Cloud storage (II) — Google (cont.), Microsoft, Facebook

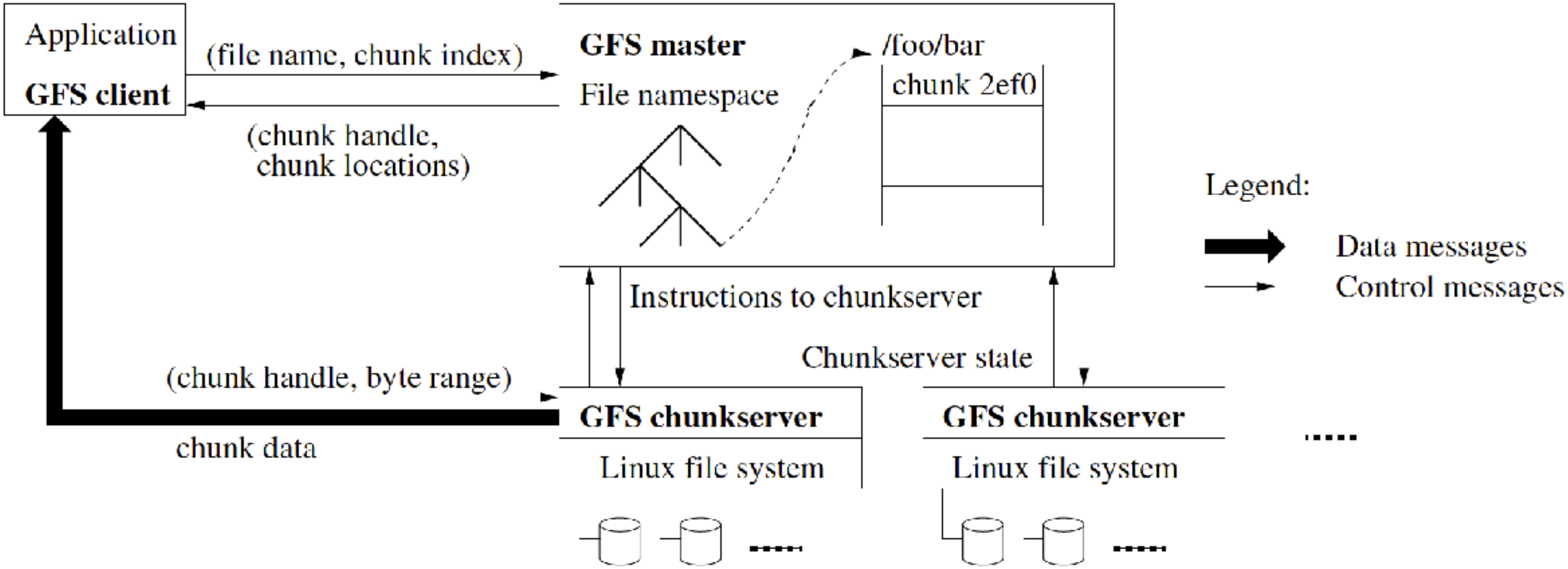
Hung-Wei Tseng

Outline

- Google File System (cont.)
- Windows Azure Storage: A Highly Available Cloud Storage Service with Strong Consistency
- f4: Facebook's Warm BLOB Storage System

