Seminar Presentation for ECE 658 Instructed by: Prof.Anura Jayasumana

Distributed File Systems

Prabhakaran Murugesan

Outline

- ❑ File Transfer Protocol (FTP)
- ❑ Network File System (NFS)
- ❑ Andrew File System (AFS)
- ❑ Performance Comparison between FTP, NFS and AFS
- ❑ Google File System (GFS) ✓
- ❑ Amazon Dynamo ✓
- ❑ Hadoop Distributed File System (HDFS) ✓ ✓✓
- ❑ Conclusion

What is DFS?

- A Distributed File System (DFS) enables programs to store and access remote files exactly as they do local ones, allowing users to access files from any computer on a network.

- This is an area of active research interest today.

- The resources on a particular machine are local to itself. Resources on other machines are remote.

- A file system provides a service for clients. The server interface is the normal set of file operations: create, read, etc. on files.

Key Features of DFS

- □ Data sharing of multiple users
- □ User mobility
- □ Location transparency
- □ Location independence
- □ Backups and System Monitoring

time

Distributed File System Requirements

- ❑ Transparency
  - – Access transparency
  - – Location transparency
  - – Mobility transparency
  - – Performance transparency
  - – Scaling transparency
- ❑ Concurrent file updates
- ❑ File replication
- ❑ Hardware and operating system heterogeneity

Distributed File System Requirements

- ❑ Fault tolerance
- ❑ Consistency
- ❑ Security
- ❑ Efficiency

V.V. Imp

Look and feel about traditional File Systems

Three major file systems:

File Transfer Protocol(FTP)

- ✓ Connect to a remote machine and interactively send or fetch an arbitrary file.
- ✓ User connecting to an FTP server specifies an account and password.
- ✓ Superseded by HTTP for file transfer.

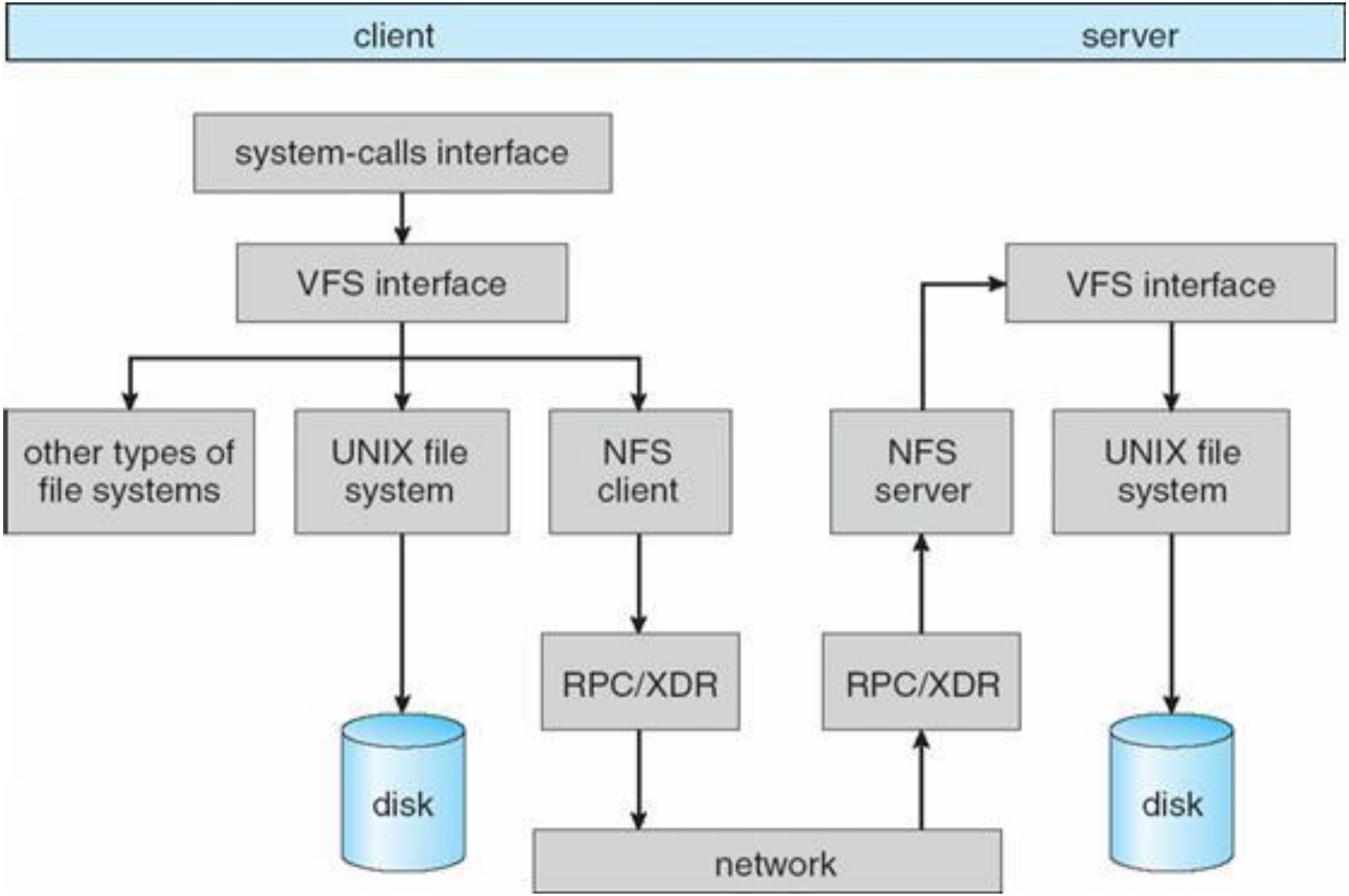
Sun's Network File System.

- One of the most popular and widespread distributed file system in use today since its introduction in 1985.
- Motivated by wanting to extend a Unix file system to a distributed environment. But, further extended to other OS as well.
- Design is transparent. Any computer can be a server, exporting some of its files, and a client , accessing files on other machines.
- High performance: try to make remote access as comparable to local access through caching and read-ahead.

Highlights of NFS

- NFS is stateless
  - All client requests must be self-contained
- The virtual file system interface
  - VFS operations
  - VNODE operations
- Fast crash recovery
  - Reason behind stateless design
- UNIX semantics at Client side
  - Best way to achieve transparent access

NFS Architecture



Andrew File System

- ❑ Distributed network file system which uses a set of trusted servers to present a homogeneous, location transparent file name space to all the client workstations.
- ❑ Distributed computing environment developed at Carnegie Mellon University (CMU) for use as a campus computing and information system [Morris et al. 1986].
- ❑ Intention is to support information sharing on a large scale by minimizing client-server communication
- ❑ Achieved by transferring whole files between server and client computers and caching them at clients until the servers receives a more up-to-date version.

Features of AFS

- Uniform namespace
- Location-independent file sharing
- Client-side caching
- Secure authentication
- Replication
- Whole-file serving
- Whole-file caching

How does AFS work?

- Implemented as 2 software components that exists as UNIX processes called Vice and Venus

- Vice: Name given to the server software that runs as a user-level UNIX process in each server computer.

