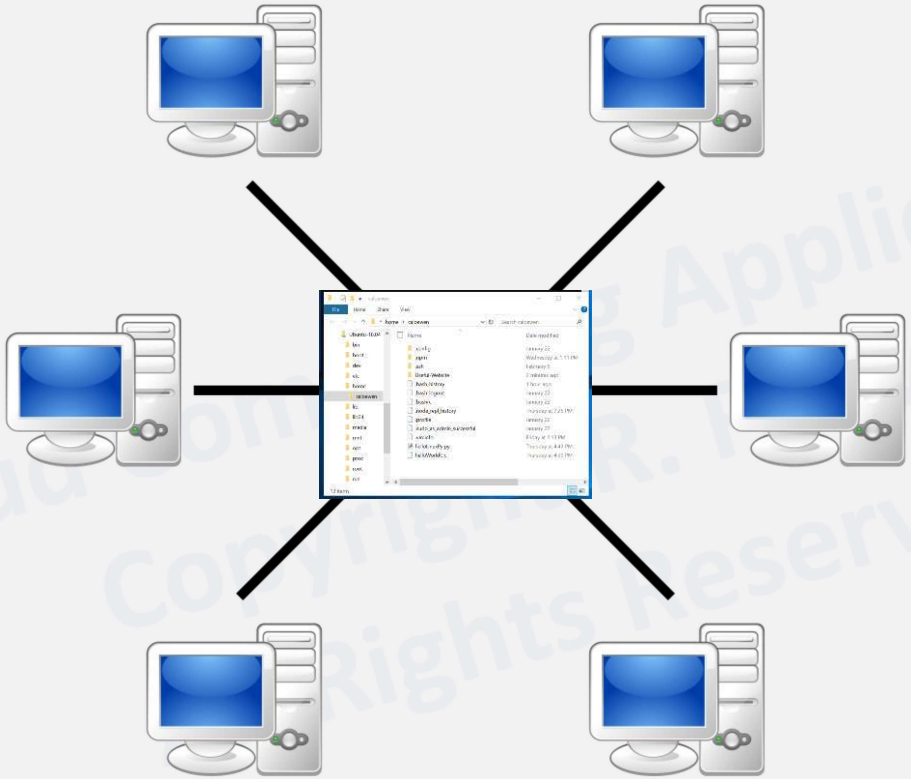
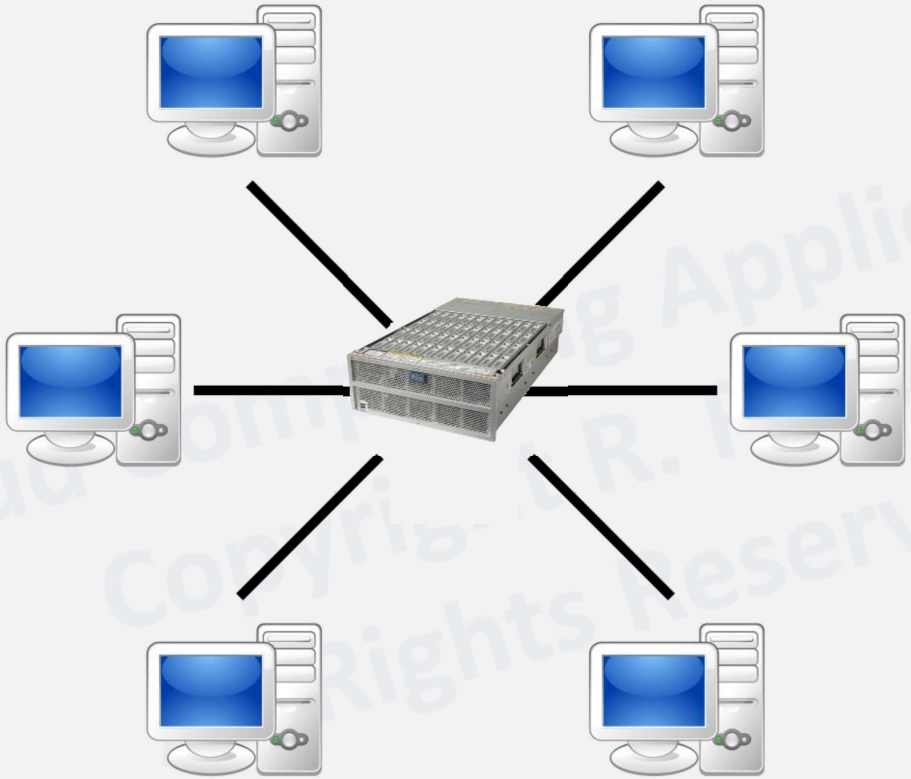
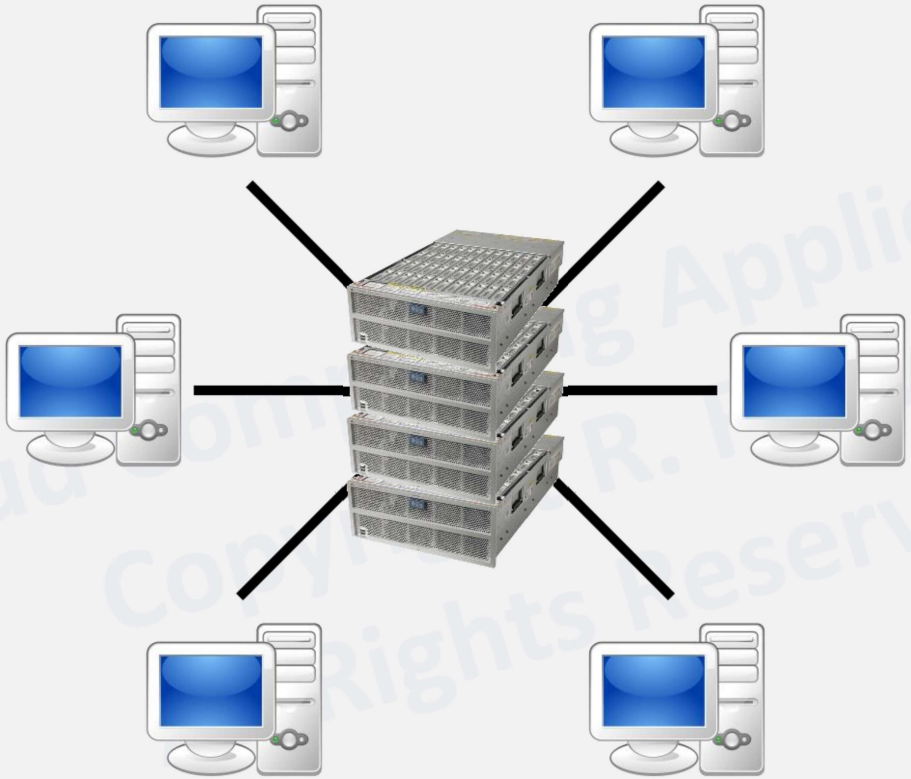
Logical View Networked File System



Network Attached Storage (NAS)



Network Attached Storage (NAS)



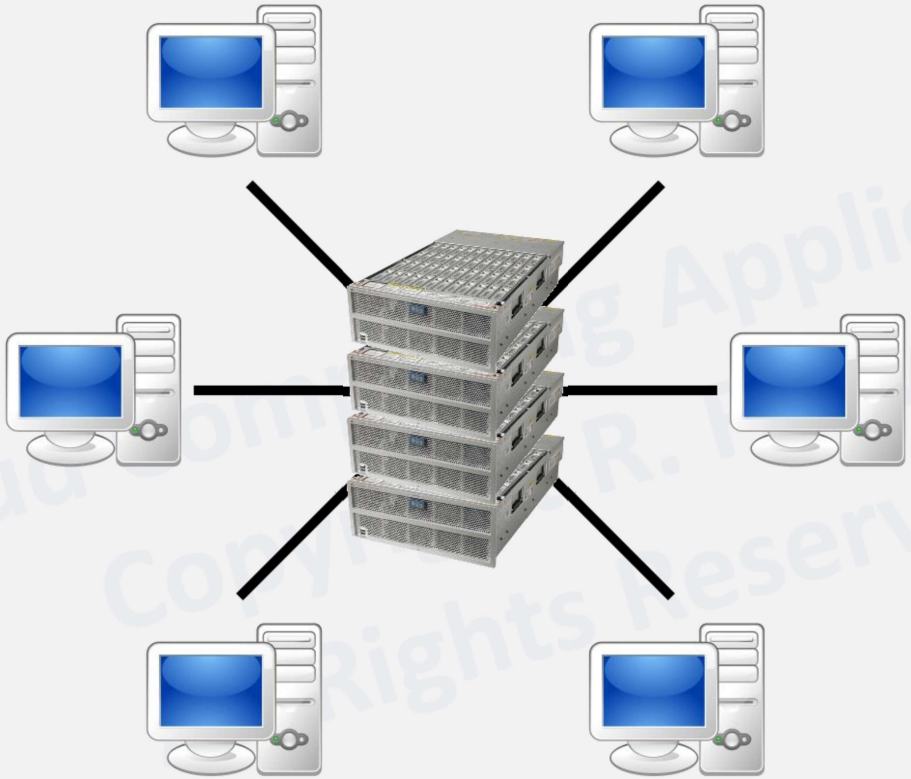
Clustered File Systems

- A clustered file system allows files to be accessed using the same interfaces and semantics as local files – for example, mounting/unmounting, listing directories, read/write at byte boundaries, system's native permission model.
  - Fencing
  - Concurrency
  - Consistency
- Examples
  - NFS
    - Unix
    - V4 , V4.1 (pNFS extension)
  - SMB
    - Windows
  - Lustre
    - HPC
  - Ceph
    - Many OpenStack implementations use Ceph as the storage substrate
  - Gluster
    - Classic file serving, second-tier storage, and deep archiving

