Cloud Scale Storage Systems

Sean Ogden
October 30, 2013

Evolution

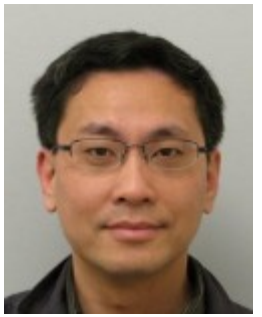
- P2P routing/DHTs (Chord, CAN, Pastry, etc.)
- P2P Storage (Pond, Antiquity)
  - Storing Greg's baby pictures on machines of untrusted strangers that are connected with wifi
- Cloud storage
  - Store Greg's baby pictures on trusted data center network at Google

Cloud storage – Why?

- Centralized control, one administrative domain
- Can buy seemingly infinite resources
- Network links are high bandwidth
- Availability is important
- Many connected commodity machines with disks is cheap to build
  - Reliability from software

The Google File System





Sanjay Ghemawat, Howard Gobioff, Shun-tak Leung

GFS Assumptions and Goals

- Given
  - Large files, large sequential writes
  - Many concurrent appending applications
  - Infrequent updates
  - Trusted network
- Provide
  - Fast, well defined append operations
  - High throughput I/O
  - Fault tolerance

GFS Components

- Centralized master
- Chunk Server
- Clients

GFS Architecture

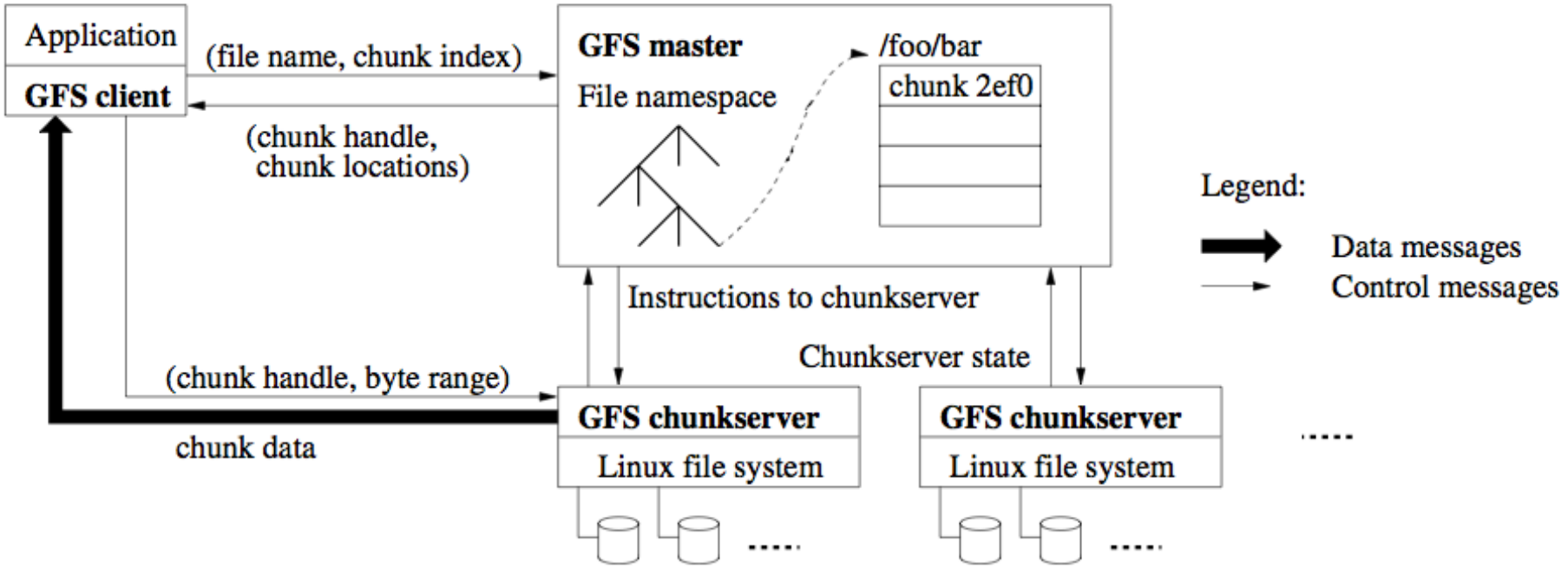


Figure 1: GFS Architecture

GFS Chunk Server

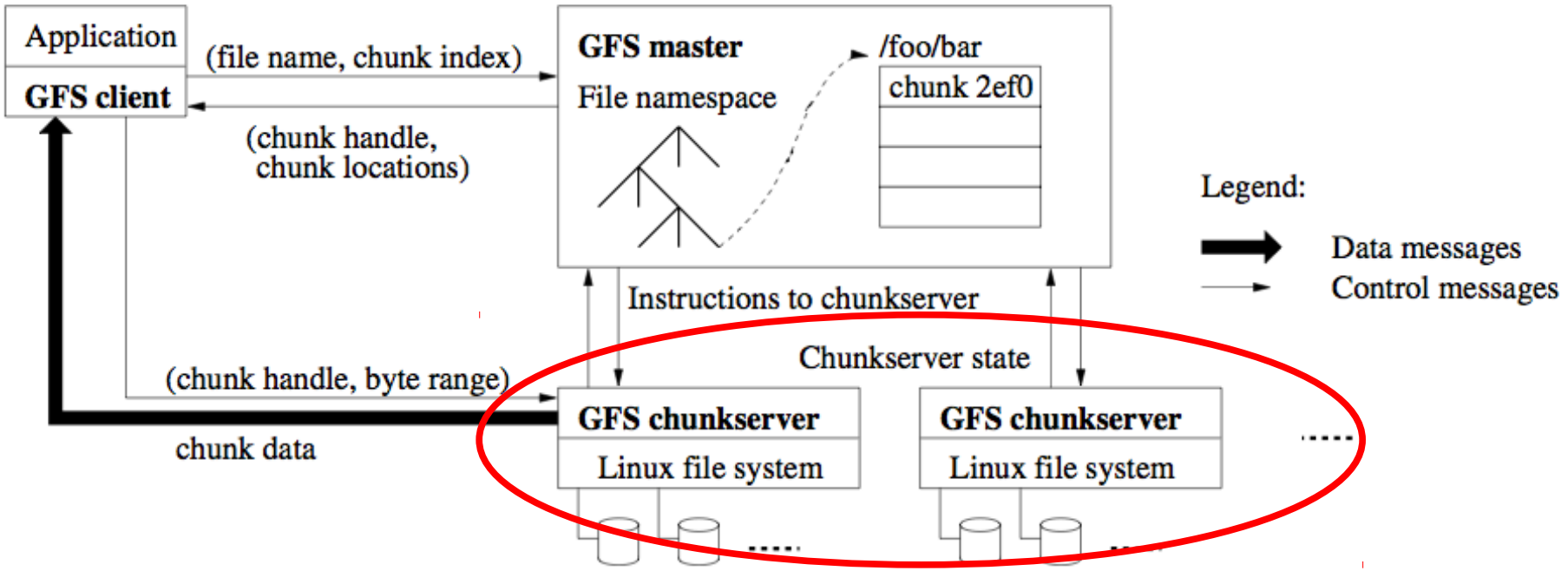


Figure 1: GFS Architecture

GFS Chunk server

- Holds chunks of data, 64MB by default
- Holds checksums of the chunks
- Responds to queries from master
- Receives data directly from clients
- Can be a delegate authority for a block