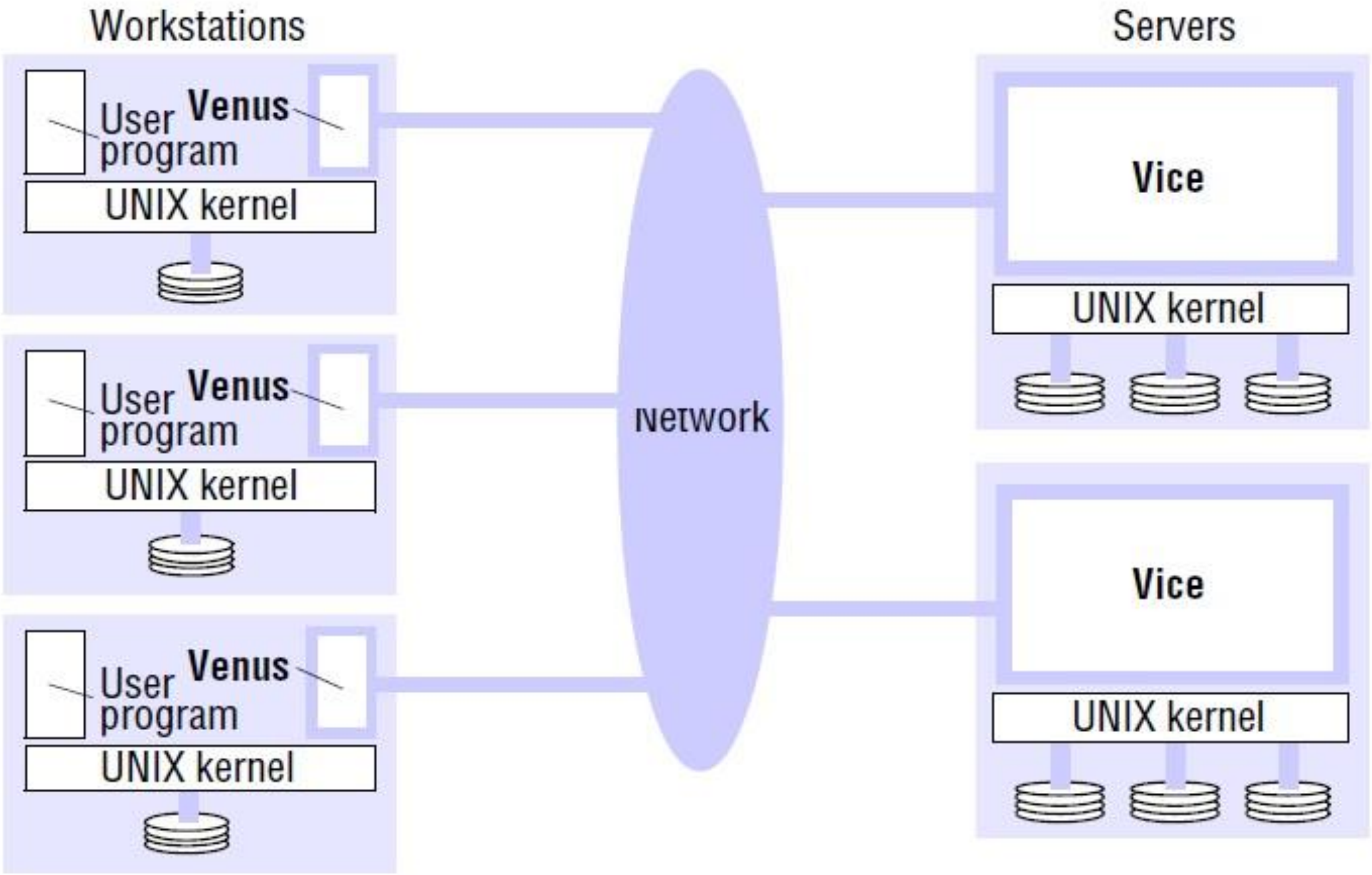
- Venus: User level process that runs in each client computer and corresponds to the client module in our abstract model.

- Files available to user are either *local* or *shared*

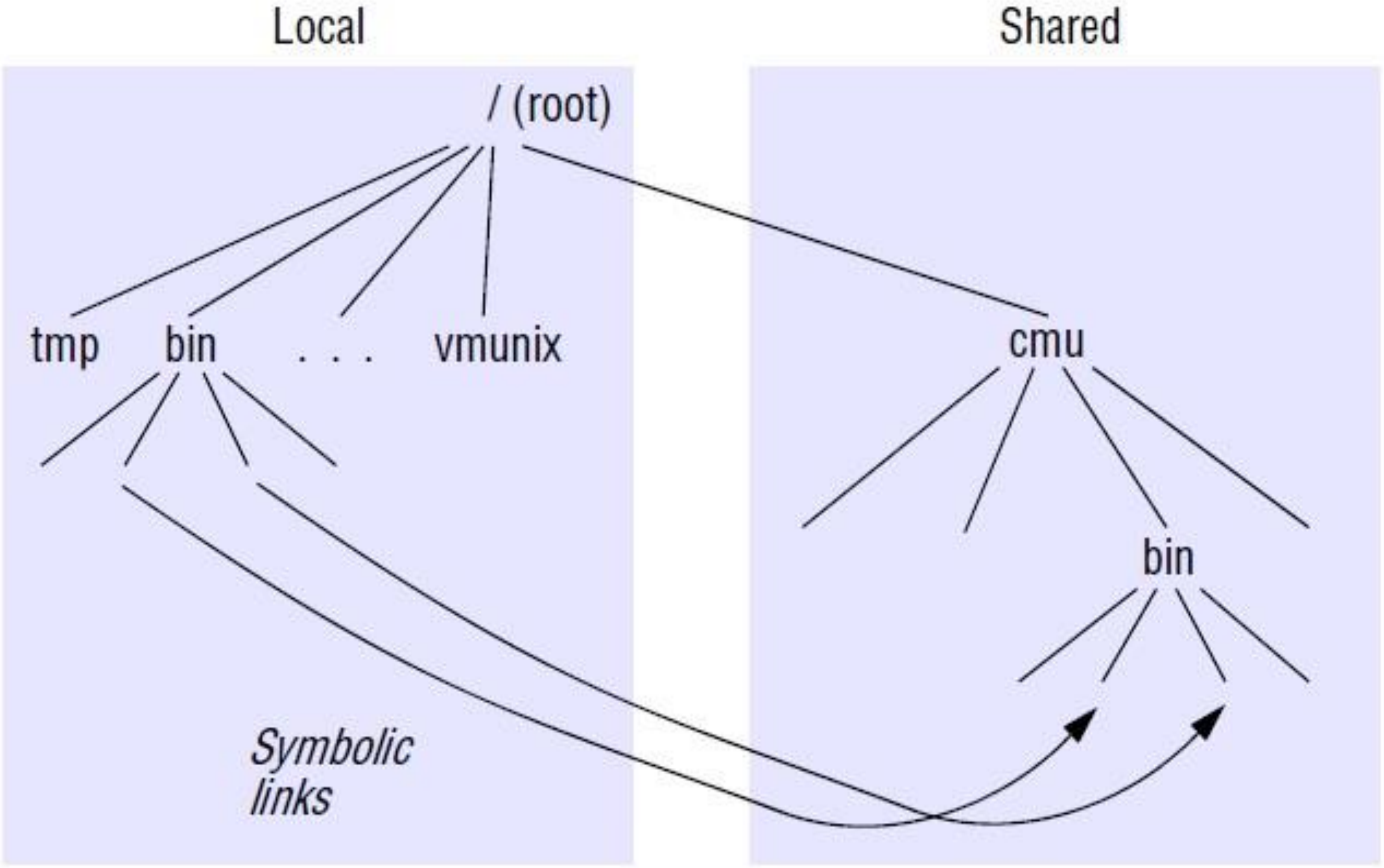
- Local files are stored on a workstation's disk and are available only to local user processes.