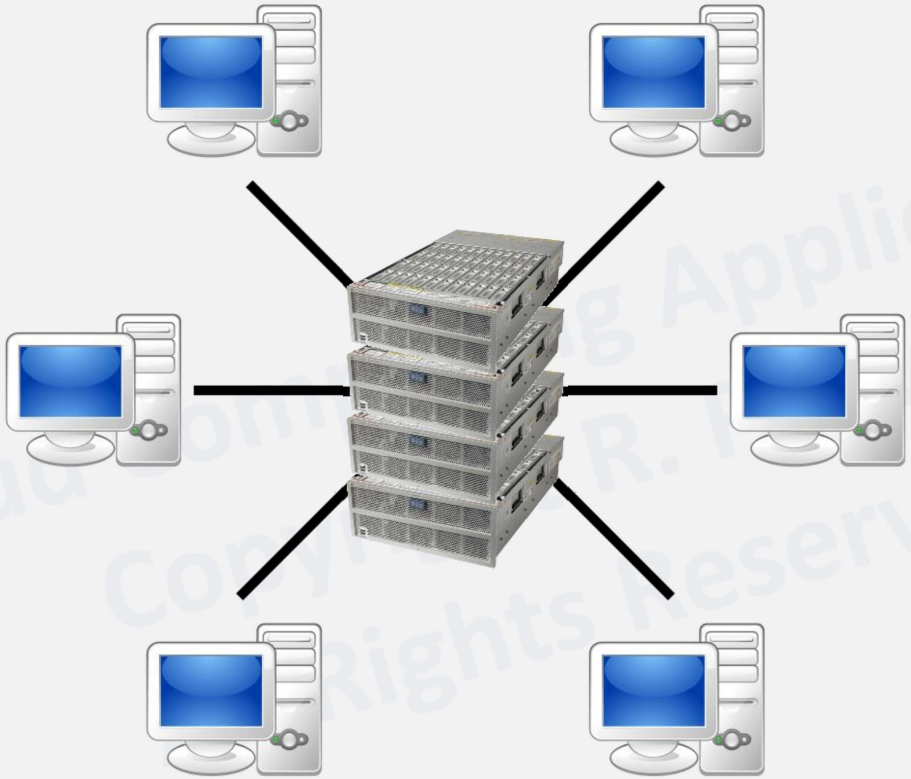
Clustered File System Consistency

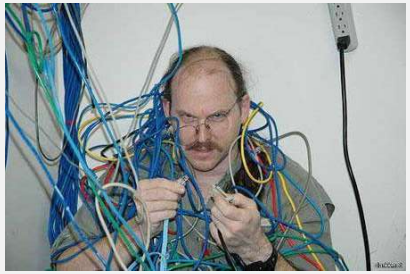
- Ideally, a parallel file system would completely hide the complexity of distributed storage and exhibit a behavior as it is specified by POSIX
  - Fencing
- Relatively easy on one Machine
- Much harder on a cluster of servers
- Maintaining CAP is extremely hard
  - **Consistency**: Every read receives the most recent write or an error
  - **Availability**: Every request receives a (non-error) response, without the guarantee that it contains the most recent write
  - **Partition tolerance**: The system continues to operate despite an arbitrary number of messages being dropped (or delayed) by the network between nodes

Network Attached Storage (NAS)

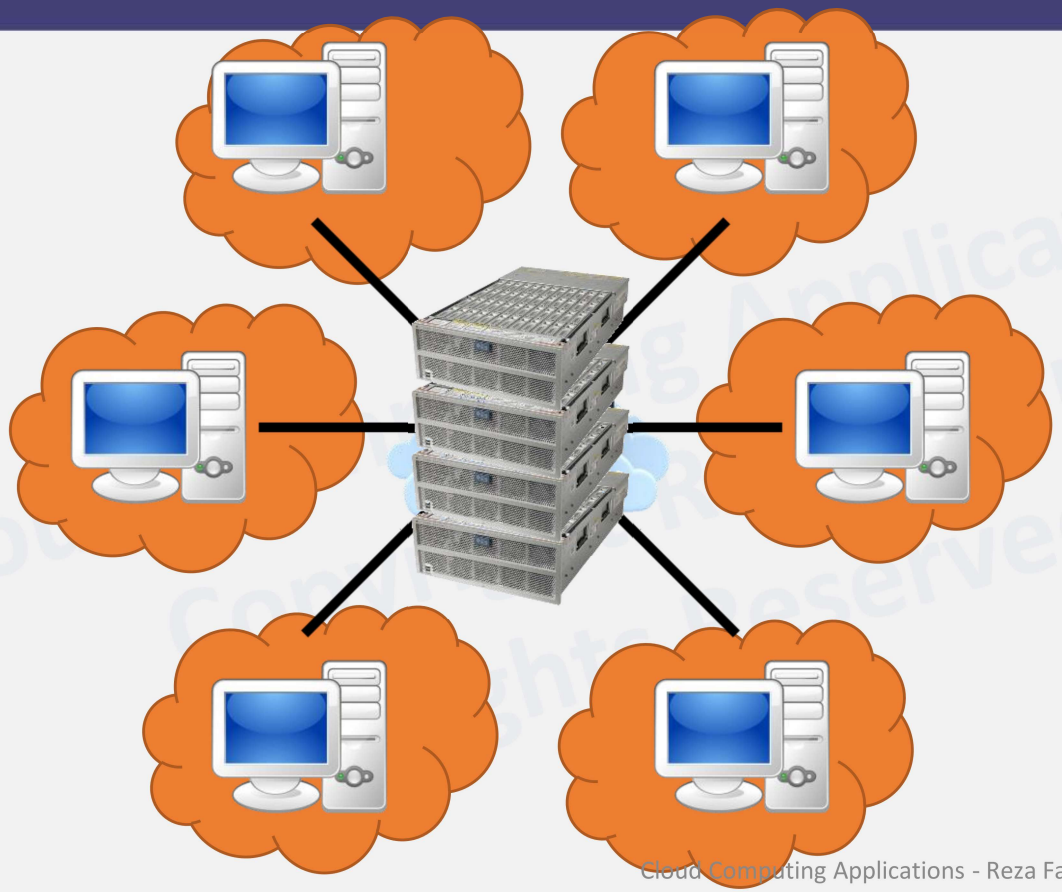


Network Attached Storage (NAS)



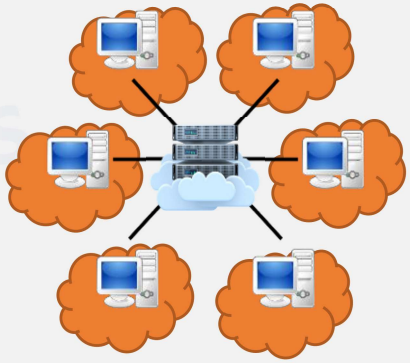


Network Attached Storage (NAS)



Cloud-Based File Systems

- Roll your own Clustered Distributed File System
  - Grab a number of “storage optimized” virtual machines, or metal machines
    - Each with large and fast instance block stores
      - A.k.a. SSDs / HDDs attached to the instance
    - Keep those instances on 24/7
    - Install a distributed file system on them
      - Lustre, MS DFS, NFS, Gluster, Ceph, Hadoop, etc.
- Managed filesystem deployed by the Cloud Provider



Cloud-managed file system

- Amazon
  - AWS FSx for Lustre
  - AWS FSx for Windows File Server
  - AWS EFS
- Azure
  - Azure Files
    - SMB access protocol
    - REST API
  - Azure DataLake Storage
    - Hadoop compatible file system
- Google
  - Cloud Filestore
    - NFS v3
    - Up to 64 TB

Google Filestore, IBM Cloud File Storage, AWS EFS

- Google Filestore

Simple commands to create a Filestore instance with gcloud.

```
gcloud filestore instances create nfs-server \
    --project=[PROJECT_ID] \
    --zone=us-central1-c \
    --tier=STANDARD \
    --file-share=name="vol1",capacity=1TB \
    --network=name="default"
```

Simple commands to install NFS, mount your file share, and set access permissions.

```
sudo apt-get -y update
sudo apt-get -y install nfs-common
sudo mkdir /mnt/test
sudo mount 10.0.0.2:/vol1 /mnt/test
sudo chmod go+rw /mnt/test
```

	Standard	Premium
Max read throughput	100 MB/s (1 TB), 180 MB/s (10+ TB)	1.2 GB/s
Max write throughput	100 MB/s (1 TB), 120 MB/s (10+ TB)	350 MB/s
Max IOPS	5,000	60,000
Max capacity per share	63.9 TB	63.9 TB
Typical customer availability	99.9%	99.9%
Protocol	NFSv3	NFSv3
Price	See Cloud Filestore pricing for more information	See Cloud Filestore pricing for more information

- IBM Cloud File Storage
  - Up to 12 TB
  - Up to 48,000 IOPS
- One beefy machine can handle them
  - AWS i3en.24xlarge: 8 x 7,500 NVMe SSD, 100 Gbps network
    - $60,405 / year per reserved instance
    - Cost of 64 TB standard storage EFS: $0.3*12*1024*64 = $235,929
    - Provisioned IO up to 1 GBps ~= 10 Gbps

Are Managed File Systems Distributed?