GFS Master

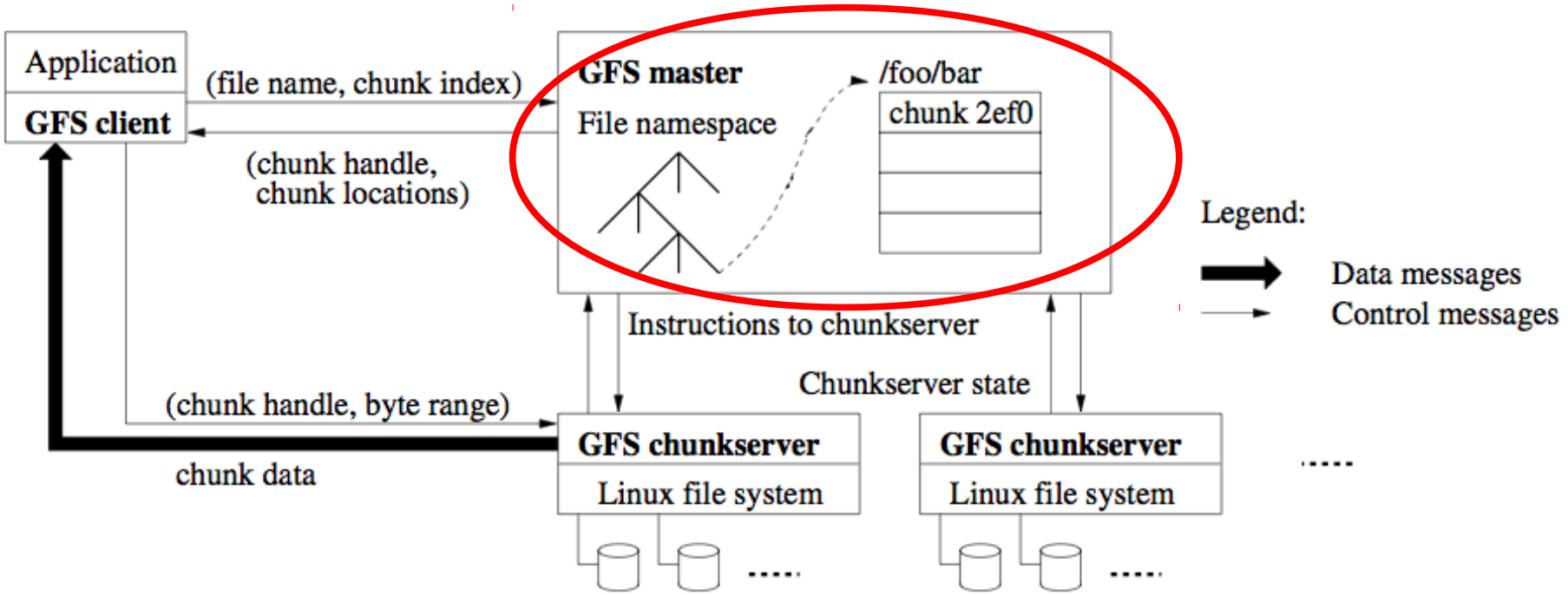


Figure 1: GFS Architecture

GFS Master

- Holds file system metadata
  - What chunk server holds which chunk
  - Metadata table is not persistent
- Directs clients
- Centralized
  - Ease of implementation
  - Can do load balancing
  - Not in the data path
- Replicated for fault tolerance