- Shared files are stored on servers, copies of them are cached on the local disk of work stations.

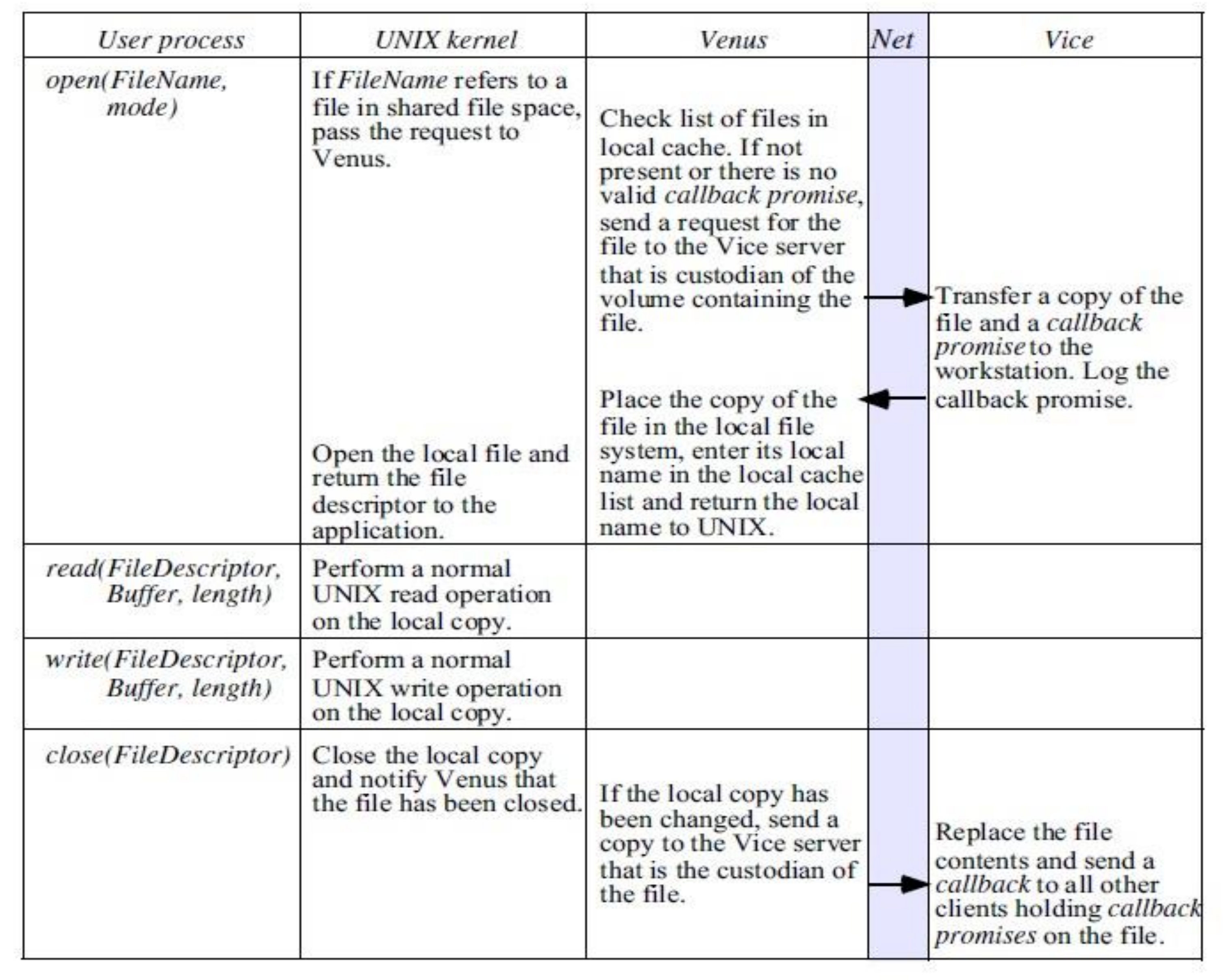
AFS Distribution of processes



File name space seen by clients of AFS



Implementation of File System Calls in AFS



User process	*UNIX kernel*	*Venus*	*Net*	*Vice*
open(FileName, mode)	If *FileName* refers to a file in shared file space, pass the request to Venus. Open the local file and return the file descriptor to the application.	Check list of files in local cache. If not present or there is no valid *callback promise*, send a request for the file to the Vice server that is custodian of the volume containing the file. Place the copy of the file in the local file system, enter its local name in the local cache list and return the local name to UNIX.	→ ←	Transfer a copy of the file and a *callback promise* to the workstation. Log the callback promise.
read(FileDescriptor, Buffer, length)	Perform a normal UNIX read operation on the local copy.			
write(FileDescriptor, Buffer, length)	Perform a normal UNIX write operation on the local copy.			
close(FileDescriptor)	Close the local copy and notify Venus that the file has been closed.	If the local copy has been changed, send a copy to the Vice server that is the custodian of the file.	→	Replace the file contents and send a *callback* to all other clients holding *callback promises* on the file.

How Cache Consistency done in AFS?

- ✓ When Vice supplies a copy of file to a venus process, it provides callback promise.
- ✓ Callback have 2 states: valid and canceled.
- ✓ When venus process receives callback, callback state is set to cancelled. If not, cached copy of the file is used.
- ✓ When workstation is restarted from failure, cache validation request is done. This contains file modification timestamp.
- ✓ If the timestamp is current, server responds with valid and token is reinstated. If not, server responds with cancelled and token is set to cancelled.
- ✓ Callback must be renewed before open.

Traditional Distributed File System Issues

Naming

Is the name access independent? location independent?

FTP: location and access independent.

NFS: location dependent through client mount points. Largely transparent for ordinary users, but the same remote file system could be mounted differently on different machines. Access independent.

AFS: location independent.

Migration and Directories

Can files be migrated between file server machines?

FTP: Sure, but end user must be aware.

NFS: Must change mount points on the client machines.

AFS: On a per volume basis.

How the directories are handled?

FTP: Directory listing handled as remote command.

NFS: Unix-like.

AFS: Unix-like.

Sharing Semantics and File locking

What type of file sharing semantics are supported if 2 process accessing the same file

FTP: User – level copies. No support.

NFS: Mostly unix semantics.

AFS: Session semantics.

Does the system support locking of files?

FTP: Not at all.

NFS: Has mechanism, but external to NFS in v3. Internal to file system in version 4.

AFS: Does support.

Caching and File Replication

Is file caching supported?

FTP: None. User has to maintain their own copy.

NFS: File attributes and file data blocks are cached separately. Cached attributes are validated with the server on file open.