Distributed architecture

decoupled data and control paths —
only control path goes through master

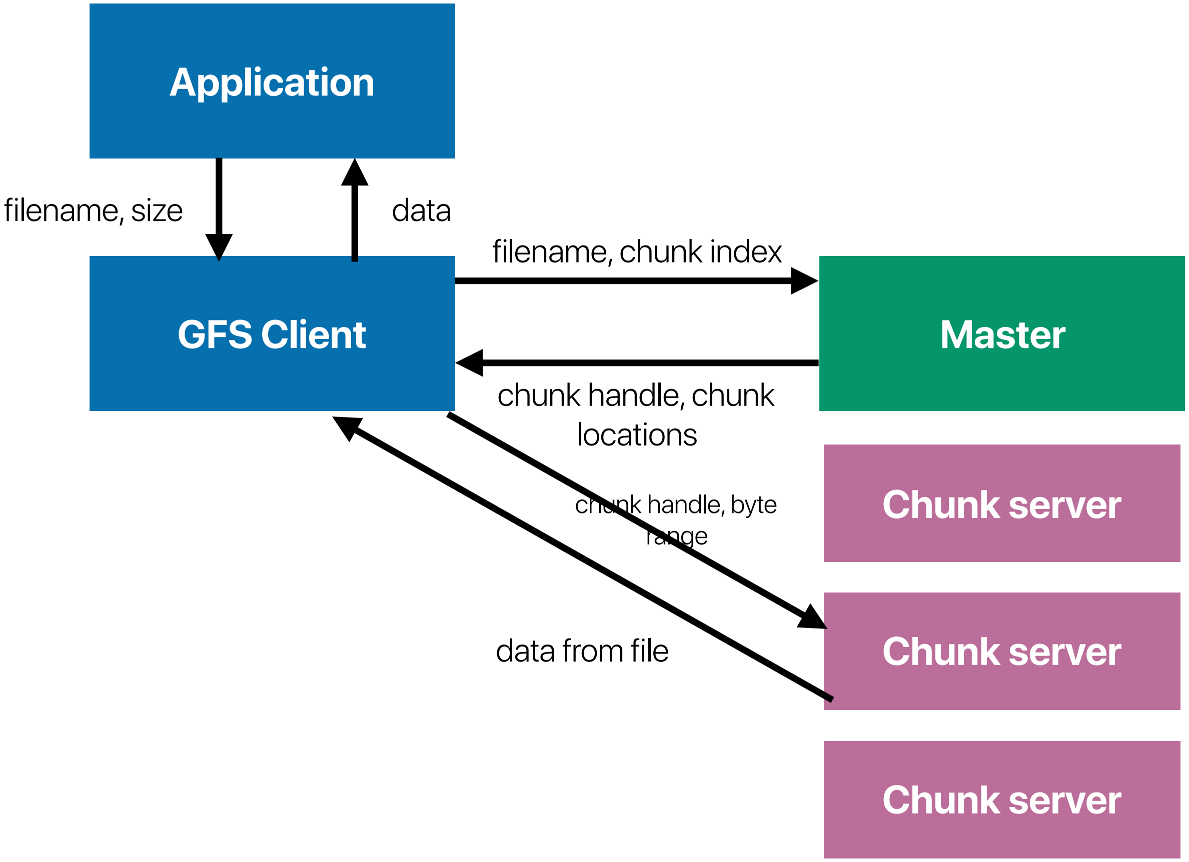


load balancing, replicas among chunkservers

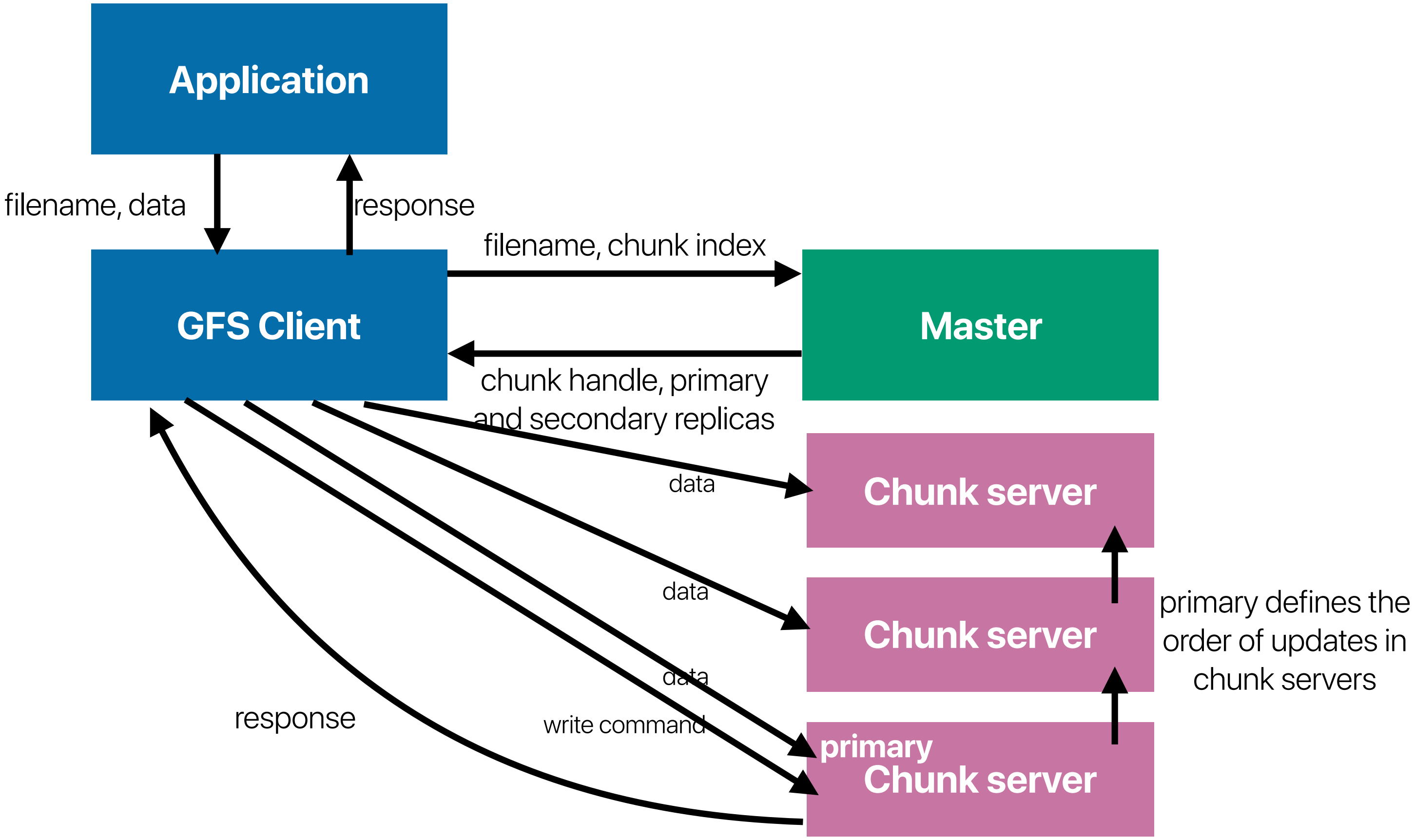
Distributed architecture

- Single master
  - maintains file system metadata including namespace, mapping, access control and chunk locations.
  - controls system wide activities including garbage collection and chunk migration.
- Chunkserver
  - stores data chunks
  - chunks are replicated to improve reliability (3 replicas)
- Client
  - APIs to interact with applications
  - interacts with masters for control operations
  - interacts with chunkservers for accessing data
  - Can run on chunkservers

Reading data in GFS



Writing data in GFS



GFS: Relaxed Consistency model

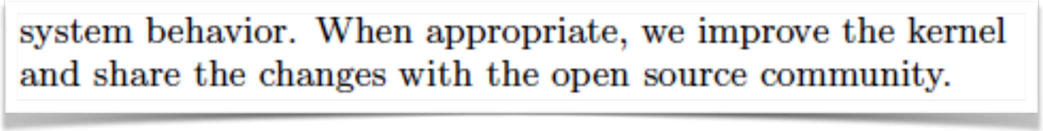
- Distributed, simple, efficient
- Filename/metadata updates/creates are atomic
- Consistency modes

	Write — write to a specific offset	Append — write to the end of a file
Serial success	Defined	Defined with interspersed with inconsistent
Concurrent success	Consistent but undefined	
Failure	inconsistent	

  - Consistent: all replicas have the same value
  - Defined: replica reflects the mutation, consistent
- Applications need to deal with inconsistent cases themselves

Real world, industry experience

- Linux problems (section 7)
  - Linux driver issues — disks do not report their capabilities honestly
  - The cost of fsync — proportion to file size rather than updated chunk size
  - Single reader-writer lock for mmap
  - Due to the open-source nature of Linux, they can fix it and contribute to the rest of the community
- **GFS is not open-sourced**



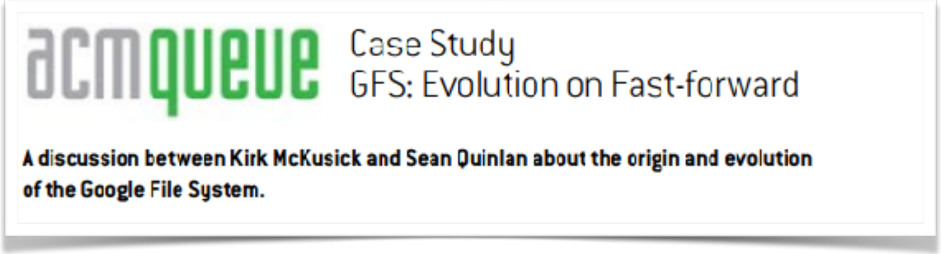
system behavior. When appropriate, we improve the kernel and share the changes with the open source community.

Single master design

- GFS claims this will not be a bottleneck
- In-memory data structure for fast access
- Only involved in metadata operations — decoupled data/control paths
- Client cache
- What if the master server fails?

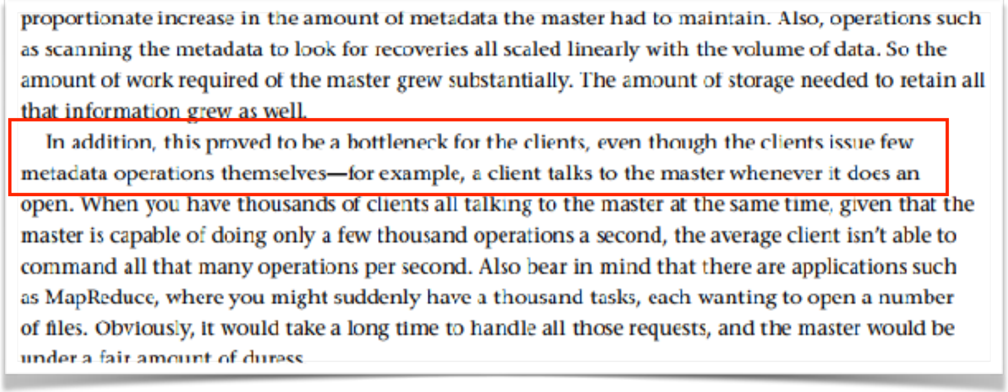
The evolution of GFS

- Mentioned in “Spanner: Google's Globally-Distributed Database”, OSDI 2012 — “tablet’s state is stored in set of B-tree-like files and a write-ahead log, all on a distributed file system called Colossus (the successor to the Google File System)”
- Single master



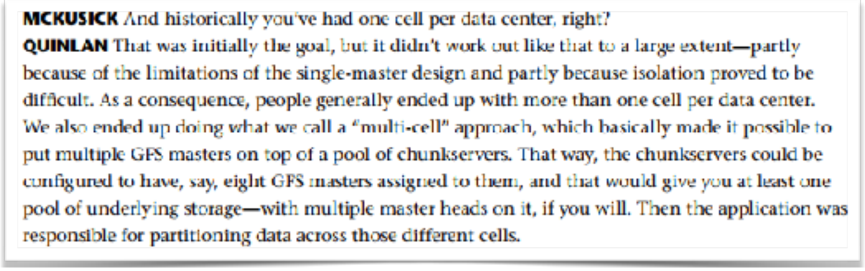
Case Study
GFS: Evolution on Fast-forward

A discussion between Kirk McKusick and Sean Quinlan about the origin and evolution of the Google File System.



proportionate increase in the amount of metadata the master had to maintain. Also, operations such as scanning the metadata to look for recoveries all scaled linearly with the volume of data. So the amount of work required of the master grew substantially. The amount of storage needed to retain all that information grew as well.

In addition, this proved to be a bottleneck for the clients, even though the clients issue few metadata operations themselves—for example, a client talks to the master whenever it does an open. When you have thousands of clients all talking to the master at the same time, given that the master is capable of doing only a few thousand operations a second, the average client isn’t able to command all that many operations per second. Also bear in mind that there are applications such as MapReduce, where you might suddenly have a thousand tasks, each wanting to open a number of files. Obviously, it would take a long time to handle all those requests, and the master would be under a fair amount of duress.