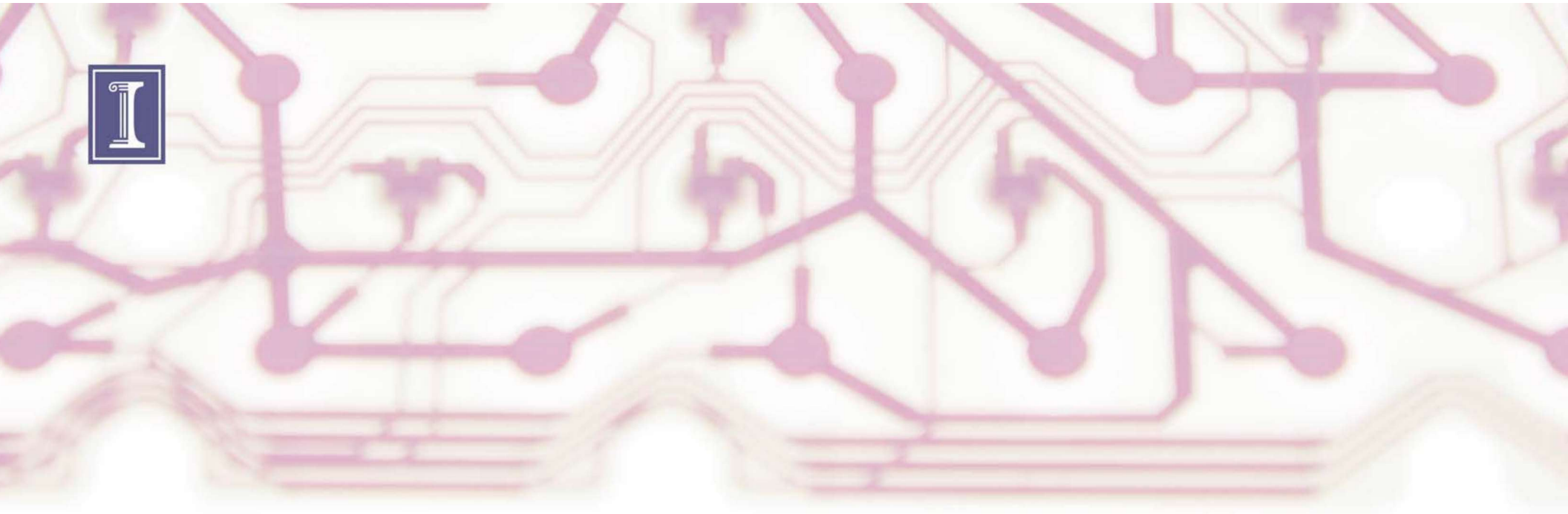
- FSx for Windows File Server
  - Max size: 64 TB
  - Can Utilize Microsoft DFS to unify data from many file servers for hundreds of petabytes
    - Shared namespace: Location transparency
    - Replication: Redundancy
- FSx for Lustre
  - Max size: 100 TB
  - Throughput: Read 50-200 MB/s per TB, can burst to 3,000 MB/s per TB
  - E.g. 50.4 TB runs on 22 file servers
    - Hundreds of GB/s of throughput
- AWS EFS
  - Maximum size: "Petabytes"
  - The throughput available to a file system scales as a file system grows
    - 50 MB/s per TB, can burst to 100 MB/s per TB
  - Supports NFS v4.1
- Azure
  - Azure Files
    - 100 TB
  - Data Lake Storage Gen 2
    - HDFS semantics
    - Built on top of Azure Blob Storage
    - Distributed file system
      - Can serve "many exabytes"
      - Throughput measured in gigabits per second (Gbps)

Distributed file Systems Design Goals

- *Access transparency*: clients are unaware that files are distributed and can access them in the same way as local files are accessed.
- *Location transparency*: a consistent namespace exists encompassing local as well as remote files. The name of a file does not give its location.
- *Concurrency transparency*: all clients have the same view of the state of the file system. This means that if one process is modifying a file, any other processes on the same system or remote systems that are accessing the files will see the modifications in a coherent manner.
- *Failure transparency*: the client and client programs should operate correctly after a server failure.
- *Heterogeneity*: file service should be provided across different hardware and operating system platforms.
- *Scalability*: the file system should work well in small environments (1 machine, a dozen machines) and also scale gracefully to bigger ones (hundreds through tens of thousands of systems).
- *Replication transparency*: Clients should be unaware of the file replication performed across multiple servers to support scalability.
- *Migration transparency*: files should be able to move between different servers without the client's knowledge.

Summary

- Clustered File Systems
- File Systems in the Cloud



CLOUD COMPUTING APPLICATIONS

Amazon AWS EFS

Roy Campbell & Reza Farivar

Amazon AWS EFS

- Elastic File System
- Motivation: enterprise customers need a large distributed file system
  - S3 is large and distributed, but it is an object store without performance guarantees and eventual consistency model
  - Block storage (EBS, instance store) are small
    - Enterprise can build a distributed file system on top of these, but it requires operational expertise
  - Glacier: Good for only archival storage
- EFS provides a fully NFSv4 compliant network file system

Amazon AWS EFS

- SSD backed
- Highly available and highly durable
  - files, directories, and links are stored redundantly across multiple Availability Zones within an AWS region
- Grow or shrink as needed
  - No need to pre-provision capacity

Amazon AWS EFS

		Amazon EFS	Amazon EBS PIOPS
Performance	Per-operation latency	Low, consistent	Lowest, consistent
	Throughput scale	Multiple GBs per second	Single GB per second
Characteristics	Data Availability/Durability	Stored redundantly across multiple AZs	Stored redundantly in a single AZ
	Access	1 to 1000s of EC2 instances, from multiple AZs, concurrently	Single EC2 instance in a single AZ
	Use Cases	Big Data and analytics, media processing workflows, content management, web serving, home directories	Boot volumes, transactional and NoSQL databases, data warehousing & ETL

CLOUD COMPUTING APPLICATIONS

Cloud Storage: Archive Storage and Backup

Prof. Reza Farivar

Cloud Archive Storage

- The availability of the storage has a direct impact on costs
- If you know file requests are very infrequent, you can store them in cold storage
- Cloud providers offer archive storage at much reduced costs
  - However, access to them is relatively expensive
- Different tiers of archive storage

Amazon Archive Storage: S3 Glacier

- AWS S3 Glacier
  - $0.004 per GB / mo
  - Compare with S3
    - Frequent access tier: $0.023 per GB / mo
    - Infrequent access tier: $0.0125 per GB / mo
- Glacier Retrieval
  - retrieval option from 1 minute to 12 hours

Retrieval Time	Data Retrievals
Expedited (typically available within 1 – 5 minutes)	$0.03 per GB / mo
Standard (typically accessible within 3 – 5 hours)	$0.01 per GB / mo
Bulk	$0.0025 per GB / mo

  - Retrieval request

Retrieval Time	Retrieval Requests
Expedited	$10.00 per 1,000 requests
Standard	$0.05 per 1,000 requests
Bulk	$0.025 per 1,000 requests

- S3 Glacier Deep Archive
  - For long-term data archiving that is accessed once or twice in a year and can be restored within 12 hours
  - $0.00099 per GB / mo

** Prices and bandwidths are a snapshot in time, might be different now*

AWS S3 tiers comparison

	S3 Standard	S3 Intelligent-Tiering*	S3 Standard-IA	S3 One Zone-IA†	S3 Glacier	S3 Glacier Deep Archive
Designed for durability	99.999999999% (11 9's)	99.999999999% (11 9's)	99.999999999% (11 9's)	99.999999999% (11 9's)	99.999999999% (11 9's)	99.999999999% (11 9's)
Designed for availability	99.99%	99.9%	99.9%	99.5%	99.99%	99.99%
Availability SLA	99.9%	99%	99%	99%	99.9%	99.9%
Availability Zones	≥3	≥3	≥3	1	≥3	≥3
Minimum capacity charge per object	N/A	N/A	128KB	128KB	40KB	40KB
Minimum storage duration charge	N/A	30 days	30 days	30 days	90 days	180 days
Retrieval fee	N/A	N/A	per GB retrieved	per GB retrieved	per GB retrieved	per GB retrieved
First byte latency	milliseconds	millseconds	milliseconds	milliseconds	select minutes or hours	select hours
Storage type	Object	Object	Object	Object	Object	Object
Lifecycle transitions	Yes	Yes	Yes	Yes	Yes	Yes

Service	Cost per 1TB / month
AWS EFS	$300
AWS FSx Lustre	$290
EBS gp2	$100
AWS EFS infrequent access	$25
S3 standard	**$23**
S3 infrequent	**$13**
S3 Glacier	$4
S3 Glacier deep archive	$1

** Prices and bandwidths are a snapshot in time, might be different now*

Cloud-managed Backups

- Some cloud providers allow managed backups from their block stores or managed file systems or other data stores
  - Distributed Backup is not a simple task
- Example: Amazon backup

Resource Type	Warm Storage	Cold Storage
Amazon EFS File System Backup	$0.05 per GB-Month	$0.01 per GB-Month
Amazon EBS File System Backup	$0.05 per GB-Month	n/a†
Amazon RDS Database Snapshot	$0.095 per GB-Month	n/a†
Amazon DynamoDB Table Backup	$0.10 per GB-Month	n/a†
AWS Storage Gateway Volume Backup	$0.05 per GB-Month	n/a†

- Amazon restore

Resource Type	Warm Storage	Cold Storage	Item-level Restore
Amazon EFS File System Backup	$0.02 per GB	$0.03 per GB	$0.5 per request
Amazon EBS Volume Snapshot	Free	n/a†	n/a**
Amazon RDS Database Snapshot	Free	n/a†	n/a**
Amazon DynamoDB Table Backup	$0.15 per GB	n/a†	n/a**
AWS Storage Gateway Volume Backup	Free	n/a†	n/a**

- Example: Azure
  - Only supports backups of VMs (block stores) and SQL workloads