AFS: File level caching with callbacks. Session semantics. Concurrent sharing is not possible.

Is file replication supported?

FTP: No.

NFS: minimal support in version 4.

AFS: For read only volumes within a cell.

Scalability and Security

Is the system scalable?

FTP: Yes. Millions of users.

NFS: Not so much.

AFS: Better than NFS. Keep traffic away from file servers.

What security features available?

FTP: Account/password authorization.

NFS: RPC Unix authentication that can use KERBEROS.

AFS: Unix permission for files, access control lists for directories.

State/Stateless and Homogeneity

Do file system servers maintain state about clients?

FTP: No.

NFS: No.

AFS: Yes

Is hardware/software homogeneity required?

FTP: No.

NFS: No.

AFS: No.

Other older File Systems

1. CODA: AFS spin-off at CMU. Disconnection and fault recovery.

2. Sprite: research project at UCB in 1980's. To build a distributed Unix system.

3. Echo. Digital SRC.

4. Amoeba Bullet File Server: Tanenbaum research project.

5. xFs: serverless file system—file system distributed across multiple machines. Research project at UCB.

~~The~~ Google File System (GFS)

Sanjay Ghemawat, Howard Gobioff,
and Shun-Tak Leung

Google

Google File Systems

❑ Distributed File System developed solely to meet the demands of Google's data processing needs.

❑ It is widely deployed within Google as the storage platform for the generation and processing of data used by Google's service and for research and development efforts which requires large datasets.

❑ While sharing many of the same goals as previous distributed file systems, design has been driven by observations of Google's application workloads and technological environment.

Assumptions

- High component failure rates
- Modest number of huge files
- File are write-once, mostly appended which may be done concurrently
- Large streaming reads and small random reads
- High sustainable throughput favored over low latency
- System must efficiently implement well defined semantics for multiple clients that concurrently append to the same file

GFS Design Decisions

Imp

- □ Files stored as chunks
  - ▪ Fixed size (64MB)
- □ Reliability through replication
  - ▪ Each chunk replicated across 3+ chunkservers Imp
- □ Single master to coordinate access, keep metadata
  - ▪ Simple centralized management
- □ No data caching
  - ▪ Little benefit due to large data sets, streaming reads
  - ▪ Client do cache metadata
- □ Familiar interface, but customize the API
  - ▪ Simplify the problem; focus on Google apps
  - ▪ Add snapshot and record append operations

Single Master

- ❑ Minimal involvement in reads and write to avoid bottleneck.

1st

- ❑ Easy to use global knowledge to reason about
  - ▪ Chunk placements
  - ▪ Replication decisions