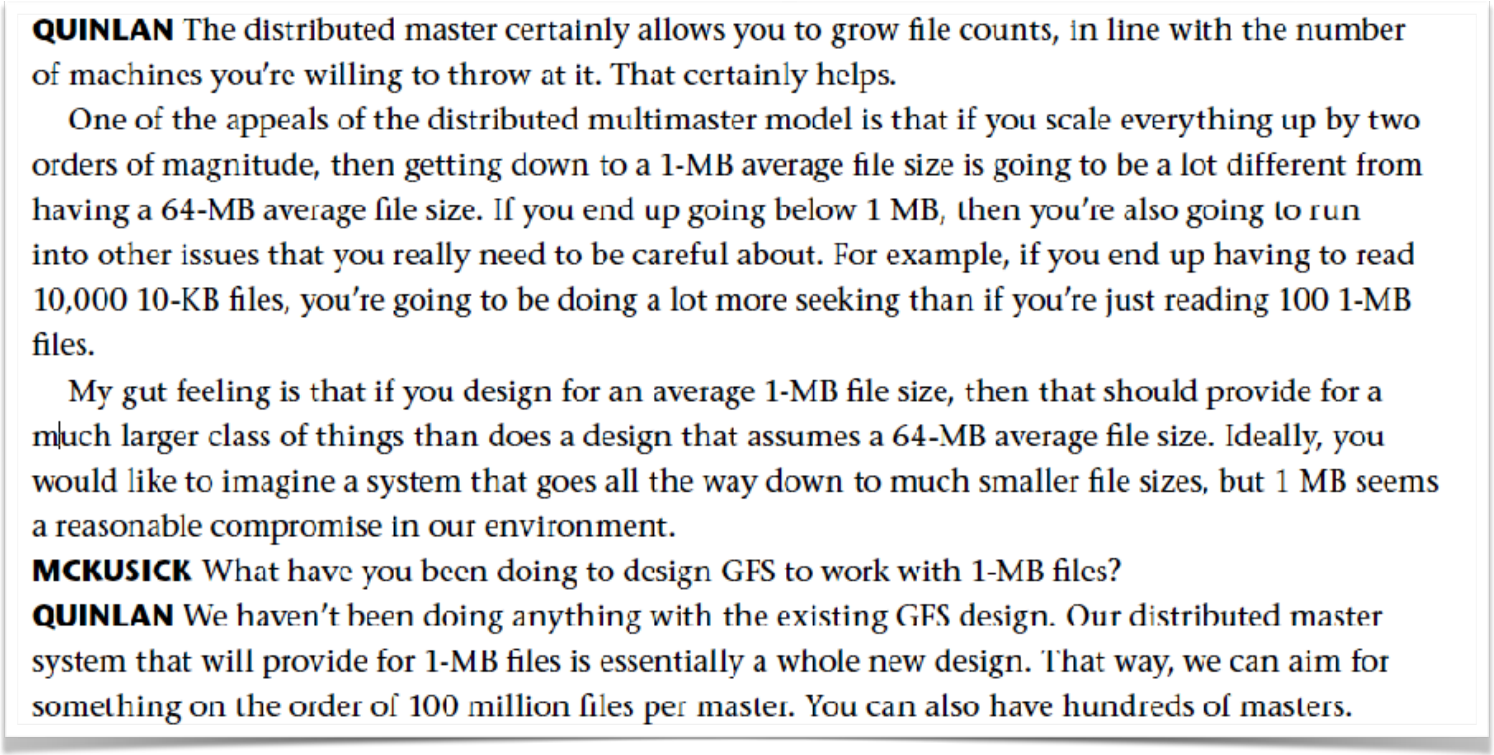


MCKUSICK And historically you've had one cell per data center, right?

QUINLAN That was initially the goal, but it didn't work out like that to a large extent—partly because of the limitations of the single-master design and partly because isolation proved to be difficult. As a consequence, people generally ended up with more than one cell per data center. We also ended up doing what we call a “multi-cell” approach, which basically made it possible to put multiple GFS masters on top of a pool of chunkservers. That way, the chunkservers could be configured to have, say, eight GFS masters assigned to them, and that would give you at least one pool of underlying storage—with multiple master heads on it, if you will. Then the application was responsible for partitioning data across those different cells.

The evolution of GFS

- Support for smaller chunk size — gmail



QUINLAN The distributed master certainly allows you to grow file counts, in line with the number of machines you're willing to throw at it. That certainly helps.

One of the appeals of the distributed multimaster model is that if you scale everything up by two orders of magnitude, then getting down to a 1-MB average file size is going to be a lot different from having a 64-MB average file size. If you end up going below 1 MB, then you're also going to run into other issues that you really need to be careful about. For example, if you end up having to read 10,000 10-KB files, you're going to be doing a lot more seeking than if you're just reading 100 1-MB files.

My gut feeling is that if you design for an average 1-MB file size, then that should provide for a much larger class of things than does a design that assumes a 64-MB average file size. Ideally, you would like to imagine a system that goes all the way down to much smaller file sizes, but 1 MB seems a reasonable compromise in our environment.

MCKUSICK What have you been doing to design GFS to work with 1-MB files?

QUINLAN We haven't been doing anything with the existing GFS design. Our distributed master system that will provide for 1-MB files is essentially a whole new design. That way, we can aim for something on the order of 100 million files per master. You can also have hundreds of masters.

Lots of other interesting topics

- snapshots
- namespace locking
- replica placement
- create, re-replication, re-balancing
- garbage collection
- stable replica detection
- data integrity
- diagnostic tools: logs are your friends

Do they achieve their goals?

- Storage based on inexpensive disks that fail frequently — replication, distributed storage
- Many large files in contrast to small files for personal data — large chunk size
- Primarily reading streams of data — large chunk size
- Sequential writes appending to the end of existing files — large chunk size
- Must support multiple concurrent operations — flat structure
- Bandwidth is more critical than latency — large chunk size

Why we care about GFS

- Conventional file systems do not fit the demand of data centers
- Workloads in data centers are different from conventional computers
  - Storage based on inexpensive disks that fail frequently

    — MapReduce is fault tolerant
  - Many large files in contrast to small files for personal data

    — MapReduce aims at processing large amount of data once
  - Primarily reading streams of data — MapReduce reads chunks of large files
  - Sequential writes appending to the end of existing files

    — Output file keep growing as workers keep writing
  - Must support multiple concurrent operations

    —MapReduce has thousands of workers simultaneously
  - Bandwidth is more critical than latency

    —MapReduce only wants to finish tasks within “reasonable” amount of time

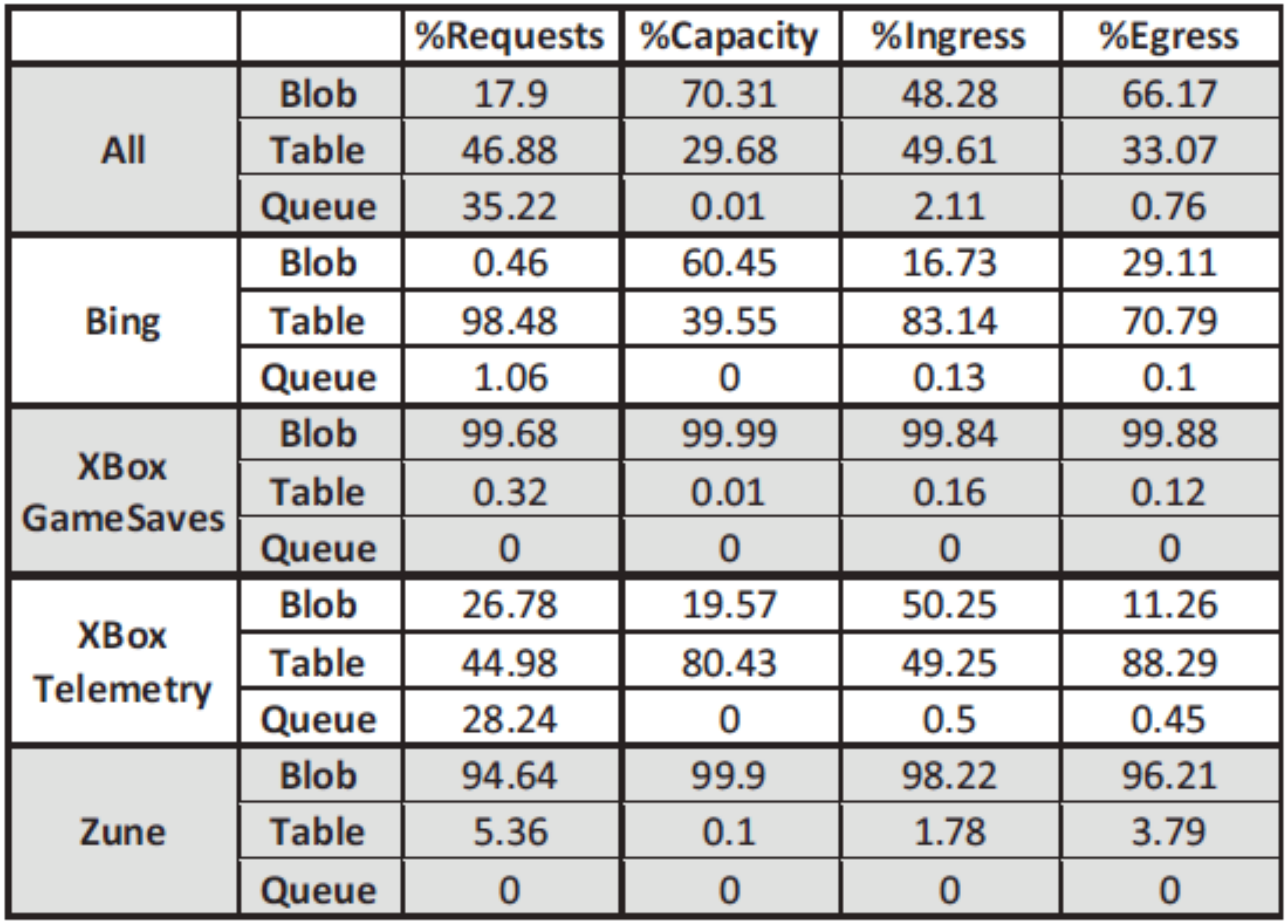
What's missing in GFS?