CLOUD COMPUTING APPLICATIONS

Ceph

Roy Campbell & Reza Farivar

Motivation

- Ceph is an emerging technology in the production-clustered environment
- Designed for:
  - Performance – Striped data over data servers
  - Reliability – No single point of failure
  - Scalability – Adaptable metadata cluster
  - More general than HDFS
    - Smaller files
- GlusterFS has more or less similar ideas
  - Uses ring-based hashing; Ceph uses CRUSH

Ceph Overview

- MDS – Meta Data Server
- ODS – Object Data Server
- MON – Monitor (now fully implemented)

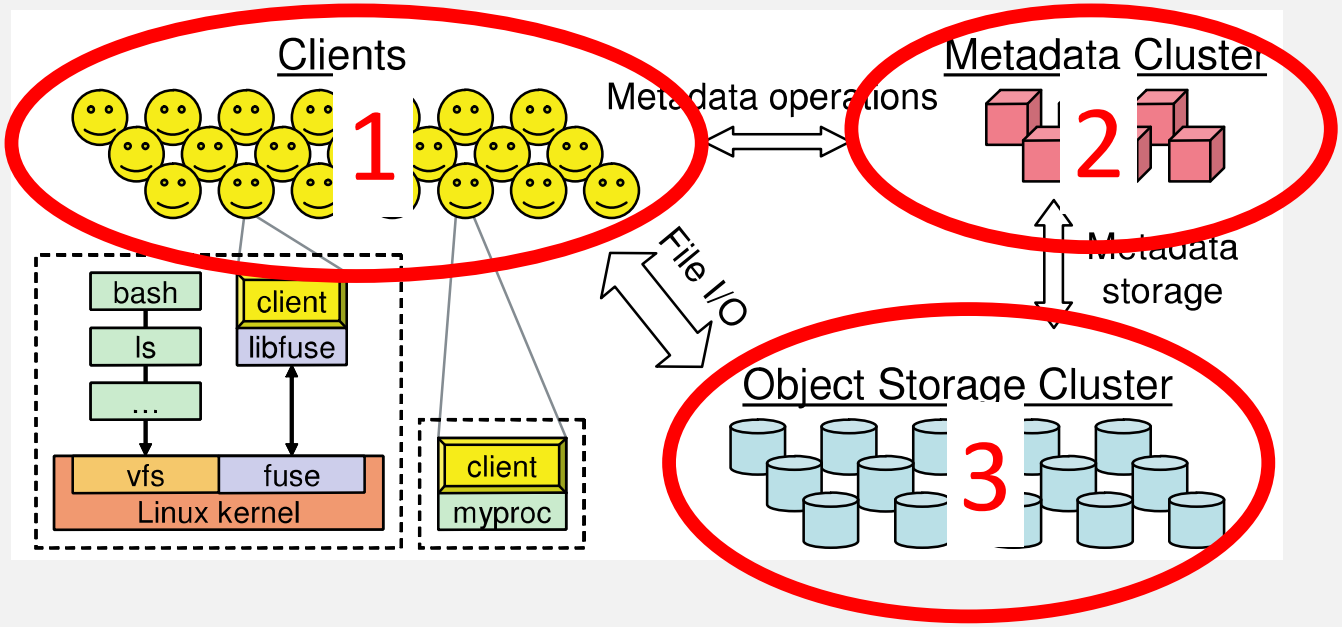
- **Decoupled data and metadata**
  - I/O directly with object servers

- **Dynamic distributed metadata management**
  - Multiple metadata servers handling different directories (subtrees)

- **Reliable autonomic distributed storage**
  - ODS's manage themselves by replicating and monitoring

Ceph Components

- Ordered: Clients, Metadata, Object Storage



Decoupled Data and Metadata

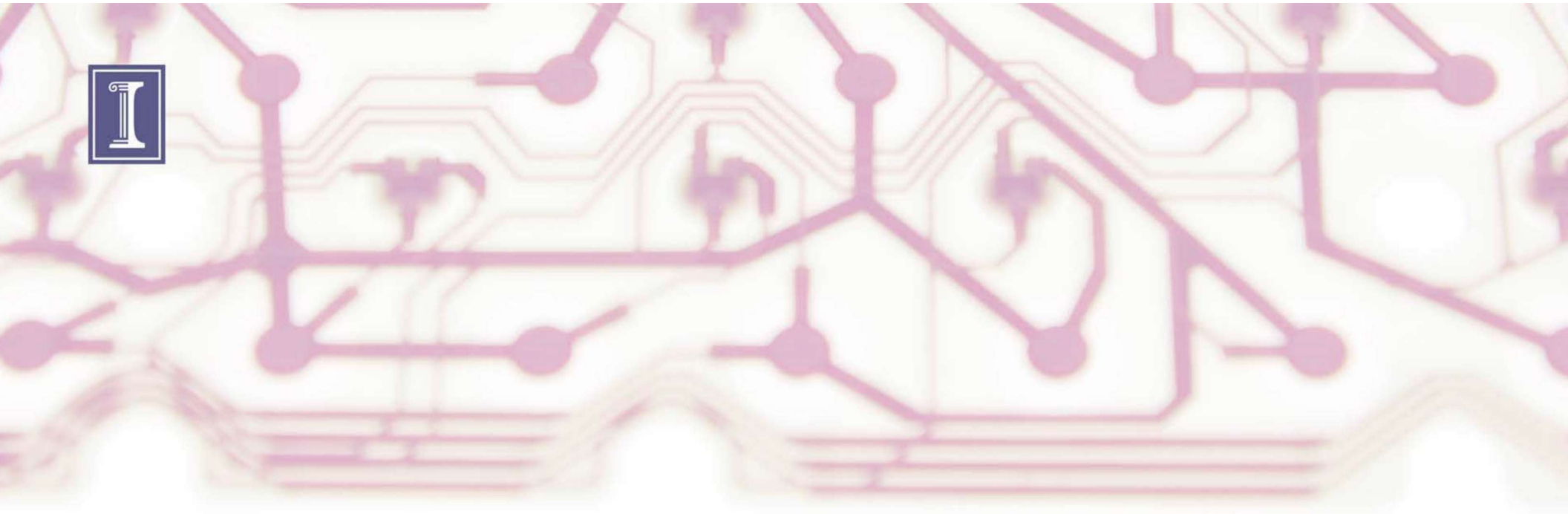
- Increases performance by limiting interaction between clients and servers
- Decoupling is common in distributed filesystems: HDFS, Lustre, Panasas…
- In contrast to other file systems, Ceph uses a function to calculate the block locations

Dynamic Distributed Metadata Management

- Metadata is split among cluster of servers
- Distribution of metadata changes with the number of requests to even load among metadata servers
- Metadata servers also can quickly recover from failures by taking over neighbors' data
- Improves performance by leveling metadata load

Reliable Autonomic Distributed Storage

- Data storage servers act on events by themselves
- Initiates replication and
- Improves performance by offloading decision making to the many data servers
- Improves reliability by removing central control of the cluster (single point of failure)



CLOUD COMPUTING APPLICATIONS

Amazon AWS Glacier

Roy Campbell & Reza Farivar

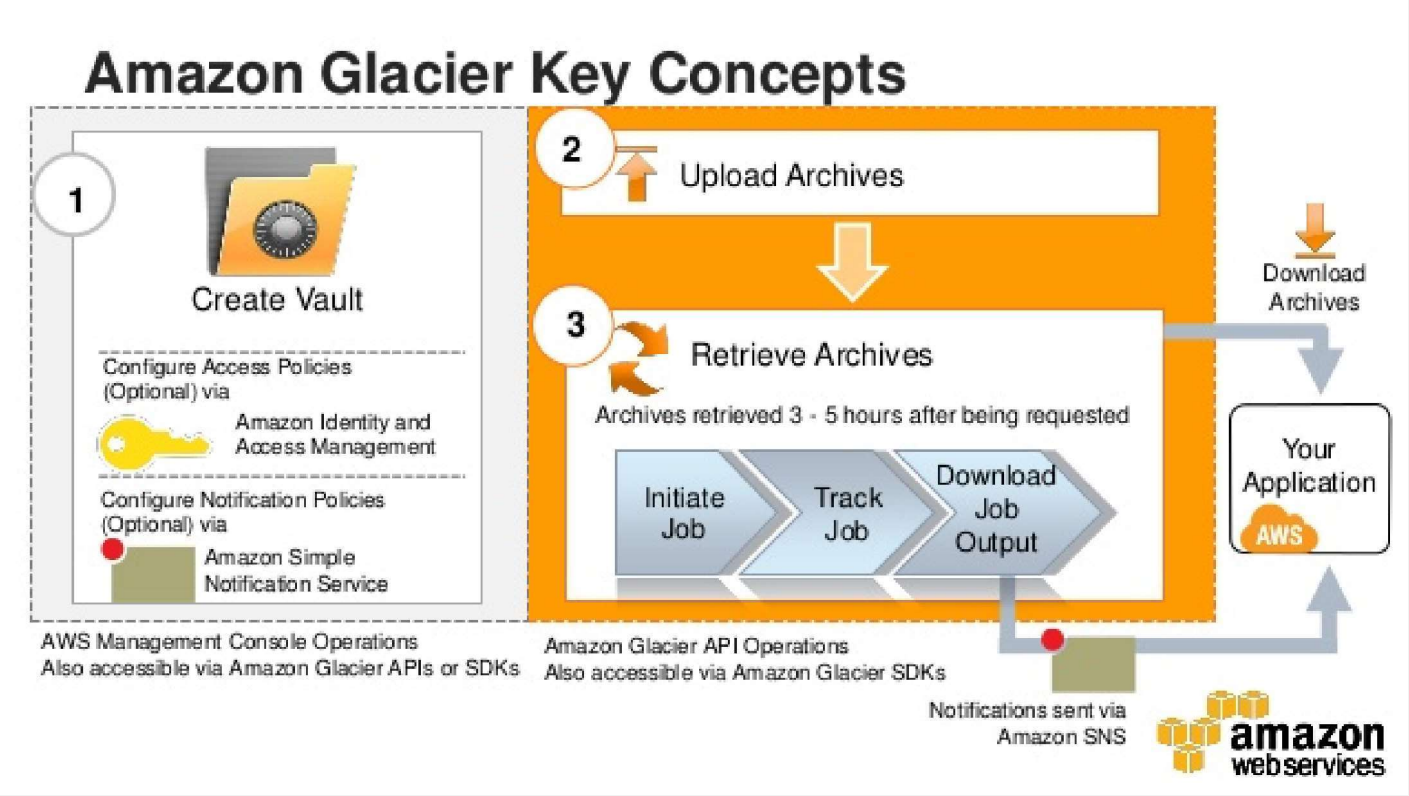
Amazon AWS Glacier

- Allows you to archive your data
- Very low cost, $0.007 per GB per month
- Very durable
  - Average annual durability of 99.999999999%
- Each single archive up to 40 TB