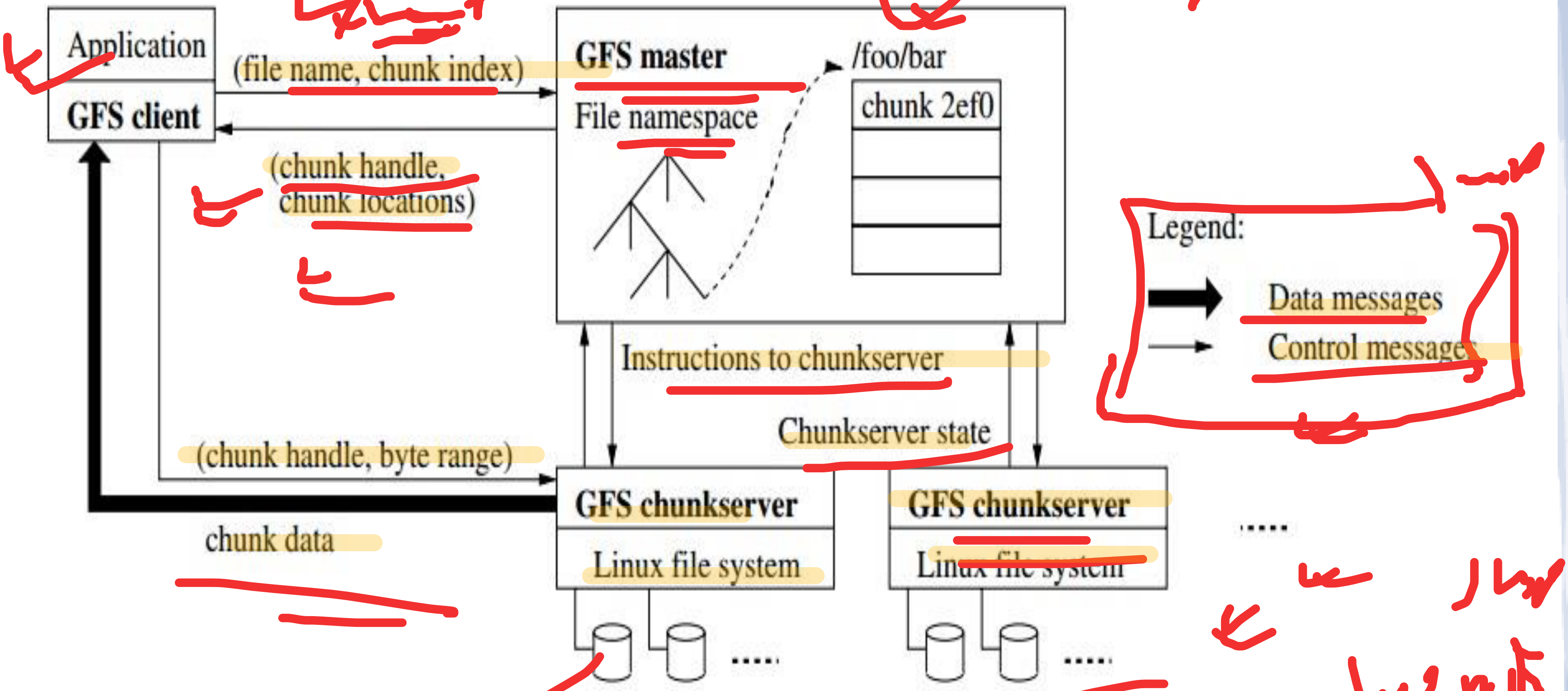
IInd

receivs

- ❑ Client caches information from Master so as to avoid multiple interaction

3rd

GFS Architecture



64 mb chunks

multiple chunk servers

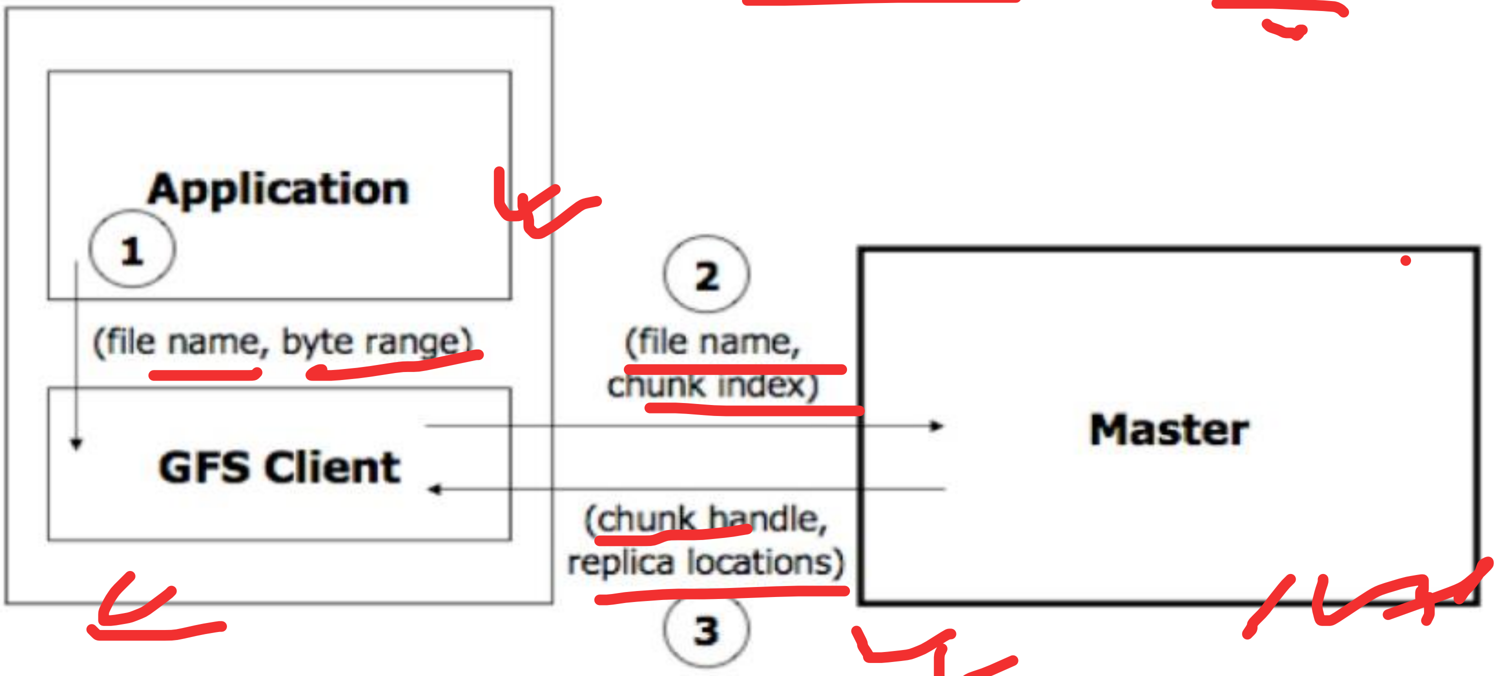
Why 64 MB?

- Much larger than typical file system block sizes
- Reduces client interaction with the master
    - Can cache info for Multi-TB working set
- Reduces network overhead
- Reduces the size of metadata stored in the master
    - 64 bytes of metadata per 64 MB chunk

imp

Read Algorithm

imp

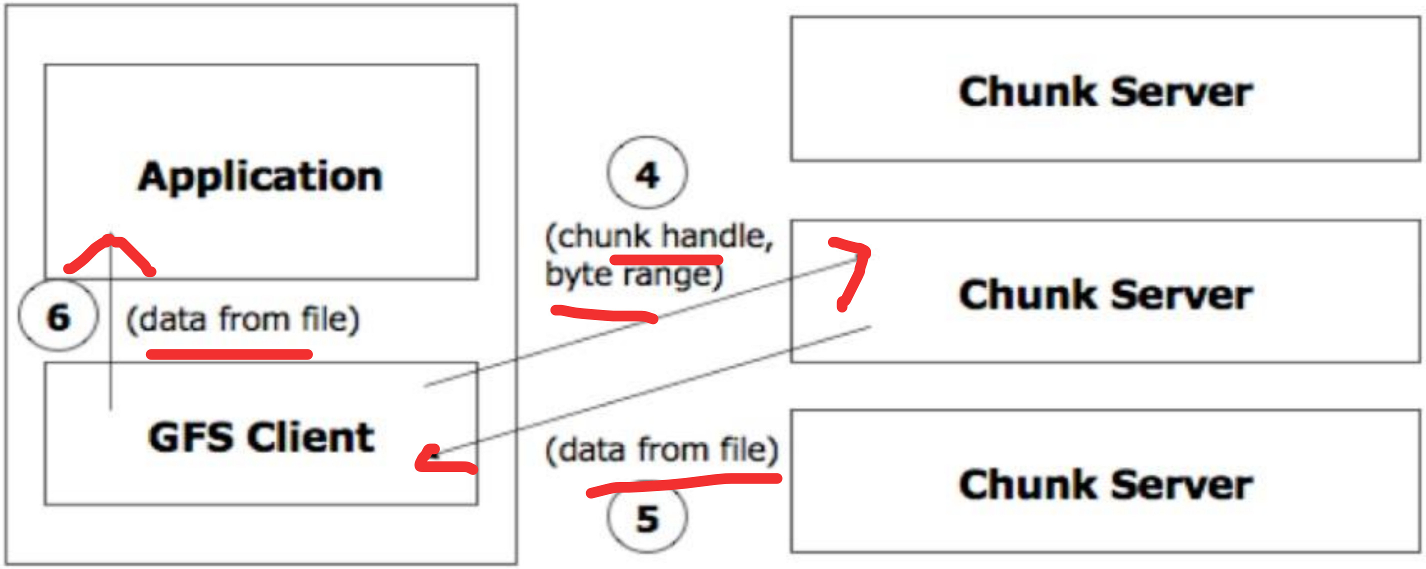


1. Application originates read request

2. GFS client translates request and sends it to master

3. Master responds with chunk handle and replica locations

Read Algorithm



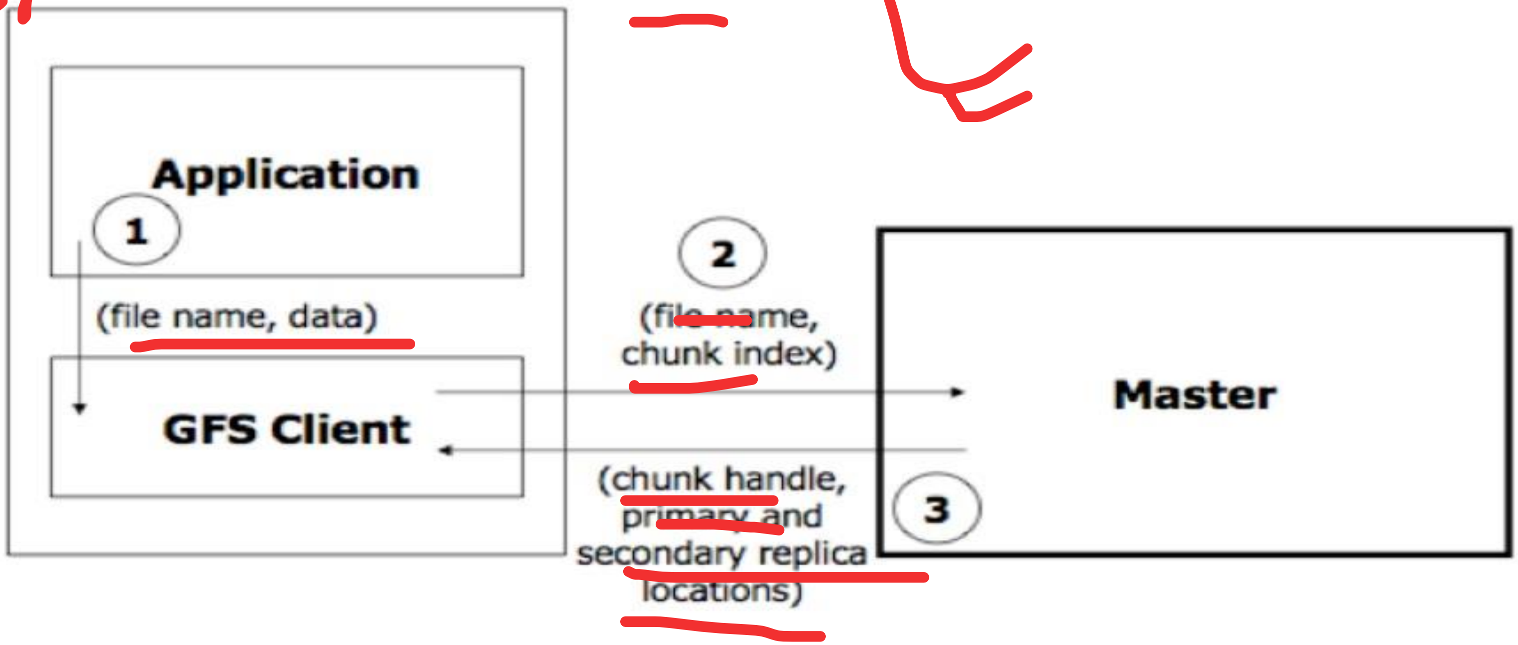
4. Client picks a location and sends the request