- GFS only supports consistency models
- Scalability — single master
- No geo-redundancy
- Only efficient in dealing with large data

Windows Azure Storage: A Highly Available Cloud Storage Service with Strong Consistency

Brad Calder, Ju Wang, Aaron Ogus, Niranjan Nilakantan, Arild Skjolsvold, Sam McKelvie, Yikang Xu, Shashwat Srivastav, Jiesheng Wu, Huseyin Simitci, Jaidev Haridas, Chakravarthy Uddaraju, Hemal Khatri, Andrew Edwards, Vaman Bedekar, Shane Mainali, Rafay Abbasi, Arpit Agarwal, Mian Fahim ul Haq, Muhammad Ikram ul Haq, Deepali Bhardwaj, Sowmya Dayanand, Anitha Adusumilli, Marvin McNett, Sriram Sankaran, Kavitha Manivannan, Leonidas Rigas
Microsoft

Data center workloads for WAS



		%Requests	%Capacity	%Ingress	%Egress
All	Blob	17.9	70.31	48.28	66.17
	Table	46.88	29.68	49.61	33.07
	Queue	35.22	0.01	2.11	0.76
Bing	Blob	0.46	60.45	16.73	29.11
	Table	98.48	39.55	83.14	70.79
	Queue	1.06	0	0.13	0.1
XBox GameSaves	Blob	99.68	99.99	99.84	99.88
	Table	0.32	0.01	0.16	0.12
	Queue	0	0	0	0
XBox Telemetry	Blob	26.78	19.57	50.25	11.26
	Table	44.98	80.43	49.25	88.29
	Queue	28.24	0	0.5	0.45
Zune	Blob	94.64	99.9	98.22	96.21
	Table	5.36	0.1	1.78	3.79
	Queue	0	0	0	0

Why Windows Azure Storage

- A cloud service platform for social network search, video streaming, XBOX gaming, records management, and etc. in M$.
  - Must tolerate many different data abstractions: blobs, tables and queues
  - Data types:
    - **Large** Blob(Binary Large OBjects) storage: pictures, excel files, HTML files, virtual hard disks (VHDs), big data such as logs, database backups -- pretty much anything.
    - **Large** Table: database tables
    - **Small** Queue: store and retrieve messages. Queue messages can be up to 64 KB in size, and a queue can contain millions of messages. Queues are generally used to store lists of messages to be processed asynchronously.

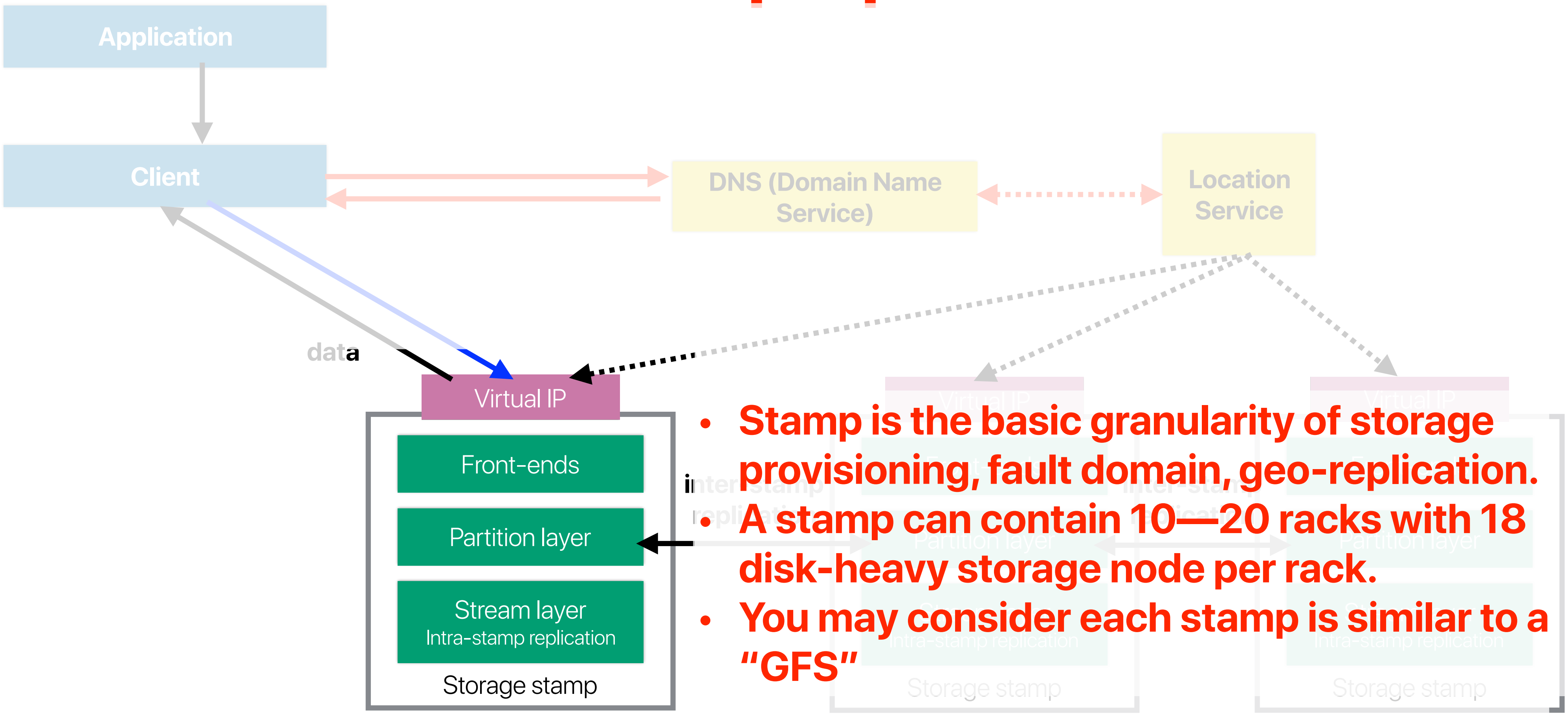
Why Windows Azure Storage (cont.)

- Learning from feedbacks in existing cloud storage
  - Strong consistency
  - Global and scalable namespace/storage
  - Disaster recovery
  - Multi-tenancy and cost of storage