GFS Client

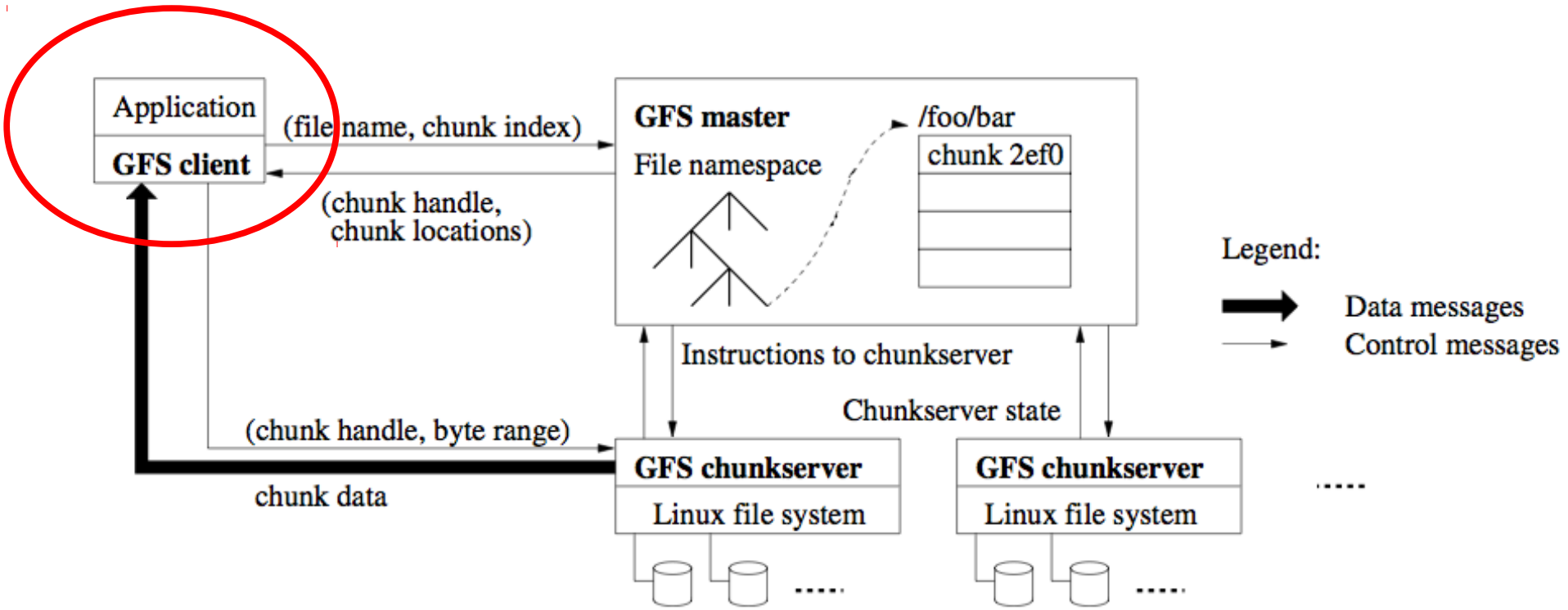


Figure 1: GFS Architecture

GFS Client

- Queries master for metadata
- Reads/writes data directly to chunk servers

Write control and Data Flow

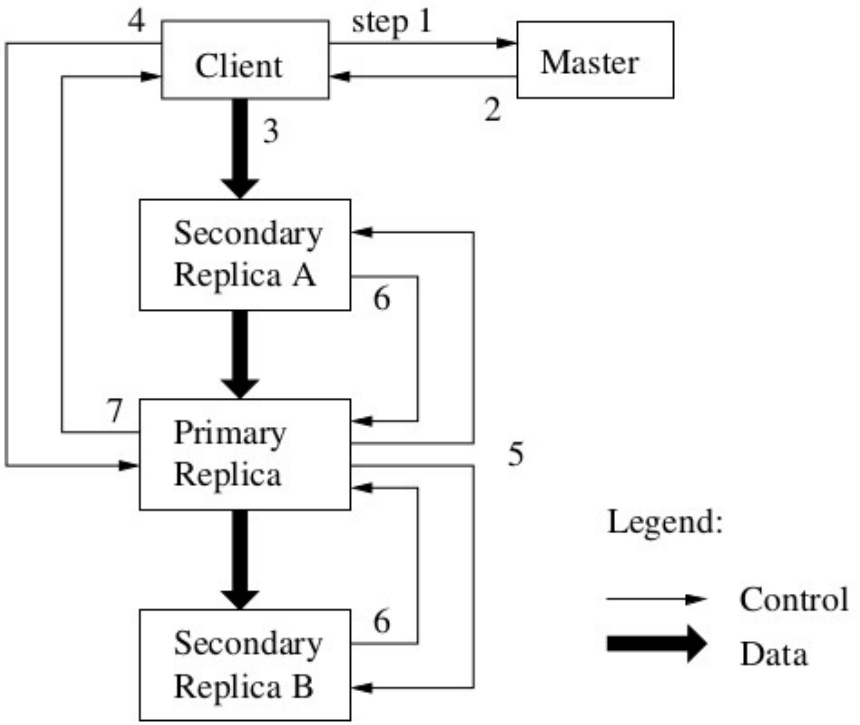