5. Chunkserver sends requested data to the client

6. Client forwards the data to the application

Write Algorithm

3 step



1. Application originates the request

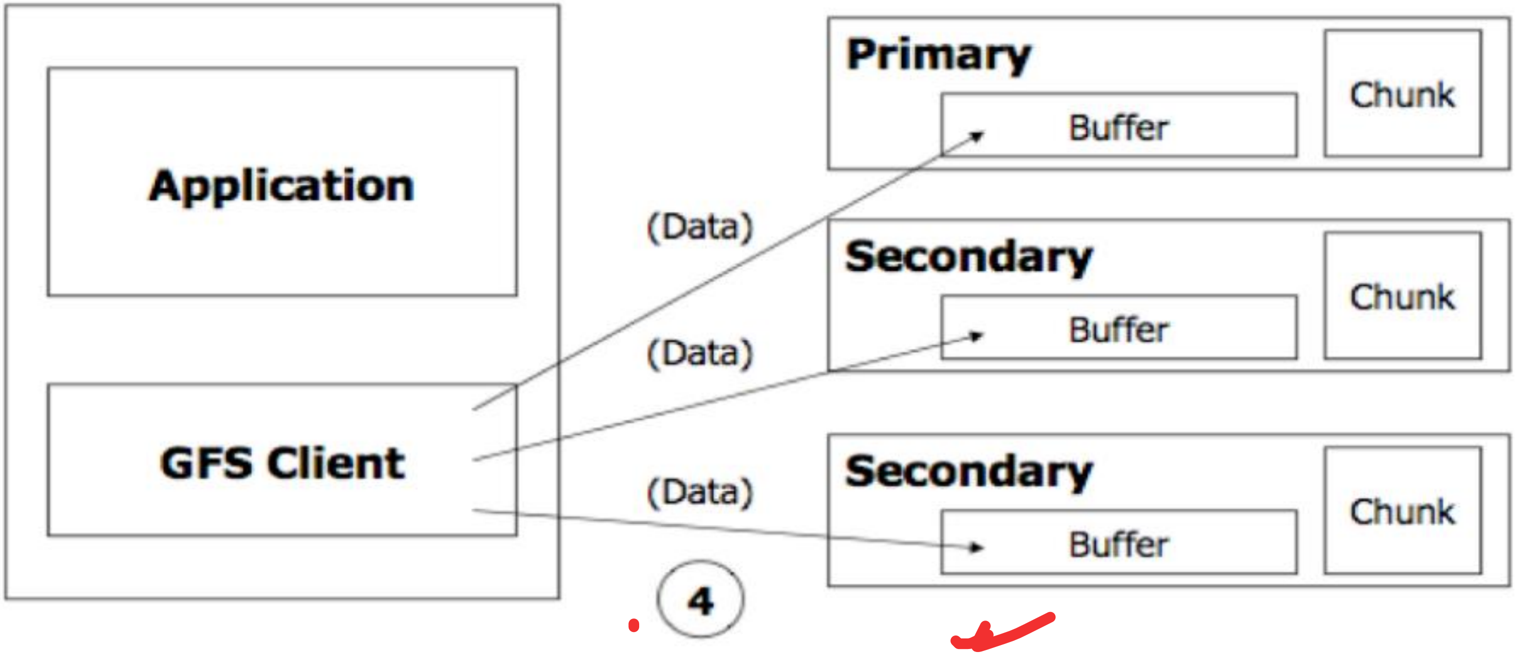
2. GFS client translates request and sends it to master