All problems in computer science can be solved by another level of indirection

–David Wheeler

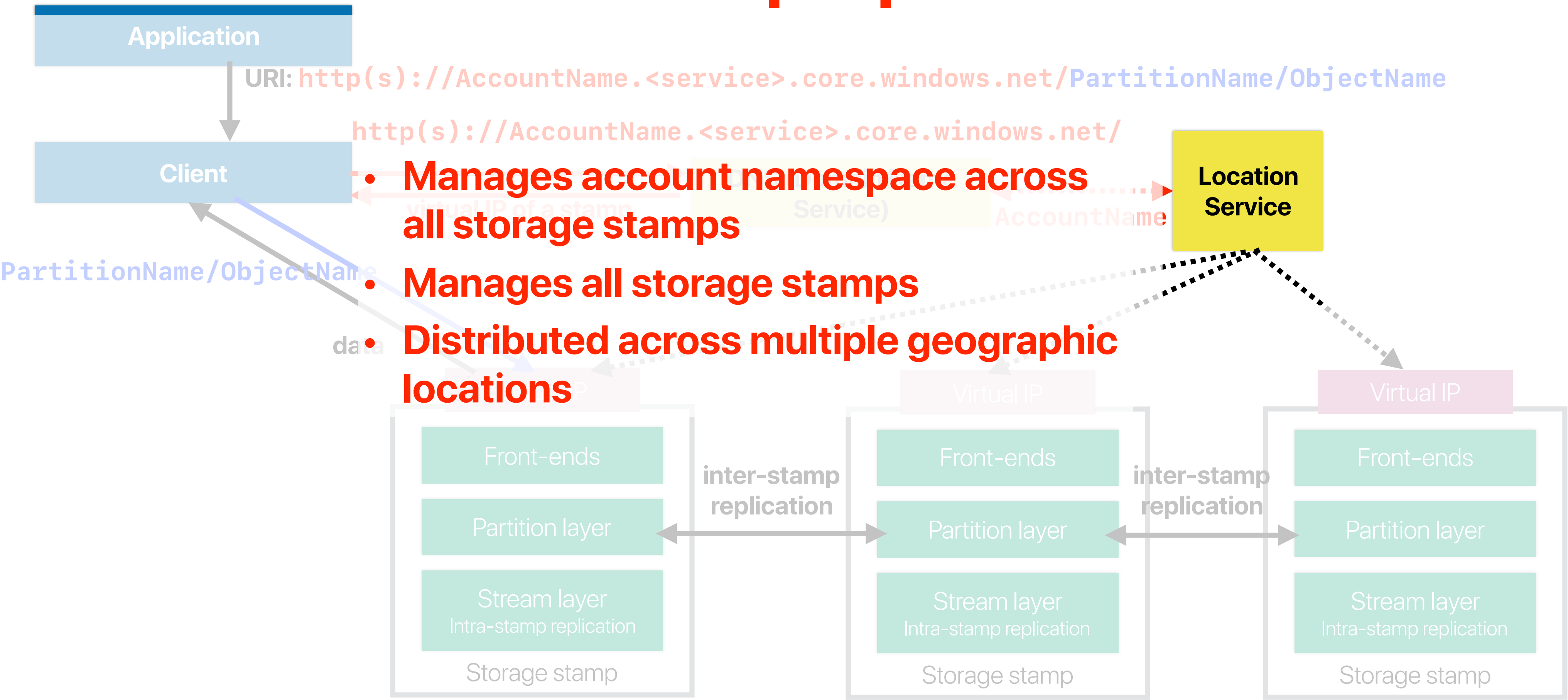
What WAS proposes?



- Stamp is the basic granularity of storage provisioning, fault domain, geo-replication.
- A stamp can contain 10—20 racks with 18 disk-heavy storage node per rack.
- You may consider each stamp is similar to a "GFS"

What WAS proposes?

- Manages account namespace across all storage stamps
- Manages all storage stamps
- Distributed across multiple geographic locations



Application

URI: http(s)://AccountName.<service>.core.windows.net/PartitionName/ObjectName

Client

http(s)://AccountName.<service>.core.windows.net/

virtual IP of a stamp

Service)

AccountName

Location Service

PartitionName/ObjectName

data

Virtual IP

Front-ends

Partition layer

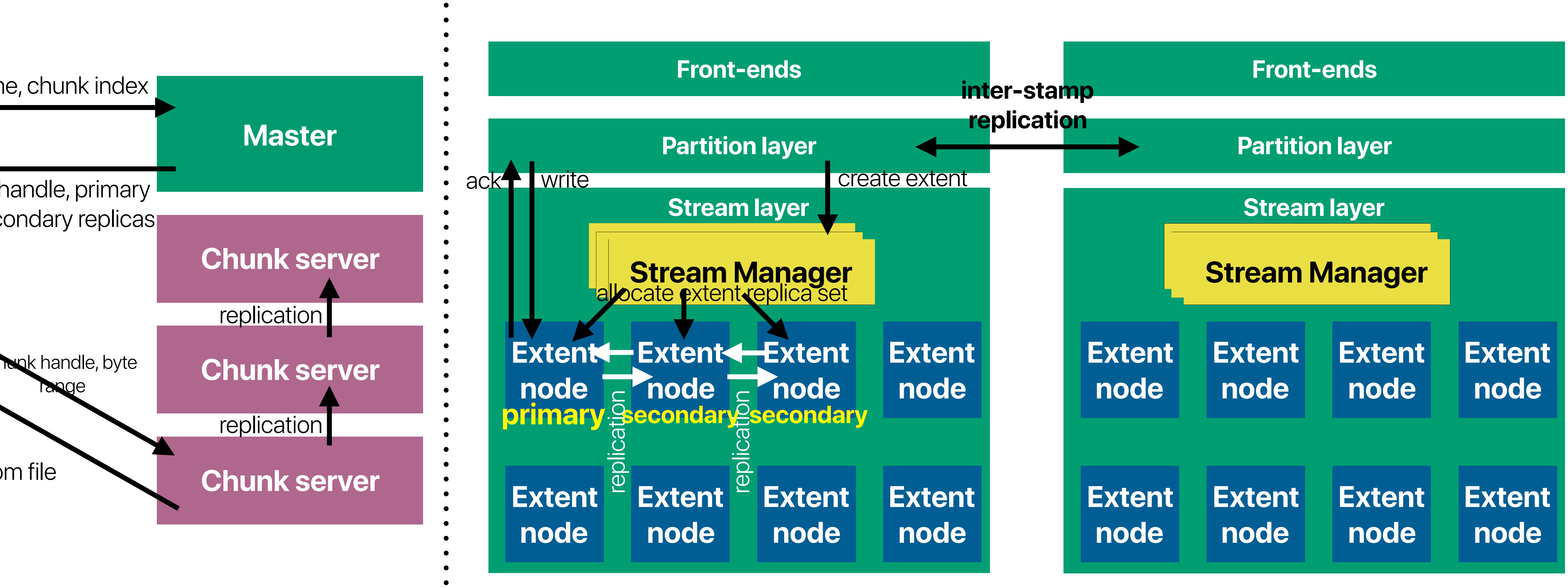
Stream layer

Intra-stamp replication

Storage stamp

inter-stamp replication

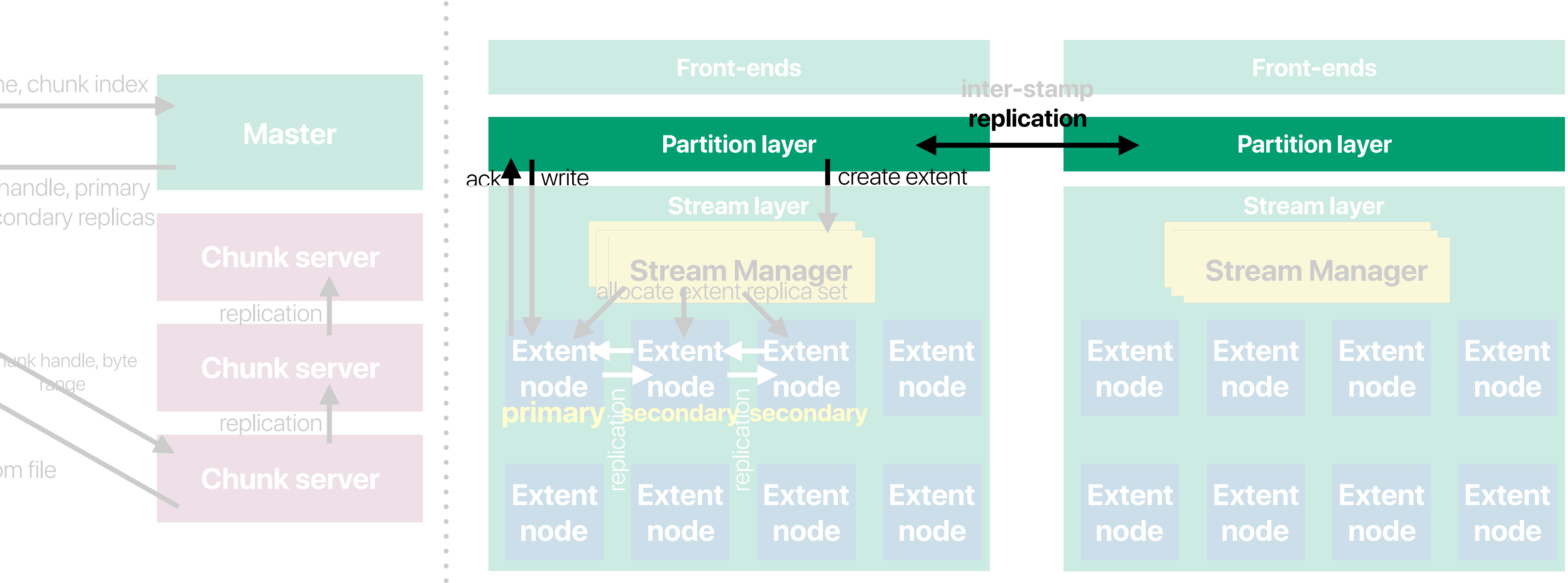
GFS v.s. stamp in WAS



Write failure

- Consider the case where 1 of 3 nodes handling a write fails and the current extent is sealed at latest commit boundary (end of extent) — that data will be on failed node
- new extent created
- SM chooses **three** new replicas to store extents
- client retries via new primary among the three new replicas
- failed node, upon restart, will coord w/ SM to synchronize its extent to the commit length decided upon