Figure 2: Write Control and Data Flow

Read control and data flow

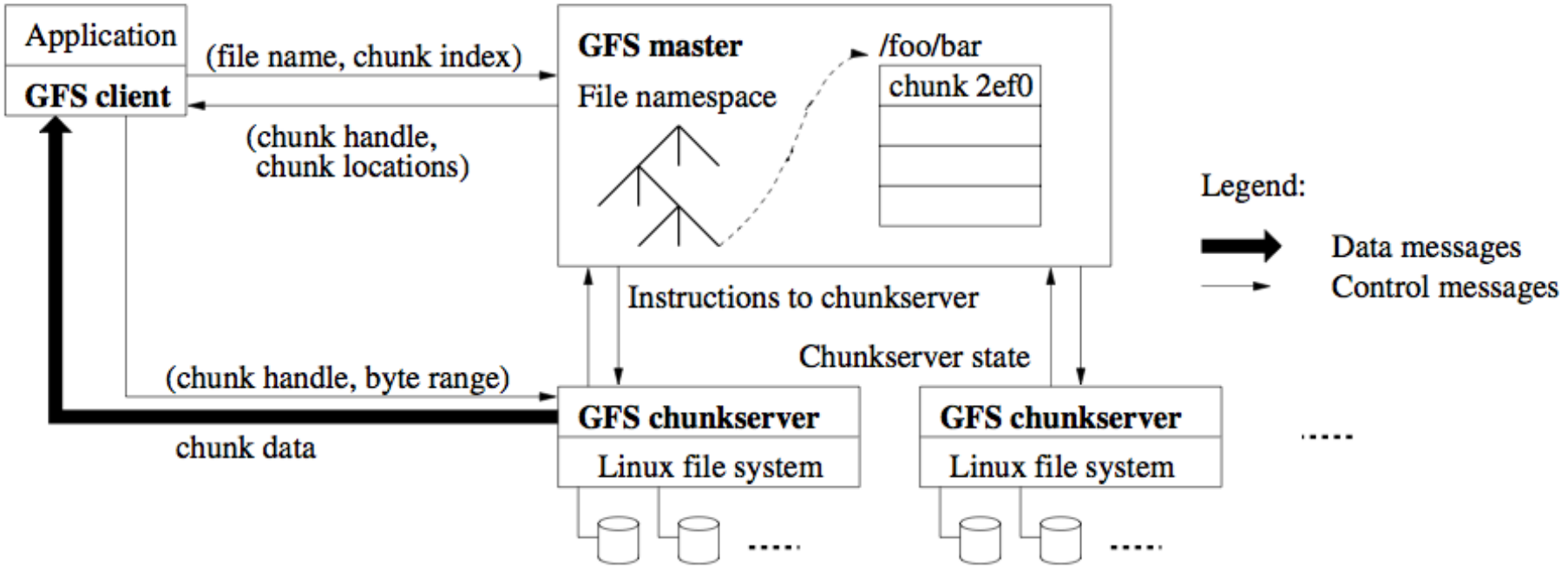


Figure 1: GFS Architecture

Supported operations

- Open
- Close
- Create
- Read
- Write
- Delete
- Atomic record append
- Snapshot

