer, broker (queue), consumer
- Distributed: replication, partitioning
- Delivery guarantee
  - at-most-once: suitable for non-critical data (log)
  - at-least-once: require deduplication
  - exactly-once: expensive to implement
- Common systems: Apache Kafka

Distributed Log

- An append-only, ***totally ordered*** sequence of records stored across multiple machines
- Bedrock of many distributed databases
  - state machine replication (SMR): two deterministic state machine will end up in the same state if given the exact same input
  - distributed log as the ground truth for the input
  - immutable: easy for replication and consensus
  - fast crash recovery
  - time travel and auditing

Distributed Data Structure (Redis)

- **Redis: RE**mote **DI**ctionary **S**erver
  - open-source in-memory data structure store
  - new popular fork: Valkey
- Core data structures
  - string: key-value store
  - list
  - hash: hash map
  - set: unique collection
  - sorted set
- Shard key space
  - hash key to slot
  - slot to server mapping is stored in every node, updated using gossip protocol

Conflict-free Replicated Data Type (CRDT)

- A data structure replicated across machines that enables safe concurrent updates
  - example: counter, set
- Three key properties
  - independent update
    - any replica can be updated locally and concurrently without coordinating with other replicas
  - automatic conflict resolution
    - the data type includes rules that deterministically reconcile divergent updates
  - eventual convergence
    - replicas may temporarily differ, but are guaranteed to converge to the same state
- Real-world use case: collaborative editing

Summary

- Cluster file system
  - BeeGFS, MooseGFS, GlusterFS
  - DeepSeek 3FS
- Distributed block storage
- (Distributed) object storage
- Evolution of distributed storage systems
- Distributed data structures

Next time

- Storage system performance