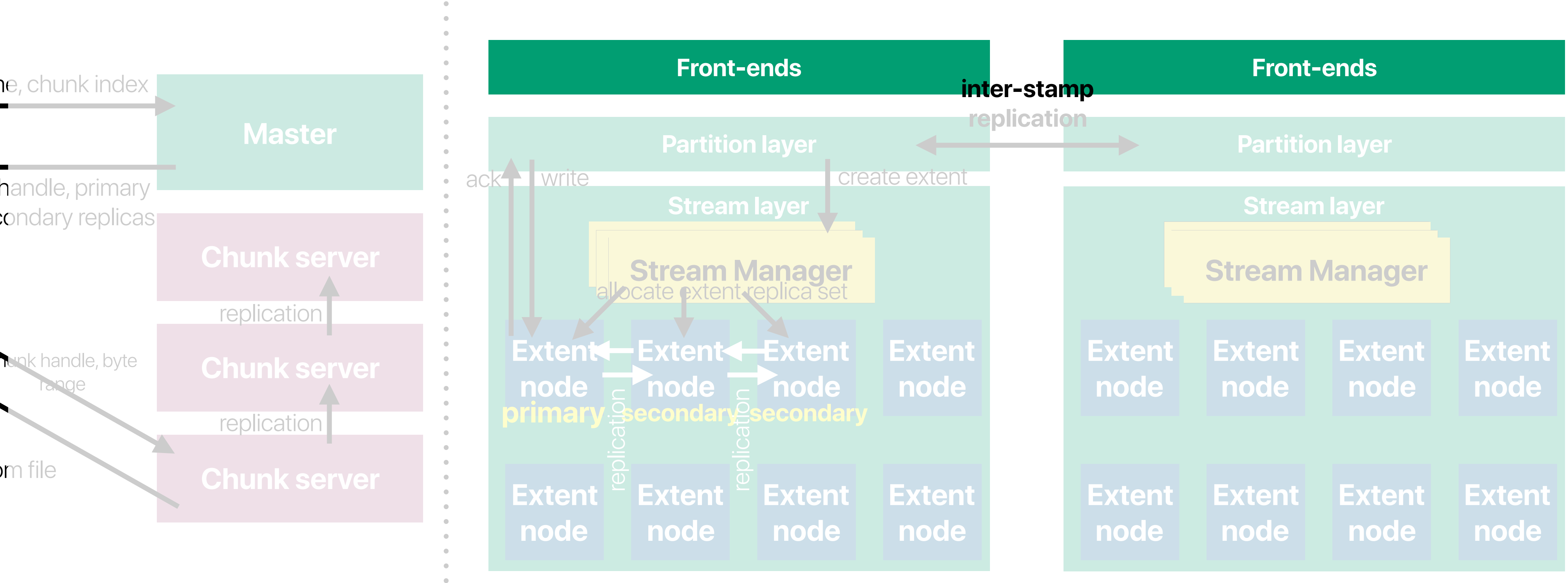
GFS v.s. stamp in WAS



Partition layer

- Managing high-level data abstractions
- Providing scalable object namespaces
- Providing transaction ordering and strong consistency for objects
- Storing object data on top of the stream layer
- Cache object data to reduce disk I/O

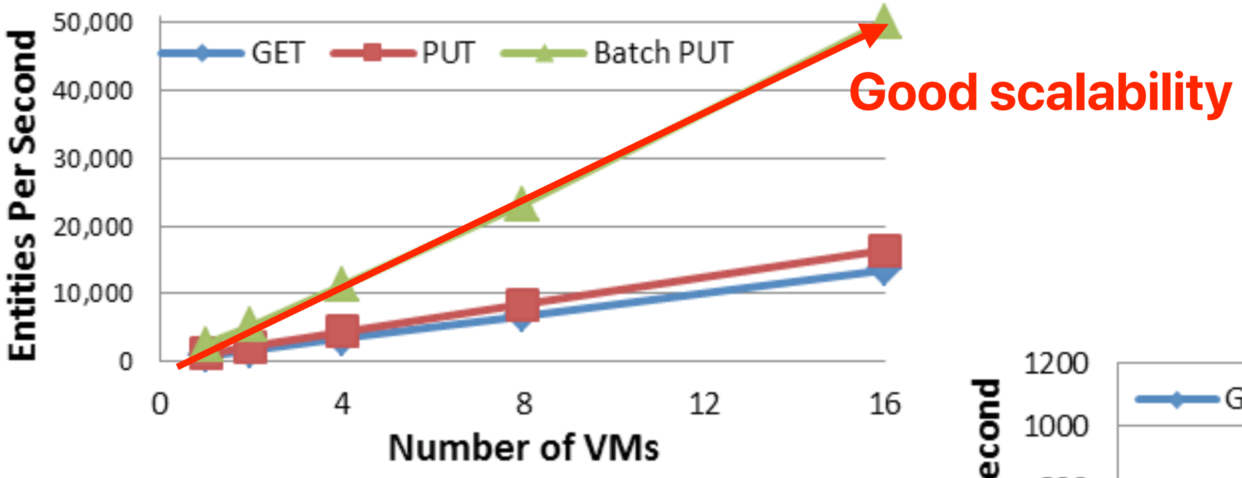
GFS v.s. stamp in WAS

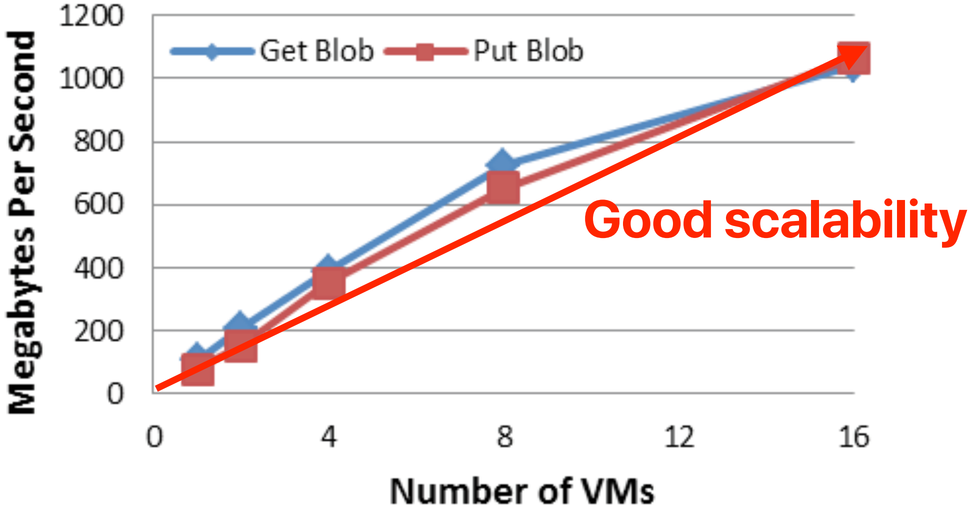


Front-end layer

- A set of **stateless** servers taking incoming requests
  - Think about the benefits of stateless in NFS
- Keep partition maps to forward the request to the right server
  - A stamp can contain 10—20 racks with 18 disk-heavy storage node per rack
- Stream large objects directly from the stream layer and cache frequently accessed data for efficiency

Are they doing well?