Consistency

- Relaxed consistency model
- File namespace mutations are atomic
- Files may be consistent and/or defined
- *Consistent*
  - All clients will see the same data
- *Defined*
  - Consistent *and* entire mutation is visible by clients

Consistency

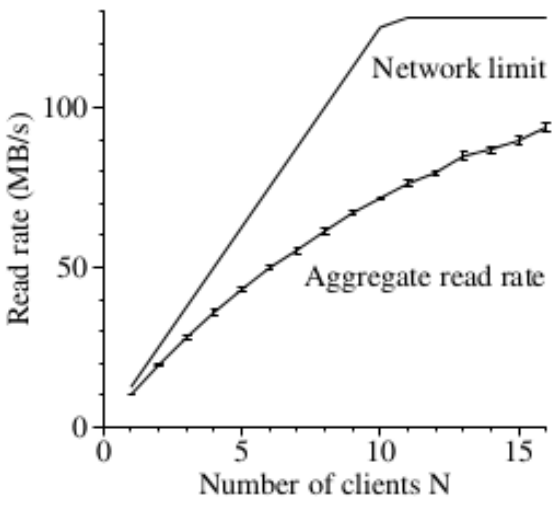
	Write	Record Append
Serial success	*defined*	*defined* interspersed with *inconsistent*
Concurrent successes	*consistent but not defined*	
Failure	*inconsistent*	

“Atomic” record appends

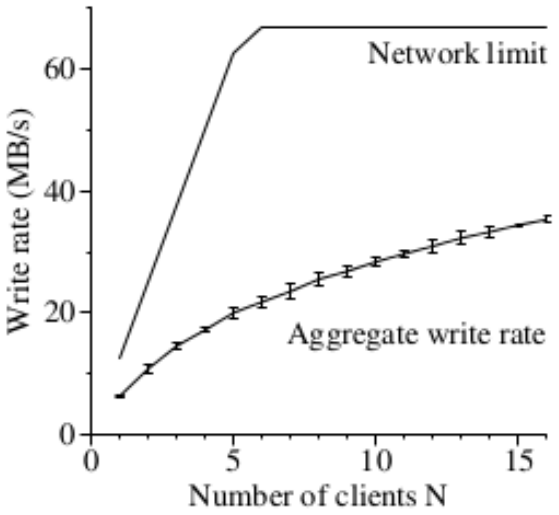
- Most frequently used operation
- “At least once” guarantee
- Failed append operation can cause blocks to have result of partially complete mutation
- Suppose we have a block that contains “DEAD”, and we append(f, “BEEF”)

Replica 1	DEAD	BEEF	BEEF
Replica 2	DEAD	BE	BEEF
Replica 3	DEAD		BEEF

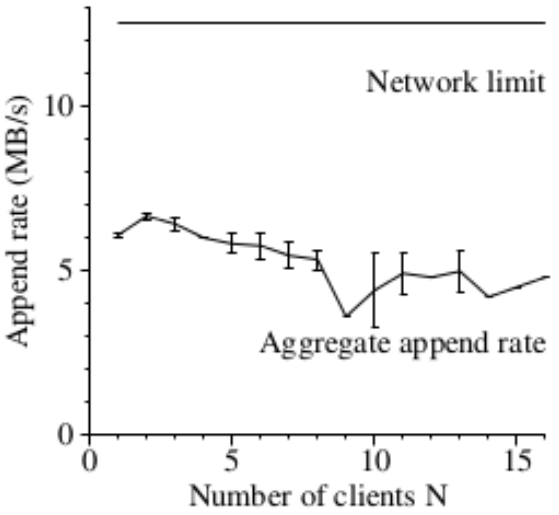
Performance



(a) Reads



(b) Writes



(c) Record appends

Performance notes

- It goes up and to the right
- Write throughput limited by network due to replication
- Master saw 200 ops/second

GFS Takeaways

- There can be benefits to a centralized master
  - If it is not in the write path
- Treat failure as the norm
- Ditching old standards can lead to drastically different designs that better fit a specific goal

Discussion

- Does GFS work for anyone outside of Google?
- Are industry papers useful to the rest of us?
- What are the pros/cons of single master in this system?
- Will there ever be a case where single master could be a problem?
- Could we take components of this and improve on them in some way for different work loads?