Amazon AWS Glacier

- Archives stored in vaults
- The main access point to glacier is S3
- Typically takes between **3 to 5 hours** to prepare a download request
  - After that you have 24 hours to download from the staging location

Amazon AWS Glacier



CLOUD COMPUTING APPLICATIONS

Cloud Storage: Storage Gateway and Mass Data Transfer

Prof. Reza Farivar

Cloud Storage Gateways

- Hybrid Cloud
  - Some VMs on your infrastructure, some on public cloud
  - Need a way to connect storage locations
- AWS Storage Gateway
  - File Gateway
    - NFS, SMB
    - S3 REST API access
      - POSIX-style metadata, including ownership, permissions, and timestamps stored as S3 user metadata
  - Tape Gateway
  - Volume Gateway
    - Presented as iSCSI protocol
    - Stores in AWS as EBS
- Azure
  - Data Box Gateway
    - SMB, NFS
    - Azure Blob Storage
    - Azure Files
  - File Sync
  - StorSimple

Cloud Mass Data Transfer

- Once a company decides to move to the cloud, there is a mass of data that needs to be transferred to the cloud
- Others may need to transfer data into and out of the cloud
- Problems:
  - The cost of mass data transfer could get astronomical
  - Network reliability could make it very hard
- Typical Solution: Cloud provider sends you a physical device
- AWS
  - Snowball: $250 / 80 TB
    - Moving data into Amazon is free, same as S3
    - In comparison, transferring 80TB into/out of S3 can take 7-8 days on a 1Gbps internet connection, if there is no data transmission error
      - 80*8*1024 / (60*60*24) = 7.58 days
    - Moving 80TB out of S3 can cost
      - 80*1024*0.08 = $6,553
    - Moving 80TB out through Snowball
      - 80*1024*0.03 = $2,457
  - Snowmobile
- Azure Data Box
  - Data Box: $250 / 100 TB
  - Data Box Disk: $50 / 8 TB
  - Data Box Heavy: $4,000 / 1 PB

AWS Snowball: 50-80 TB

- *Shipping fees also apply*

Cloud Mass Data Transfer

- Once a company decides to move to the cloud, there is a mass of data that needs to be transferred to the cloud
- Others may need to transfer data into and out of the cloud
- Problems:
  - The cost of mass data transfer could get astronomical
  - Network reliability could make it very hard
- Typical Solution: Cloud provider sends you a physical device
- AWS
  - Snowball: $250 / 80 TB
    - Moving data into Amazon is free, same as S3
    - In comparison, transferring 80TB into/out of S3 can take 7-8 days on a 1Gbps internet connection, if there is no data transmission error
      - 80*8*1024 / (60*60*24) = 7.58 days
    - Moving 80TB out of S3 can cost
      - 80*1024*0.08 = $6,553
    - Moving 80TB out through Snowball
      - 80*1024*0.03 = $2,457
  - Snowmobile
- Azure Data Box
  - Data Box: $250 / 100 TB
  - Data Box Disk: $50 / 8 TB
  - Data Box Heavy: $4,000 / 1 PB

AWS Snowmobile: 100 PB

Assuming a 7-day transfer time, bandwidth = 824 Gbps

- *Shipping fees also apply*

Summary

- Storage Gateway
- Mass Data Transfer

CLOUD COMPUTING APPLICATIONS

Cloud Storage: Internet-level Filesystem Storage

Prof. Reza Farivar

Internet-level Filesystem Storage

- Object Storage
  - AWS S3, Azure Blob Storage, etc.
- DropBox
- Google Cloud
- Box
- Apple iCloud Drive

Internet-level Filesystem Storage

- Synch and access your file system across a limited number of owned devices
- Block-level file copying
- Synch throttling
- Shared folders and links
- Data recovery (e.g. 30 days, 180 days, etc.)
- File Sync with local file system
- Web-based access
- Shared folders with other users
- Analytics services
  - Text search in documents and images
  - etc.

Data access frequency

- Unlike Object Stores (AWS S3, Azure Blobs, etc.) this market segment does not accept access pattern limits
- Cloud providers use statistics to extract average access patterns and set pricing
  - They may lose money on some customers that access / change their data all the time
  - In average, they have positive profit
- Case study: DropBox
  - Dropbox started as a startup by storing their customer's data on Amazon S3
  - IN 2016 Dropbox moved off of Amazon and onto their own datacenters for storage
  - The economy of scale had grown so far that their "cloudonomics" dictated they should build vs. rent
    - Think of Amazon AWS as an incubator for startups
    - Netflix moved off its CDN from third parties to its own network once it grew large enough
      - Netflix still uses AWS & Google Cloud for everything else: compute, analytics, etc.
      - No Datacenter

Do not include transfer costs

Service	Cost per 2TB / month
AWS EFS	$600
AWS EFS infrequent access	$50
S3 standard	$46
S3 infrequent	$25
S3 Glacier	$8
S3 Glacier deep archive	$2
Box	$15
Dropbox	$10
Google Cloud	$10
Amazon Drive	$10
Apple iCould Drive	$10
pCloud	$10
Microsoft OneDrive	$7
Sync.com	$7

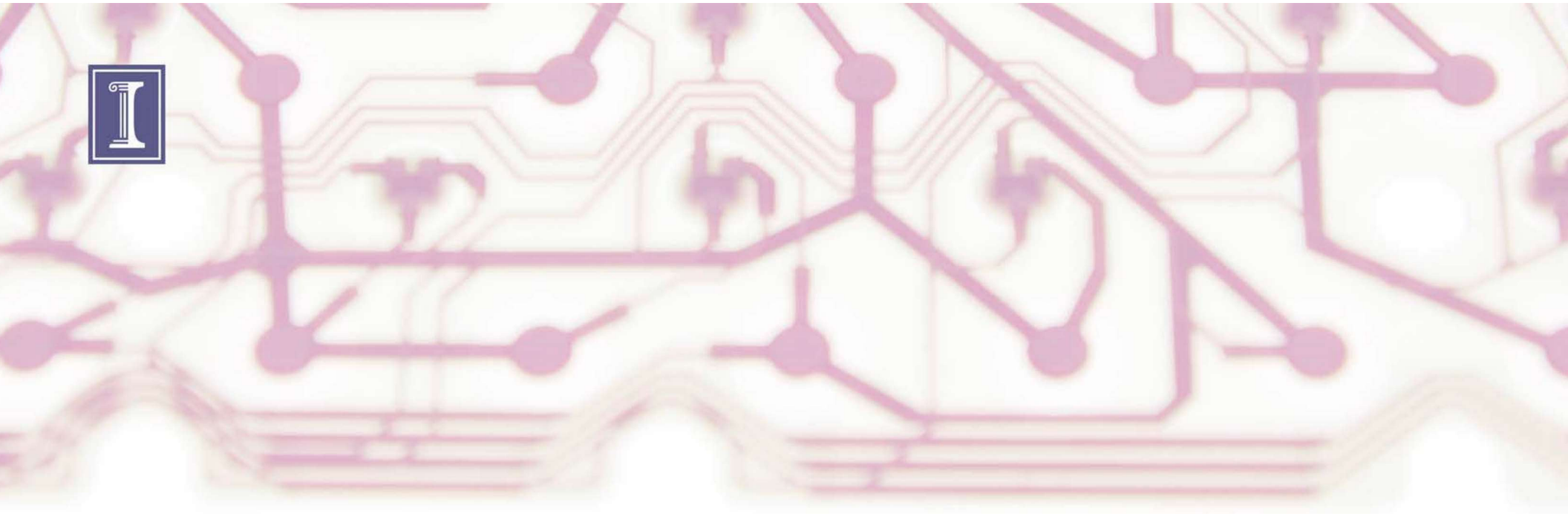
** Prices and bandwidths are a snapshot in time, might be different now*

Integration

- Custom integration in company ecosystem
  - Microsoft OneDrive: Comes with Office 365
  - Google Cloud: Integration with Google Docs, Gmail
  - Apple iCould Drive: Integration with iPhones, Macs
  - Dropbox: first to market, product typically more robust, integration with both MS Office and Google Docs
  - Box: Enterprise and government features

Summary

- End-user facing Cloud Storage
- Many competing offerings
- Utilize low frequency access pattern for low-cost offerings