GFS v.s. WAS

	GFS (OSDI 2003)	WAS (SOSP 2011)
File organizations	file chunk block	stream extent record
System architecture	master chunkserver	stream manager extent nodes
Data updates		append only updates
Consistency models	relaxed consistency	strong consistency
Data formats	files	multiple types of objects
Replications	intra-cluster replication	geo-replication
Usage of nodes	chunk server can perform both	separate computation and storage

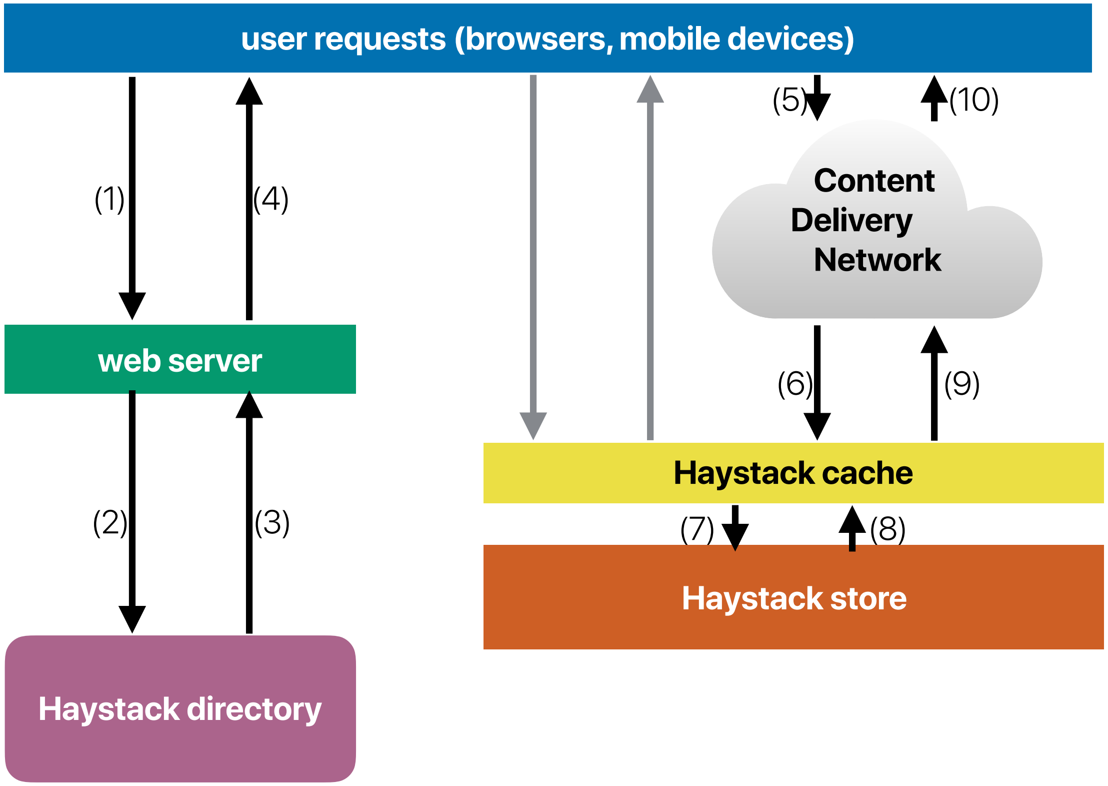
f4: Facebook's Warm BLOB Storage System

Subramanian Muralidhar, Wyatt Lloyd, Sabyasachi Roy, Cory Hill,
Ernest Lin, Weiwen Liu, Satadru Pan, Shiva Shankar,
Viswanath Sivakumar, Linpeng Tang, and Sanjeev Kumar.

The original NFS-based FB storage

- Within a data center with high-speed network, the round-trip latency of network accesses is not really a big deal
- However, the amount of metadata, especially directory metadata, is huge — cannot be cached
- As a result, each file access still requires ~ 10 inode/data requests from disks/network nodes — kill performance

Haystack

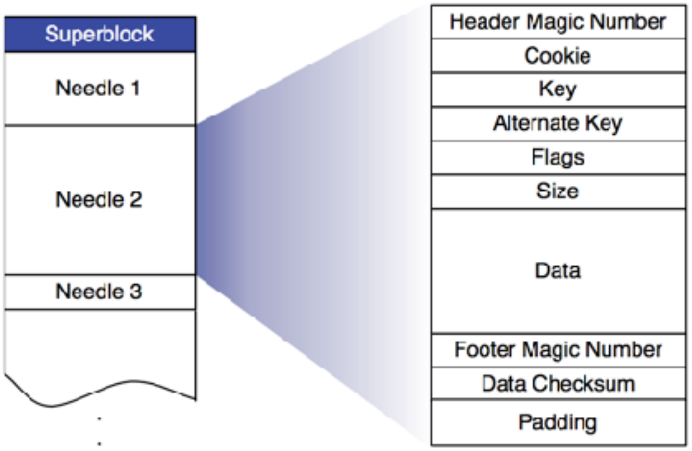


http://<CDN>/<Cache>/<Machine ID>/<Logical volume,Photo>

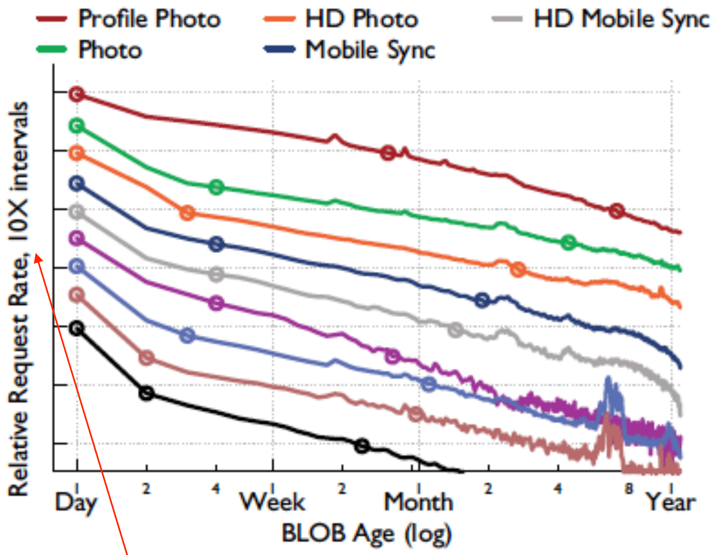
Finding a needle in Haystack: Facebook's photo storage, OSDI 2010

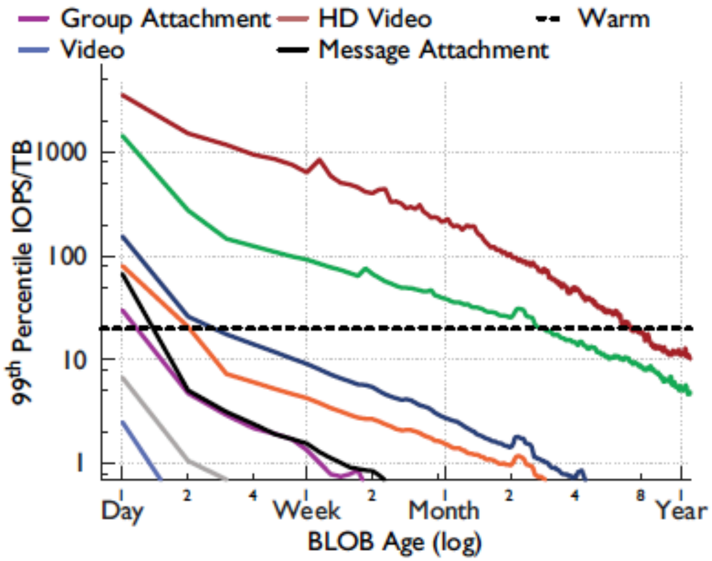
Haystack

- Each storage unit provides 10TB of usable space, using RAID-6 — 20% redundancy for parity bits
  - Each storage split into 100 physical volumes (100GB)
  - Physical volumes on different machines grouped into logical volumes
  - A photo saved to a logical volume is written to all corresponding physical volumes — 3 replicas
- Each volume is actually just a large file
  - Needle represents a photo
  - Each needle is identified through the offset
  - Sealed (the same as WAS) one the file reaches 100GB