Windows Azure Storage

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 Riga

Azure Storage Goals and Assumptions

- Given
  - Multi tenant storage service
  - Publicly accessible – untrusted clients
  - Myriad of different usage patterns, not just large files
- Provide
  - Strong consistency
  - Atomic transactions (within partitions)
  - Synchronous local replication + asynchronous georeplication
  - Some useful high level abstractions for storage

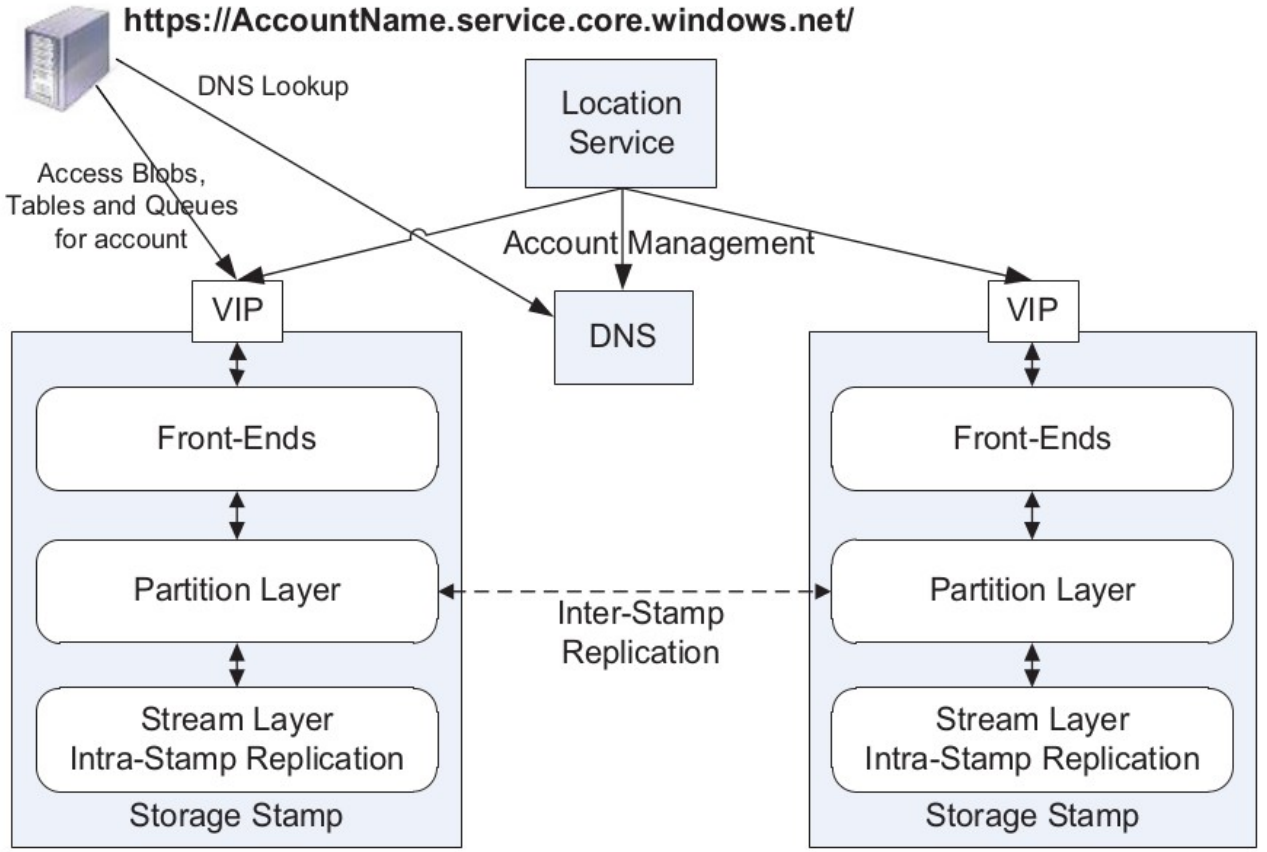
Azure vs. GFS

	GFS	Azure
Minimum block size	64 MB	~4MB
Unit of replication	Block	Extent
Mutable blocks?	Yes	No
Consistency	Not consistent	Strong
Replication	3 copies of full blocks	Erasure coding
Usage	Private within google	Public