CLOUD COMPUTING APPLICATIONS

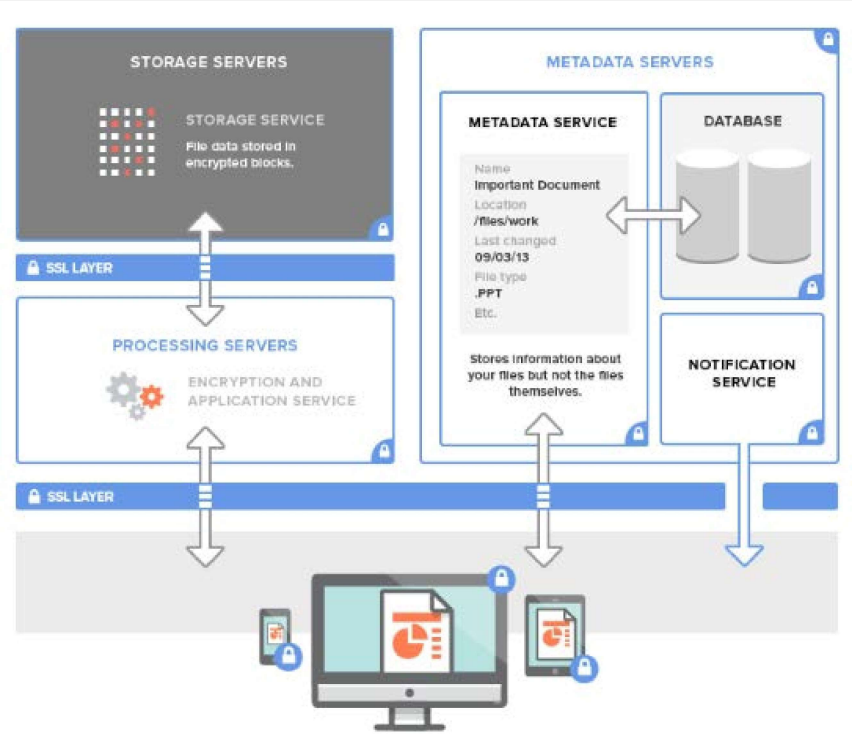
Dropbox Cloud API

Roy Campbell & Reza Farivar

Cloud Storage

- One interesting case study is Dropbox
- Dropbox offers cloud file storage
  - Easily synced across multiple devices
  - Accessible through web interface, mobile apps, and directly integrated with the file system on PCs
- Dropbox itself uses clouds!
  - Metadata stored in Dropbox servers
  - Actual files stored in Amazon S3
  - Amazon EC2 instances run the logic

Dropbox Architecture

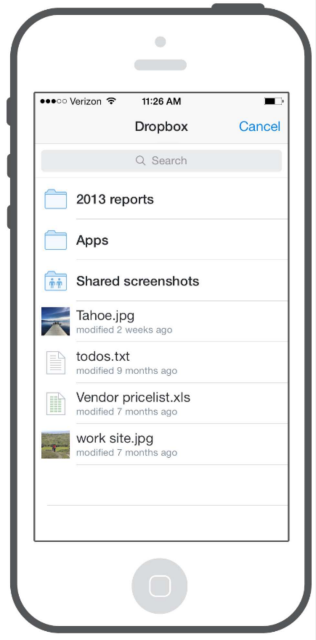


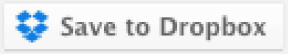
Dropbox API

- Two levels of API access to Dropbox
  - Drop-ins
    - Cross-platform UI components that can be integrated in minutes
    - *Chooser* allows instant access to files in Dropbox
    - *Saver* makes saving files to Dropbox easy
  - Core API
    - Support for advanced functionality like search, revisions, and restoring file
    - Better fit for deeper integration

Drop-In API

- Simple objects
  - *Chooser* available for JavaScript, Android and iOS
  - *Saver* on web and mobile web
- Handles all the authentication (OAuth), file browsing
- Chooser object returns the following:
  - Link: URL to access the file
  - File name
  - File Size
  - Icon
  - Thumbnails





- Saver
  - Pass in URL, filename and options

Core API

- Many languages and environments
  - Python, Ruby, PHP, Java, Android, iOS, OS X, HTTP
- Based on HTTP and OAuth
  - OAuth v1, OAuth v2
- Low-level calls to access and manipulate a user's Dropbox account
  - Create URL schemes
  - Upload files
  - Download files
  - List files and folders
  - Delta
  - Metadata access
  - Create and manage file sharing

CLOUD COMPUTING APPLICATIONS

Cloud Databases – Managed RDBMS

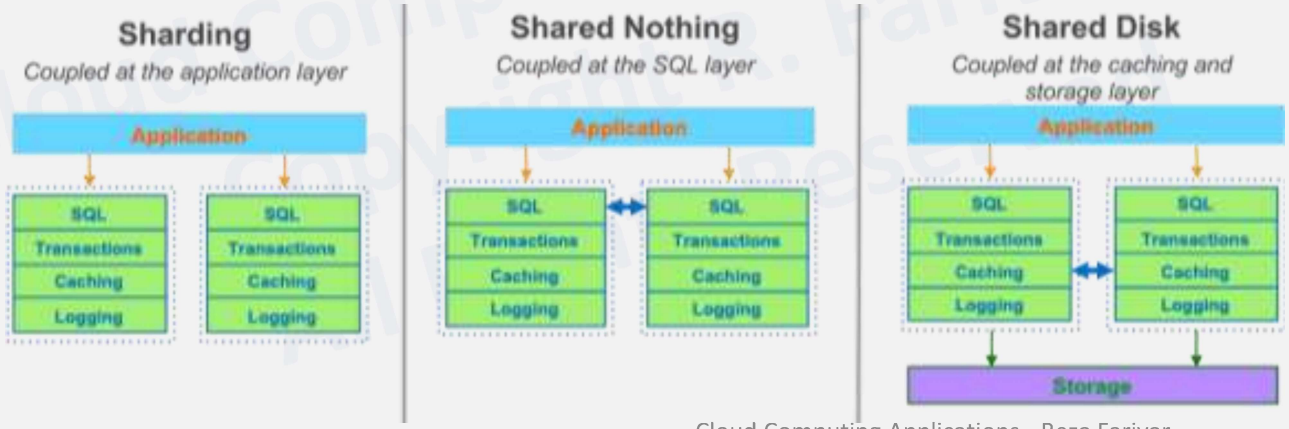
Prof. Reza Farivar

Relational Cloud Databases

- For decades, managing a relational database has been a high-skill, labor-intensive task
- Relational databases store data with predefined schemas and relationships between them
- These databases are designed to support ACID transactions, maintain referential integrity and strong data consistency.
  - Atomicity
  - Consistency
  - Isolation
  - Durability
- OLTP workloads
  - On- Line Transactional Processing (OLTP)

Relational Databases

- In large systems, failures are a norm, not an exception
- Users want to start with a small footprint and then grow massively without infrastructure limiting their velocity
- Replication
  - Storage (SAN, NAS, Aurora)
  - Database
  - Application



Sharding

- Sharding: Split data set by certain criteria and store such “shards” on separate “clusters”
  - Sharding can be considered an embodiment of the "share-nothing" architecture and essentially involves breaking a large database into several smaller databases
- One common way to split a database is splitting tables that are not joined in the same query onto different hosts
- Another method is duplicating a table across multiple hosts and then using a hashing algorithm to determine which host receives a given update

Managed Relational Databases

- Traditional single server databases running on a virtual machine
  - AWS RDS: MySQL, PostgreSQL, MariaDB, Oracle, MS SQL server
  - Azure SQL Database, Database for MySQL, PostgreSQL, MariaDB
  - Google Cloud SQL
  - IBM Cloud Databases for PostgreSQL, DB2 on cloud
- Instances are fully managed, relational MySQL, PostgreSQL, and SQL Server databases
- Cloud provider handles replication, patch management, and database management to ensure availability and performance
- Availability through failover
- Horizontal Scalability through read replicas
  - Vertical scalability by using larger machines (64 processors, 400GB RAM)

Managed Relational Databases

- Typically the database instance is accessible by most compute resources in the cloud provider's network
  - Virtual Machines (AWS EC2, Azure VMs, Google Compute Engine)
  - PaaS (Aws Elastic beanstalk, Google App Engine
  - Serverless (AWS lambda, Google Cloud Functions, Azure functions, etc.)
- Over the internet
  - SQL Proxy
  - Google Cloud SQL Proxy for public interfacing
    - ./cloud_sql_proxy -instances=***INSTANCE_CONNECTION_NAME***=tcp:3306 &
    - import pymysql
      connection = pymysql.connect(host='127.0.0.1',
      user='***DATABASE_USER***',
      password='***PASSWORD***',
      db='***DATABASE_NAME***')
- Encryption at rest and in transport

CLOUD COMPUTING APPLICATIONS

Cloud Databases – Multinode RDBMS

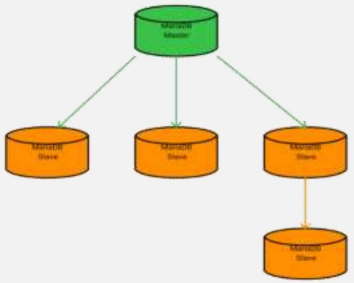
Prof. Reza Farivar

Beyond a Single Node

- Replication Options in MySQL
  - Classical MySQL Replication
    - One master, multiple slaves
    - Support for Many masters to many slaves
    - asynchronous
    - No Conflict Resolution or “Protection”
  - MySQL Group Replication
    - distributed state machine replication with strong coordination between servers
    - Built on Paxos
      - Majority vote for transaction commit
      - Network partition can stop the system
  - Multi Master - Galera
    - Multi-master
  - MySQL (NDB) Cluster
    - Synchronous

Replication in Databases

- Replication is a feature allowing the contents of one or more servers (called masters) to be mirrored on one or more servers (called slaves)
- **Scalability:** By having one or more slave servers, reads can be spread over multiple servers, reducing the load on the master.
  - The most common scenario for a high-read, low-write environment is to have one master, where all the writes occur, replicating to multiple slaves, which handle most of the reads.
- **Backup assistance:** Backups can more easily be run if a server is not actively changing the data.
  - A common scenario is to replicate the data to slave, which is then disconnected from the master with the data in a stable state. Backup is then performed from this server.