3. Master responds with chunk handle and replica locations

Imp

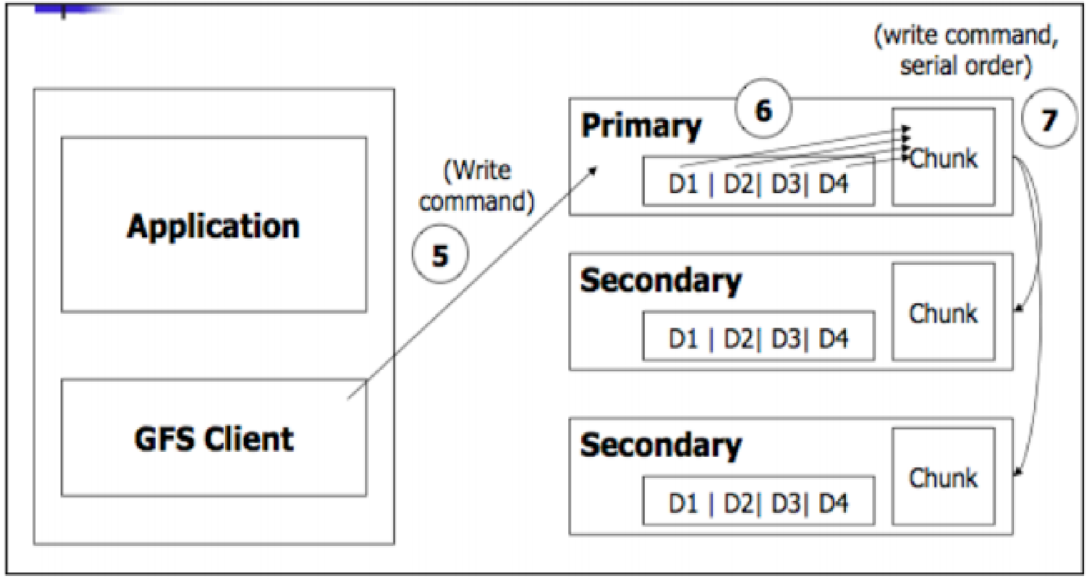
same as read

Write Algorithm



4. Client pushes write data to all locations. Data is stored in chunkserver's internal buffers

Write Algorithm

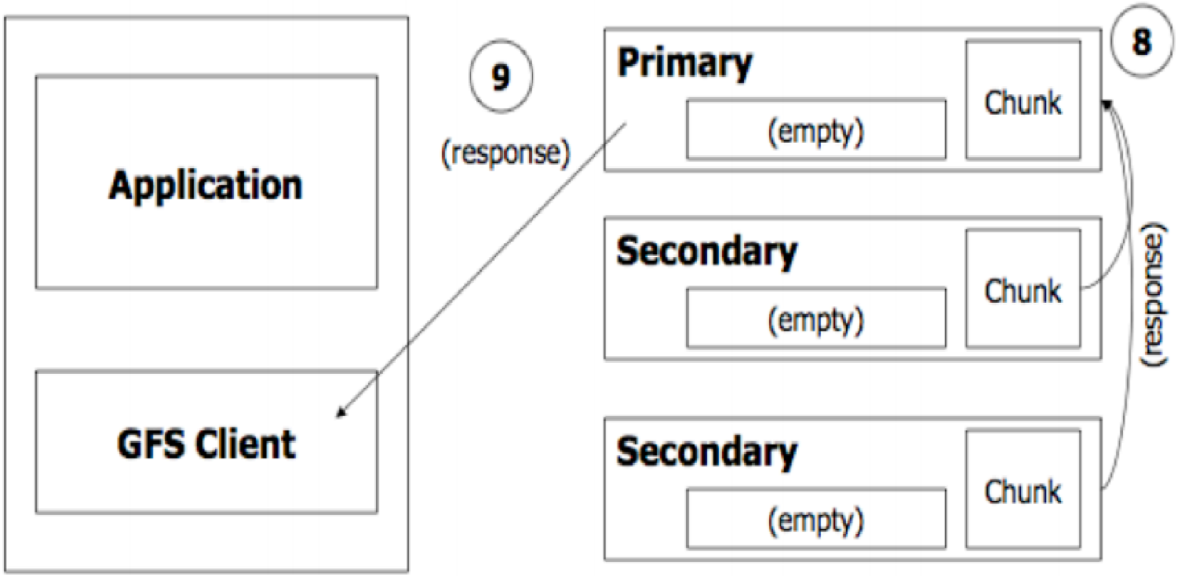


5. Client sends write command to primary

6. Primary determines serial order to the secondaries and tells them to perform the write

7. Primary sends the serial order to the secondaries and tells them to perform the write

Write Algorithm



8. Secondaries respond back to primary

9. Primary responds back to the client

Mutation

- Mutation changes metadata of chunk
  - Write
  - Append
- Each mutation is performed at all chunk replicas
- Lease mechanism
  - Master grants lease to one of the replicas
- Lease has 60 seconds timeout

Atomic Record Appends

- ❑ GFS appends it to the file atomically at least

once

  - – GFS picks the offset
  - – Works for concurrent writers
- ❑ Record append is heavily used by distributed applications
  - – eg., Google apps

Record Append Algorithm

- Client pushes write data to all locations
- Primary checks if record fits in specified chunk
- If the record does not fit:
  - Pads the chunk
  - Tells secondary to do the same
  - Informs client and has the client retry
- If record fits, then the primary:
  - Appends the record
  - Tells secondaries to do the same
  - Receives responses and responds to the client

Interesting thing happened at Google beyond GFS

❑ That’s Bigtable

- A distributed storage system which uses GFS to store log and data files

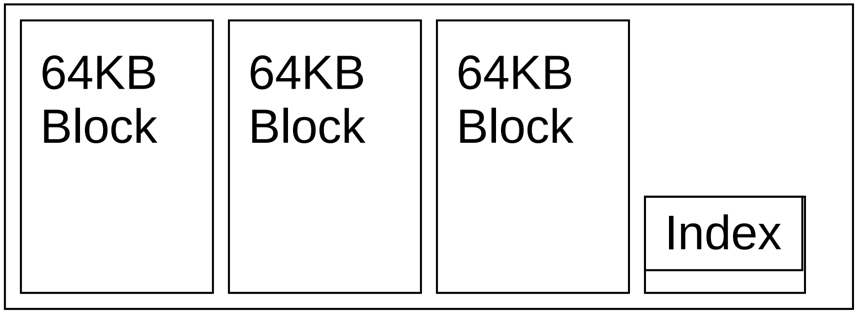
❑ More than 60 Google apps(like Google Earth, Orkut) uses Bigtable for storing data(distributed file though GFS)

❑ Designed to scale petabytes of data and thousands of machines

❑ It’s not a relational database, instead gives client a simple data model

Google SSTable file format

- ❑ Used internally to store Bigtable data
- ❑ Provides persistent ordered immutable map from keys to values
- ❑ Each SSTable contains a sequence of block and a block index