Azure Architecture

- Stream Layer
- Partition Layer
- Front End Layer

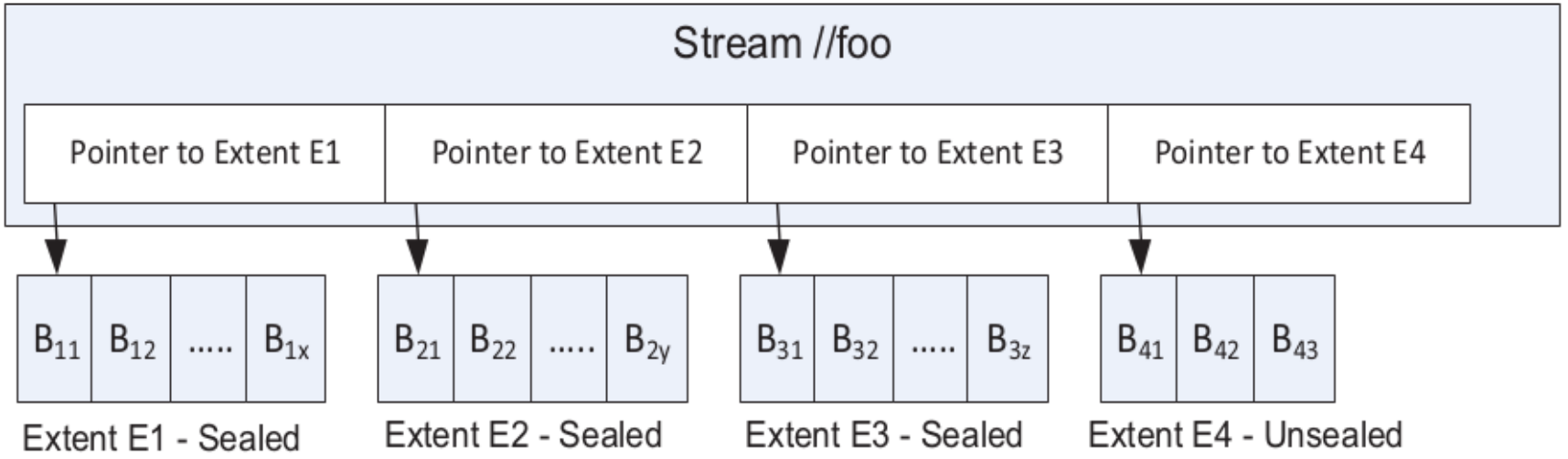
Azure Storage Architecture



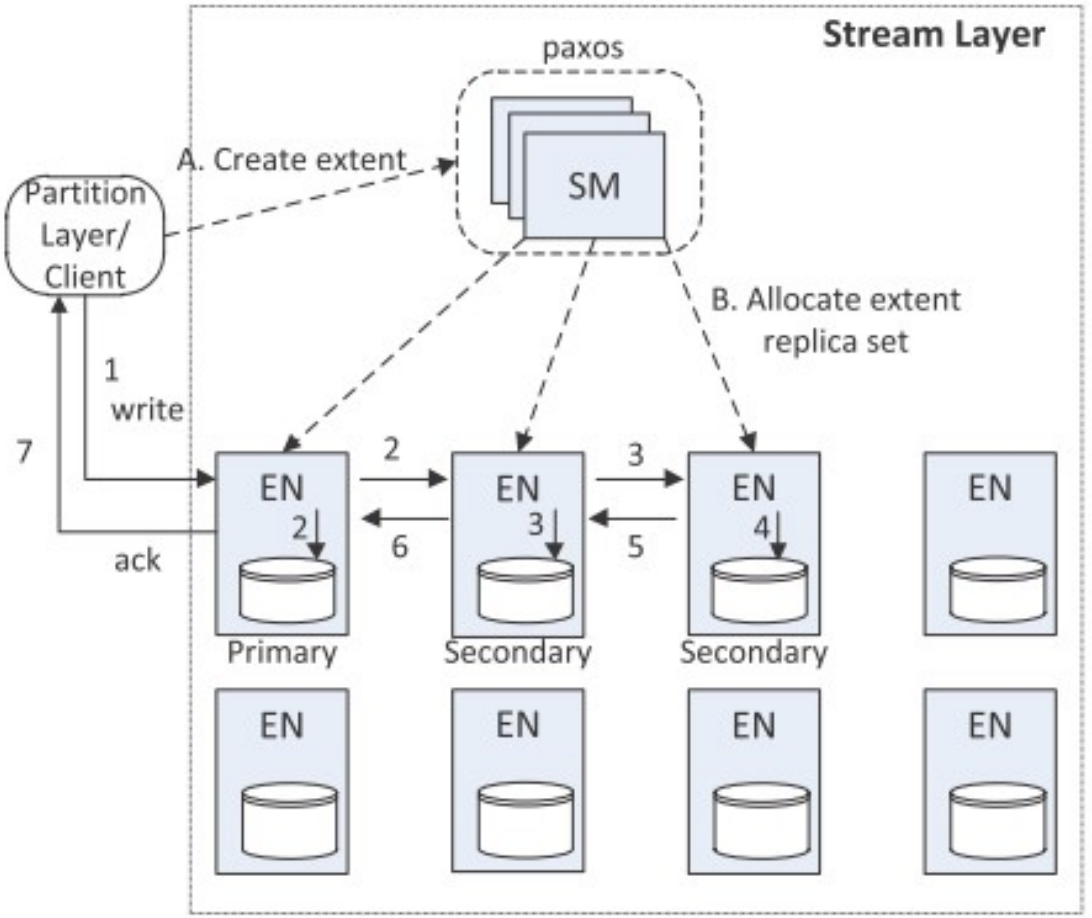
Azure Storage Stream Layer

- Provides file system abstraction
- Streams ≈ Files
  - Made up of pointers to *extents*
- *Extents* are made up of lists of blocks
- *Blocks* are the smallest unit of IO
  - Much smaller than in GFS (4MB vs. 64MB)
- Does synchronous intra-stamp replication