Replication and Binary Log

- The main mechanism used in replication is the binary log.
  - All updates to the database (data manipulation and data definition) are written into the binary log as binlog events.
  - The binary log contains a record of all changes to the databases, both data and structure, as well as how long each statement took to execute
    - It consists of a set of binary log files and an index
  - This means that statements such as CREATE, ALTER, INSERT, UPDATE and DELETE will be logged, but statements that have no effect on the data, such as SELECT and SHOW, will not be logged
- Slaves read the binary log from each master in order to access the data to replicate.

Database Replication

- A relay log is created on the slave server, using the same format as the binary log, and this is used to perform the replication
  - Old relay log files are removed when no longer needed
- A slave server keeps track of the position in the master's binlog of the last event applied on the slave
- This allows the slave server to re-connect and resume from where it left off after replication has been temporarily stopped
- It also allows a slave to disconnect, be cloned and then have the new slave resume replication from the same master
- There will be a measurable delay between the master and the replica. The data on the replica eventually becomes consistent with the data on the master
  - Use this feature for workloads that can accommodate this delay

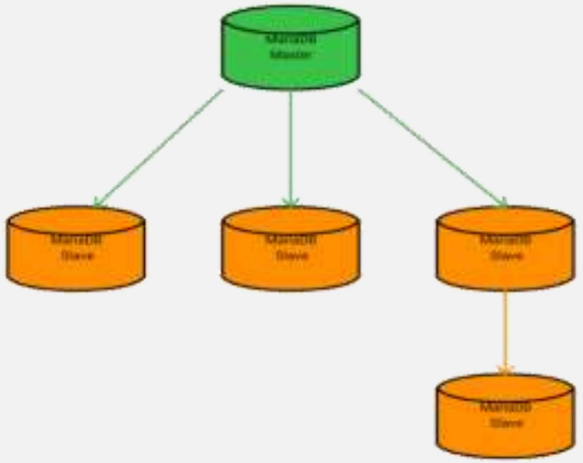
Replication Steps

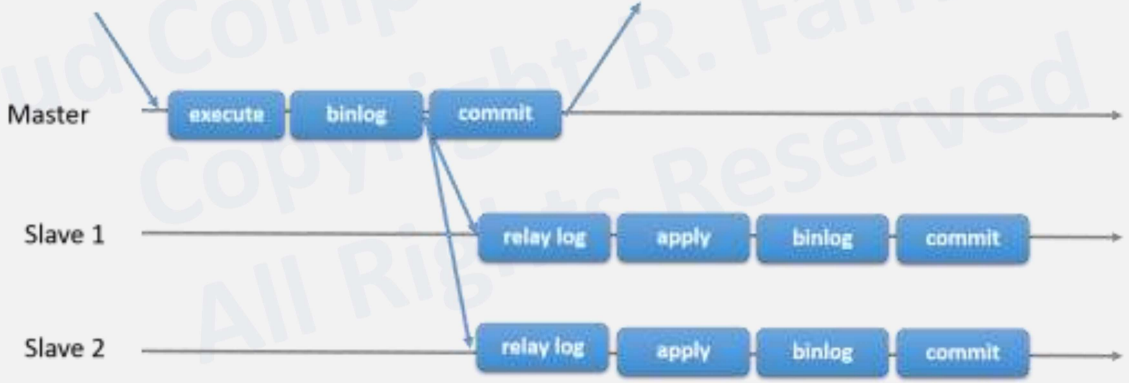
1. Replication events are read from the master by the IO thread and queued in the relay log
2. Replication events are fetched one at a time by the SQL thread from the relay log
3. Each event is applied on the slave to replicate all changes done on the master

- Replication is essentially asynchronous
  - The third step can optionally be performed by a pool of separate replication worker threads
    - **In-order** executes transactions in parallel, but orders the commit step of the transactions to happen in the exact same order as on the master
      - Transactions are only executed in parallel to the extent that this can be automatically verified
    - **Out-of-order** can execute and commit transactions in parallel
      - The application must be tolerant to seeing updates occur in different
      - Only when explicitly enabled by the application

Asynchronous Replication

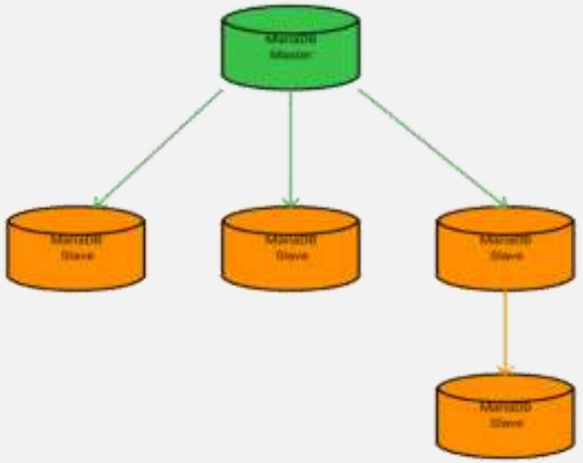
- A.k.a Standard replication
- Provides infinite read scale out
  - Most websites fit into this category, where users are browsing the website, reading articles, posts, or viewing products.
  - Updates only occur during session management, or when making a purchase or adding a comment/message to a forum.
- Provides high-availability by upgrading slave to master
- slaves read-only to ensure that no one accidentally updates them
- Eventual Consistency

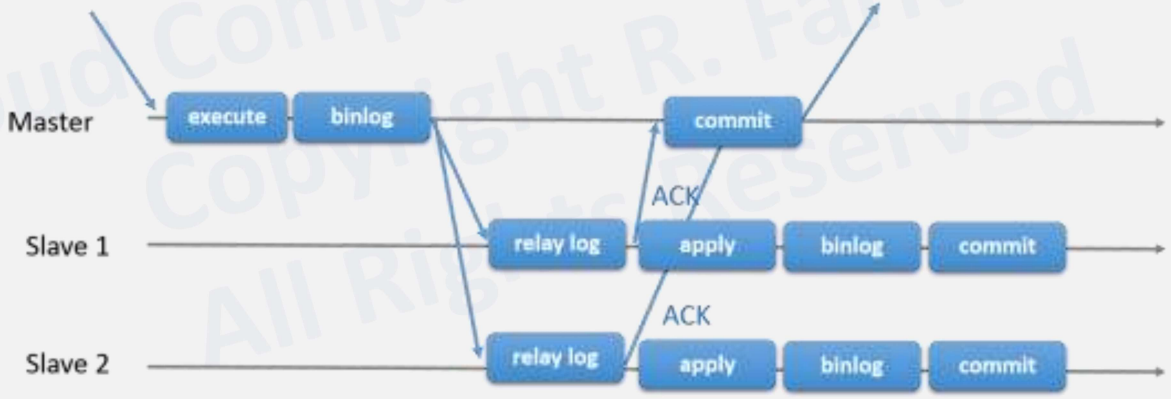




Semi-synchronous Replication

- Semi Synchronous Replication
- Better than eventual consistency
- The master waits for at least one ACK
  - Slower commits
- If no ACK received and timeout, master reverts to asynchronous
  - When at least one ACK is finally received, master goes back to semi-synchronous



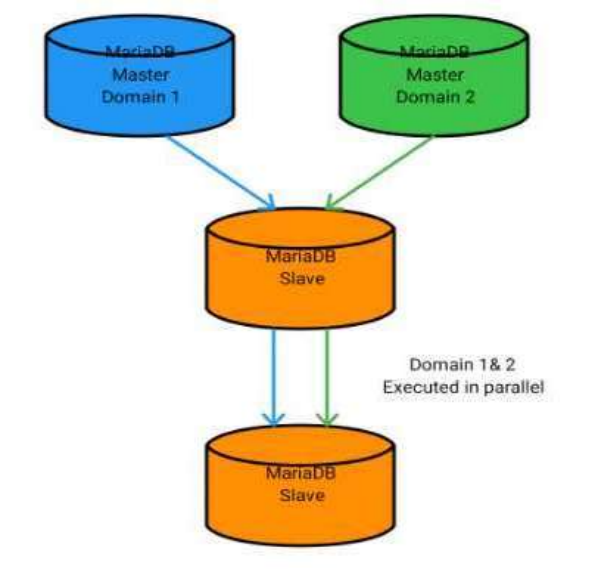


GTIDs-based Replication

- MySQL newer method based on global transaction identifiers (GTIDs)
- Transactional and therefore does not require working with log files or positions within these files
  - May simplify common replication tasks
- Replication using GTIDs guarantees consistency between master and slave as long as all transactions committed on the master have also been applied on the slave

Multi-Source Replication

- Multi-source replication means that one server has many masters from which it replicates
- Allows you to combine data from different sources
- Different domains executed independently in parallel on all slaves