“Temperature” of data



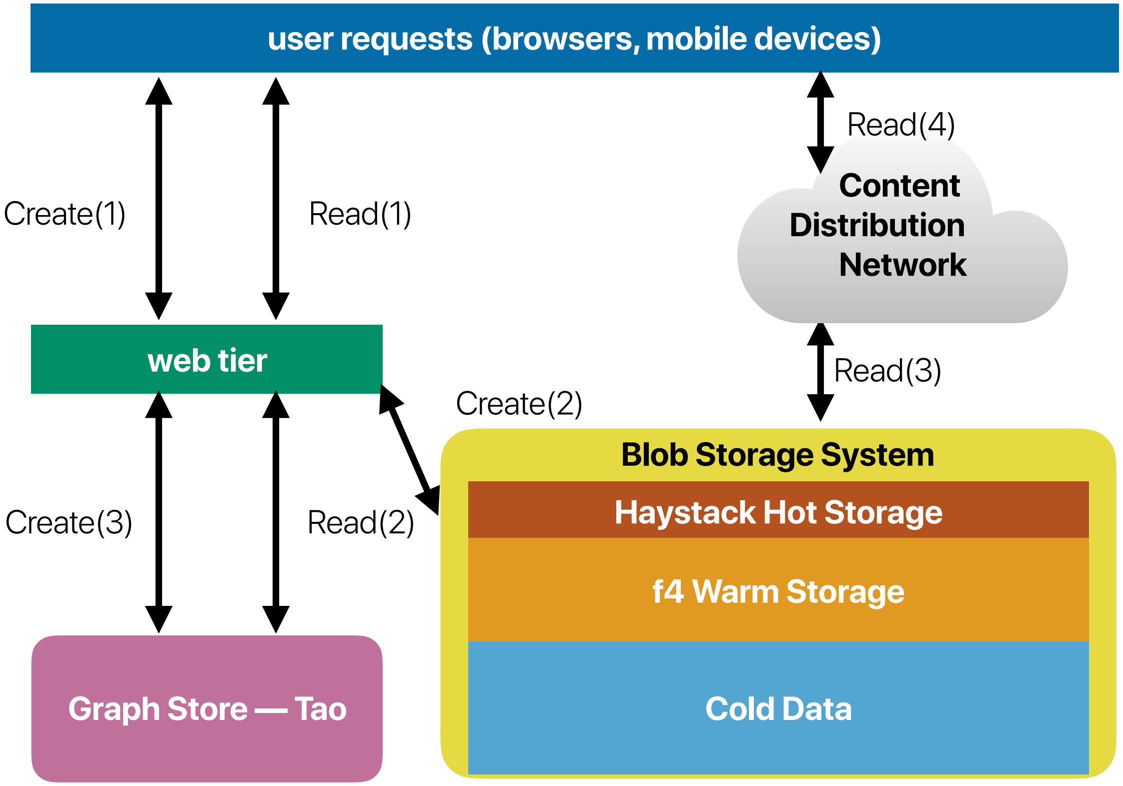


log scale — not encouraged to graph like this if you’re writing a technical document or scientific paper

“Temperature” of data

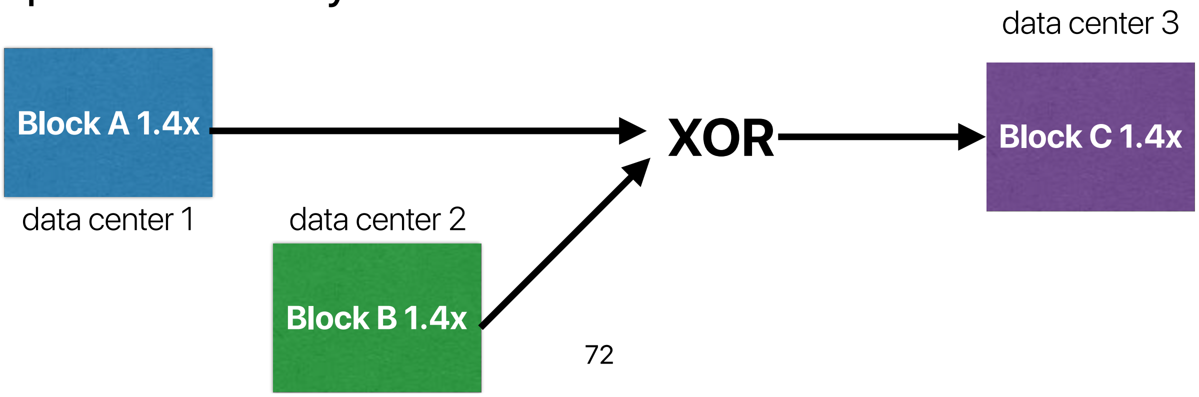
	Hot	Warm	Cold
Access Frequency	Most frequent	Less frequent	Rare
Pattern	Created often, delete often	Not so frequently read Not so frequently deleted Maybe read-only	Long-term storage, usually takes hours to retrieve
Size		65PB in 2014 and growing rapidly	

Facebook storage architecture



Storage efficiency

- Reed-Solomon erasure coding
  - Strips: 10GB data + 4GB parity — 1.4x space efficiency
  - One volume contains 10 strips
- XOR Geo-replication
  - Use XOR to reduce overhead further (e.g., Azure makes full copies)
  - Block A in DC1 + block B in DC2 -> parity block P in DC3
  - Any two blocks can be used to generate the third
  - 1.5x space efficiency
- 1.4*1.5 = 2.1x space efficiency in total



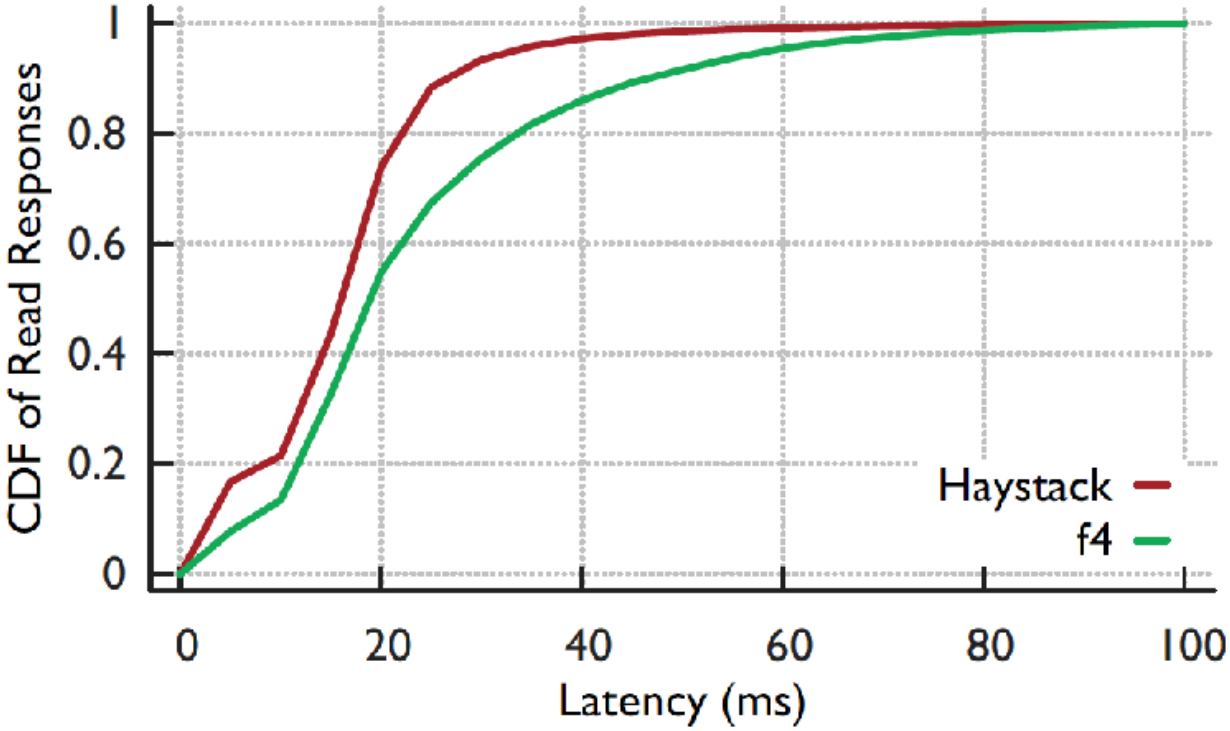
Fault tolerance

- 1%-2% HDD fail in a year
  - replicate data across multiple disks
  - Use erasure coding for storage efficiency
    - n blocks -> n + k blocks, can tolerate k simultaneous failures
    - higher cost for recovering data when there is a failure
- Host failures (periodically)
  - replicate coded blocks on different hosts
- Rack failures (multiple times/year)
  - replicate coded blocks on different racks
- Datacenter failures (rare, but catastrophic)
  - replicate blocks across data centers
  - use XOR to reduce overhead further (e.g., Azure makes full copies)
    - block A in DC1 + block B in DC2 -> parity block P in DC3
    - any two blocks can be used to generate the third
- Index files
  - use normal triple replication (tiny, little benefit in coding them)

What happens if fault occurs?

- Drive fails
  - Reconstruct blocks on another drive
  - Heavy disk, Network, CPU operation
  - one in background
- During failure, may need to reconstruct data online
  - rebuilder node reads BLOB from data + parity, reconstructs
  - only reads + reconstructs the BLOB (40KB), not the entire block (1GB)

Performance of f4



Cells

- Each cell contains 14 racks of 15 hosts, each host contains 30 4TB H.D.Ds.
- A unit of acquisition, deployment
- Storage for a set of volumes
- Similar to the idea of stamps

Make common case fast,
make rare case correct