Anatomy of a Stream



Stream Layer Architecture



Stream Layer Optimizations

- Spindle anti-starvation
  - Custom disk scheduling predicts latencey
- Durability and Journaling
  - All writes must be durable on 3 replicas
  - Use an SSD and journal appends on every EN
  - Appends do not conflict with reads

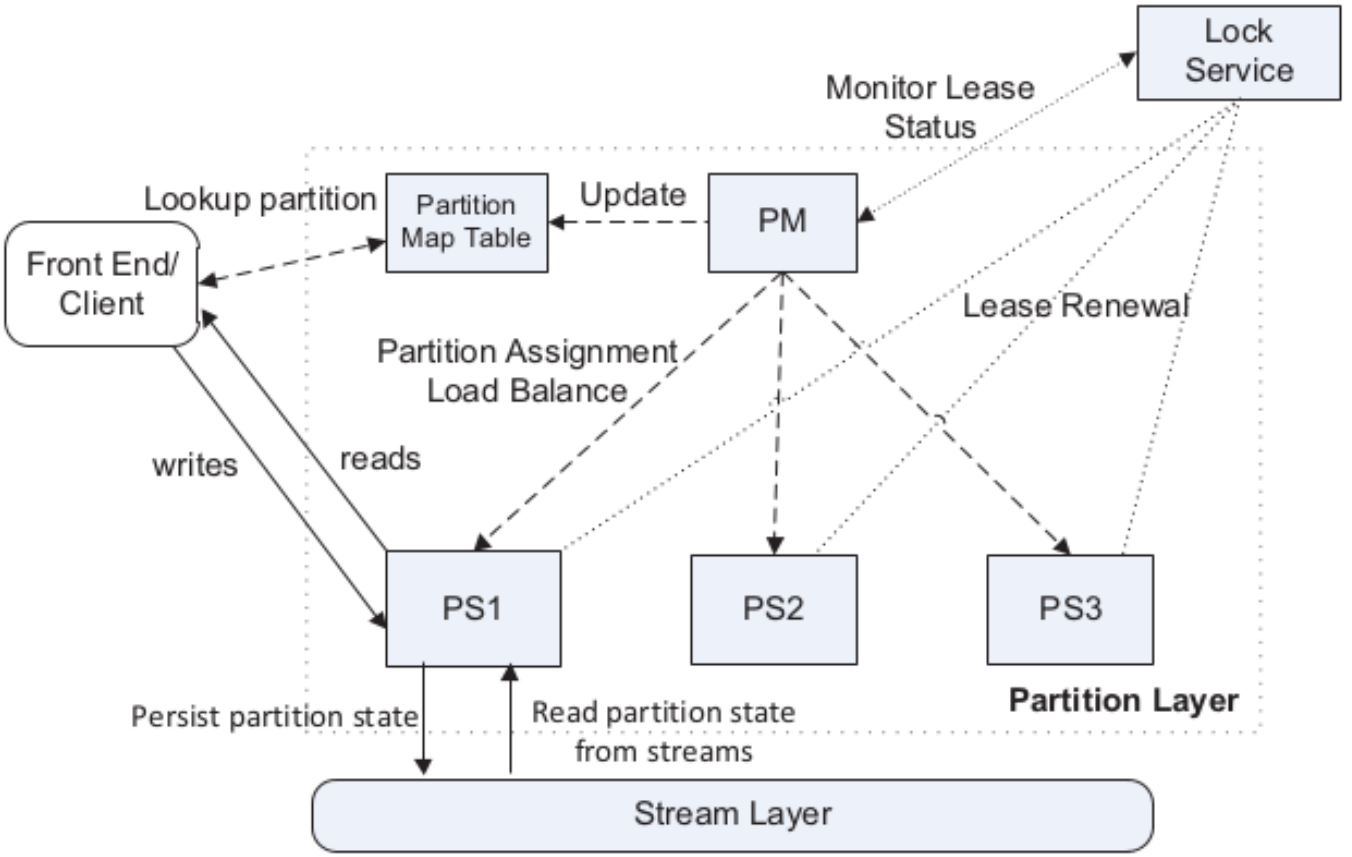
Partition Layer Responsibilities

- Manages higher level abstractions
  - Blob
  - Table
  - Queue
- Asynchronous Inter-Stamp replication

Partition Layer Architecture

- Partition server serves requests for RangePartitions
  - Only one partition server can serve a given RangePartition at any point in time
- Partition Manager keeps track of partitioning Object Tables into RangePartitions
- Paxos Lock Service used for leader election for Partition Manager

Partition Layer Architecture



Azure Storage Takeaways

- Benefits from good layered design
  - Queues, blobs and tables all share underlying stream layer
- Append only
  - Simplifies design of distributed storage
  - Comes at cost of GC
- Multitenancy challenges

Azure Storage discussion

- Did they really “beat” CAP theorem?
- What do you think about their consistency guarantee?
  - Would it be useful to have inter-namespace consistency guarantees?

Comparison