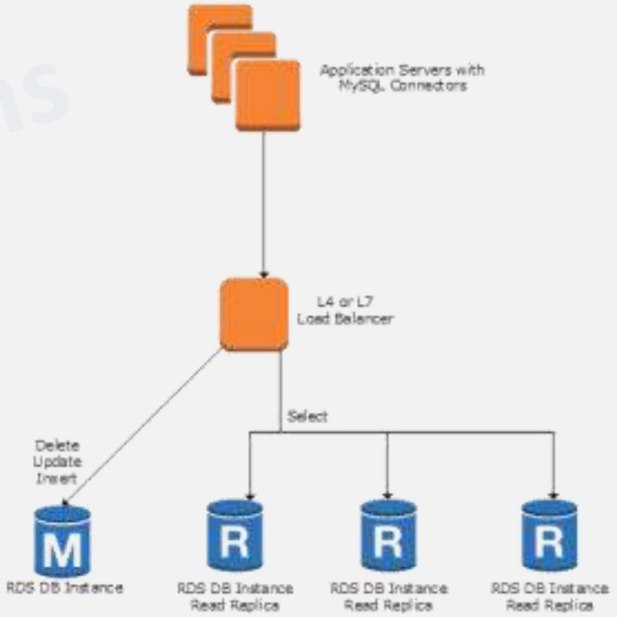


AWS RDS Horizontal Scaling

- Scaling beyond the compute or I/O capacity of a single DB instance for read-heavy database workloads
- Amazon RDS uses the MariaDB, MySQL, Oracle, PostgreSQL, and Microsoft SQL Server DB engines' built-in replication functionality to create a special type of DB instance called a read replica from a source DB instance
  - Up to five read replicas from one DB instance for MariaDB, MySQL
    - Similar limit of 5 replicas in Azure
    - Aurora allows 15 read replicas
  - specify an existing DB instance as the source
  - Amazon RDS takes a snapshot of the source instance and creates a read-only instance from the snapshot
  - Amazon RDS then uses the asynchronous replication method for the DB engine to update the read replica whenever there is a change to the source DB instance
- Updates to the source DB instance are **asynchronously** copied to the read replica
- If the read replica resides in a different AWS Region than its source DB instance, Amazon RDS sets up a secure communications channel between the source and the read replica
  - Amazon RDS establishes any AWS security configurations needed to enable the secure channel, such as adding security group entries
- Replicas can be “promoted” to full databases
  - They will reboot first

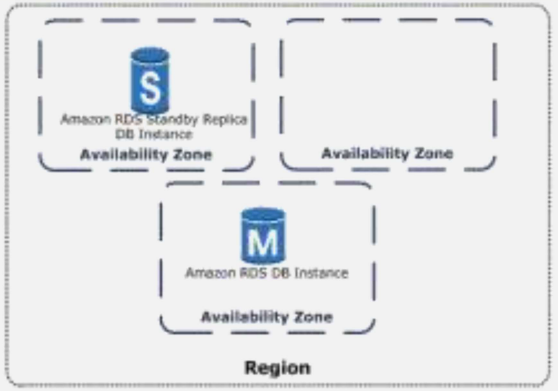
Load Balancing between RDS replicas

- Application-level load balancing
  - Different DNS record-sets using Route 53
- MySQL Connectors
  - If using the native MySQL driver, there are MySQL Connectors that allow read/write splitting and read-only endpoint load balancing without a major change in the application
- ELB does not support multiple RDS instances
- Level 4 proxy solutions
  - HAProxy: configure HAProxy to listen on one port for read queries and another port for write queries
- Level 7 proxy solutions
  - more sophisticated capability of understanding how to properly perform the read/write splits on multi-statements than a MySQL Connector does
  - This solution handles the scaling issues in a distributed database environment, so you don't have to handle scaling on the application layer, resulting in little or no change to the application itself
  - Several open-source solutions (such as MaxScale, ProxySQL, and MySQL Proxy) and also commercial solutions, some of which can be found in the AWS Marketplace



High Availability (Multi-AZ) for Amazon RDS

- Amazon RDS uses several different technologies to provide failover support
  - Multi-AZ deployments for MariaDB, MySQL, Oracle, and PostgreSQL DB instances use Amazon's failover technology
  - SQL Server DB instances use SQL Server Database Mirroring (DBM) or Always On Availability Groups (AGs)
- The high-availability feature is not a scaling solution for read-only scenarios

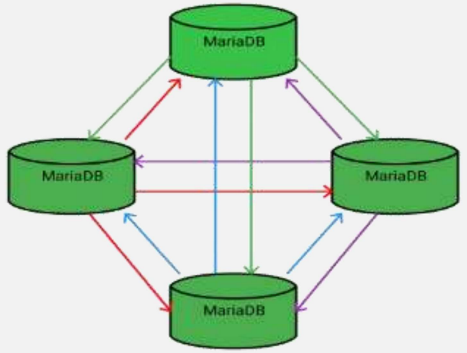


Read replicas, Multi-AZ deployments, and multi-region deployments (Amazon AWS)

Multi-AZ deploymentss	Multi-Region deployments	Read replicas
Main purpose is high availability	Main purpose is disaster recovery and local performance	Main purpose is scalability
Non-Aurora: synchronous replication; Aurora: asynchronous replication	Asynchronous replication	Asynchronous replication
Non-Aurora: only the primary instance is active; Aurora: all instances are active	All regions are accessible and can be used for reads	All read replicas are accessible and can be used for readscaling
Non-Aurora: automated backups are taken from standby; Aurora: automated backups are taken from shared storage layer	Automated backups can be taken in each region	No backups configured by default
Always span at least two Availability Zones within a single region	Each region can have a Multi-AZ deployment	Can be within an Availability Zone, Cross-AZ, or Cross-Region
Non-Aurora: database engine version upgrades happen on primary; Aurora: all instances are updated together	Non-Aurora: database engine version upgrade is independent in each region; Aurora: all instances are updated together	Non-Aurora: database engine version upgrade is independent from source instance; Aurora: all instances are updated together
Automatic failover to standby (non-Aurora) or read replica (Aurora) when a problem is detected	Aurora allows promotion of a secondary region to be the master	Can be manually promoted to a standalone database instance (non-Aurora) or to be the primary instance (Aurora)

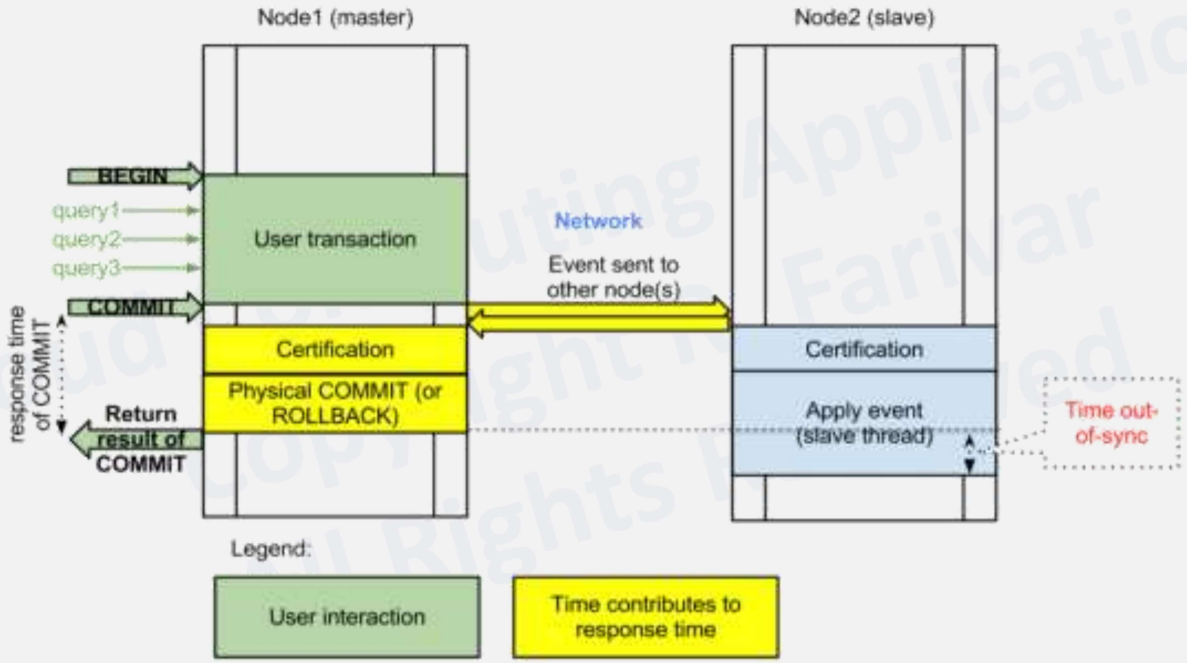
Multi-Master Cluster

- Galera (MySQL, MariaDB)
- Synchronous replication
- Active-active multi-master topology
- Read and write to any cluster node
- Automatic membership control, failed nodes drop from the cluster
- Automatic node joining
- True parallel replication, on row level
- Direct client connections, native MariaDB look & feel



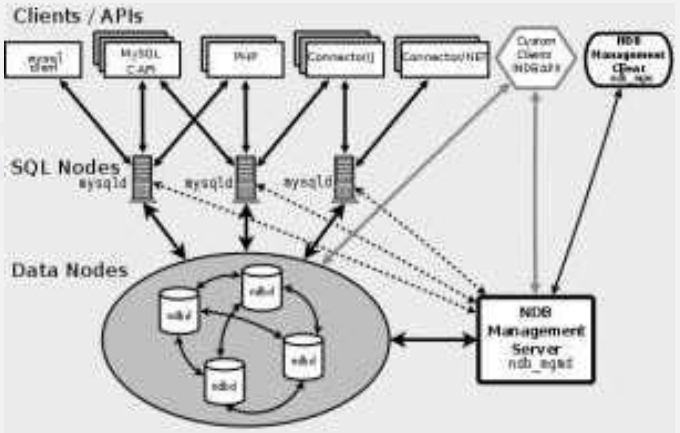
Galera Transaction Commit Flow

- Certification Based Replication
- Virtually Synchronous



MySQL NDB Cluster

- Network DataBase Engine
  - Replaces InnoDB
  - Separation of Compute and Data
    - SQL Nodes
    - Data Nodes
  - Shared Nothing Archiecture



CLOUD COMPUTING APPLICATIONS

Cloud Databases – Amazon Aurora

Prof. Reza Farivar