Amazon dynamoDB

Dynamo: Amazon’s Highly Available Key-Value Store

Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall and Werner Vogels

Amazon.com

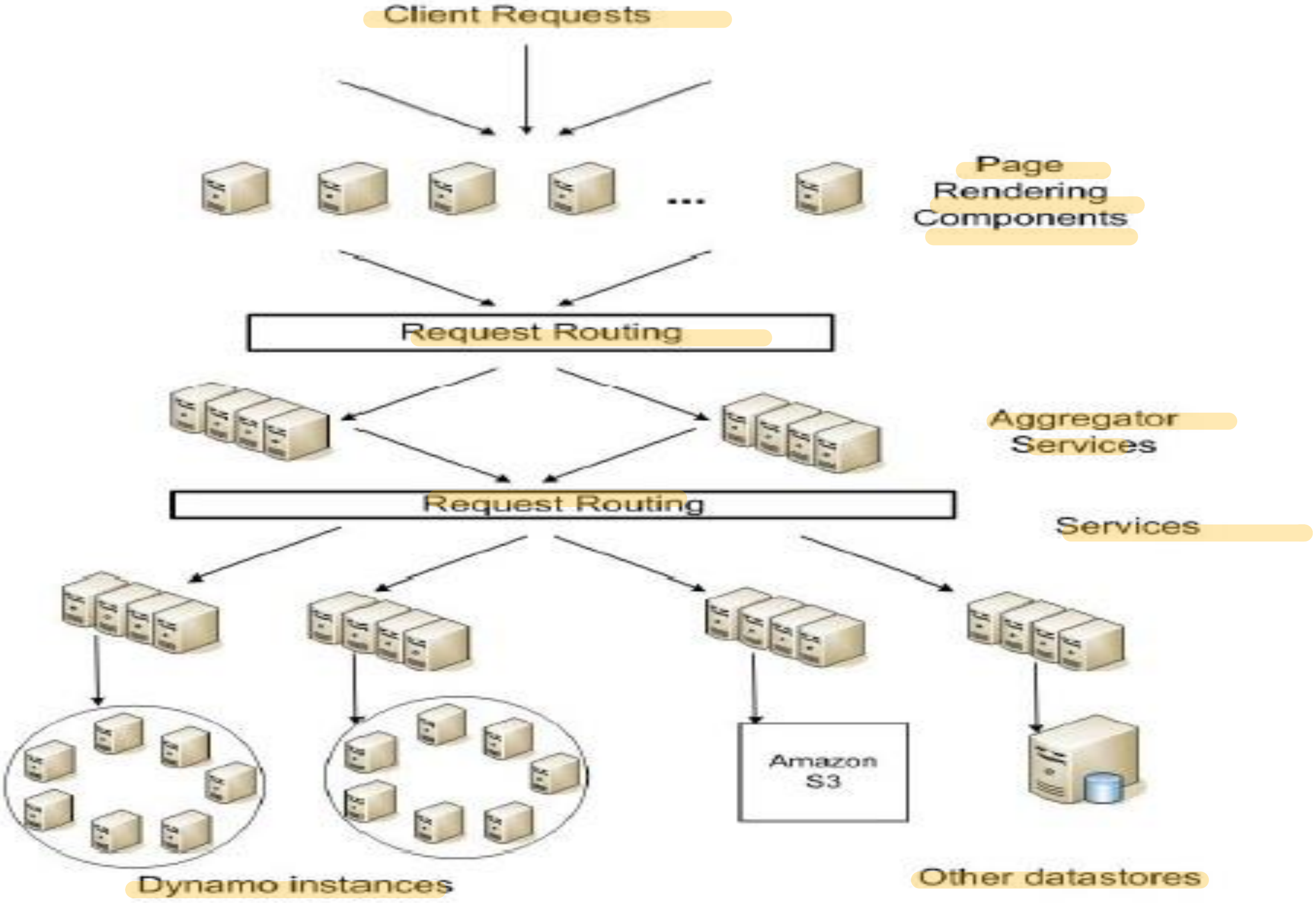
Amazon Dynamo Background

- ❑ Basically a data storage system.
- ❑ Service oriented architecture (SOA)
  - Decentralized
  - Loosely-coupled
- ❑ Hundreds of services up and running
- ❑ Needs storage scheme that is always available
  - Shopping cart service

System Assumptions and Requirements

- ❑ Query Model: simple read and write operations to a data item that is uniquely identified by a key.
- ❑ ACID Properties: Atomicity, Consistency, Isolation, Durability.
- ❑ Efficiency: latency requirements which are in general measured at the 99.9th percentile of the distribution.
- ❑ Other Assumptions: operation environment is assumed to be non-hostile and there are no security related requirements such as authentication and authorization.

Amazon Dynamo’s Architecture



Techniques used by Dynamo

- ❑ Consistent hashing along with Replication
  - Used for data partitioning
- ❑ Decentralized, quorum protocol
  - To maintain consistency during updates
- ❑ Gossip protocols
  - Memberships
  - Failure detection

Amazon Dynamo Highlights

- ❑ Dynamo is targeted mainly at applications that need an “always writeable” data store where no updates are rejected

- ❑ Applications do not require support for hierarchical namespaces (a norm in many file systems)

- ❑ Dynamo is built for latency sensitive applications that require at least 99.9% of read and write operations to be performed within a few hundred milliseconds

- ❑ Zero-hop DHT, where each node maintains enough routing information locally to route a request to the appropriate node directly.

L05/L06

✓ Hadoop Distributed File System

(HDFS)

[illegible]

Basic features

□ Highly fault-tolerant (any node fail, backup in other node is there)

□ High throughput

□ Suitable for applications with large data sets

□ Streaming access to file system data

□ Can be built out of commodity hardware

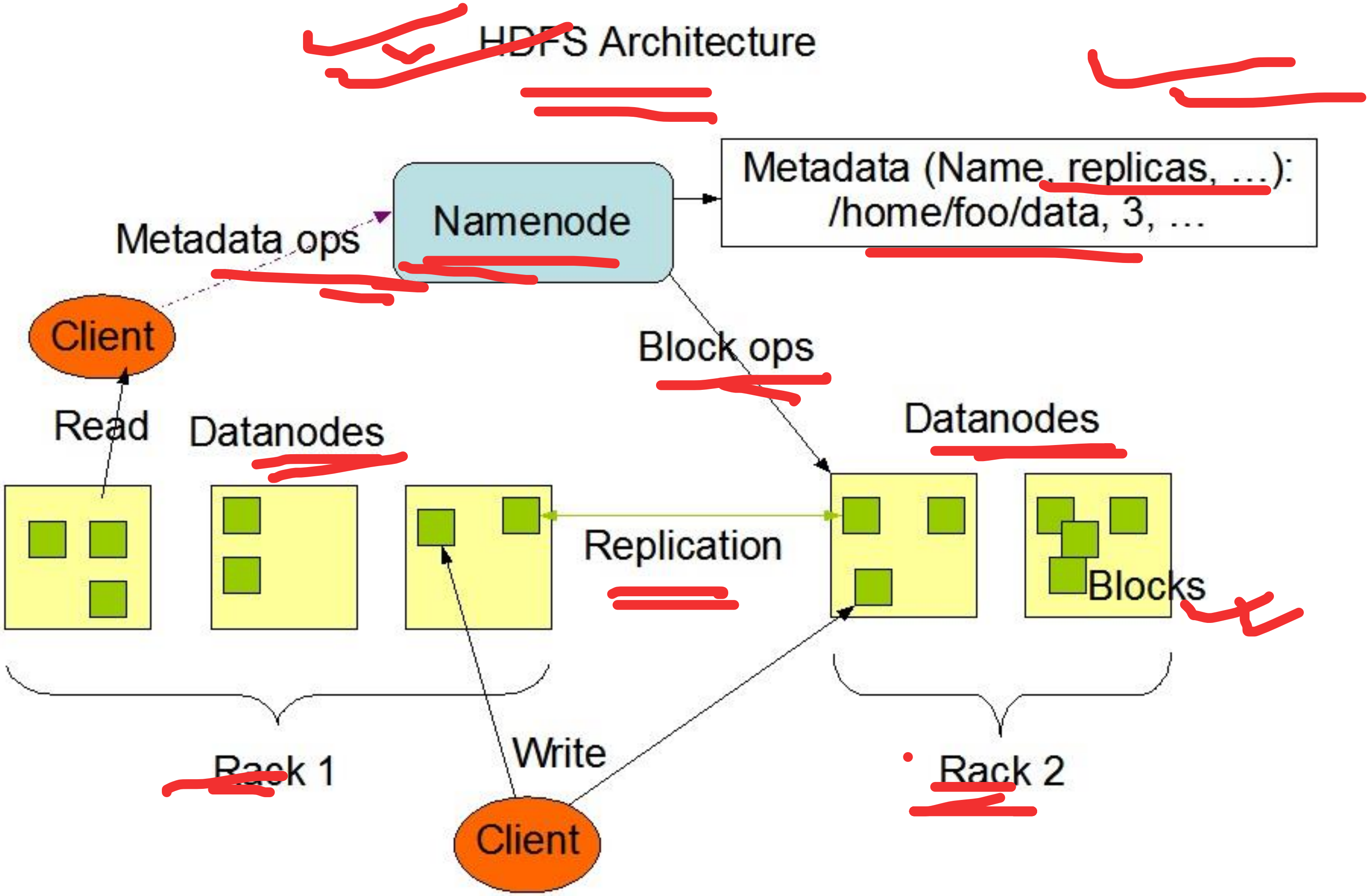
Fault tolerance

- □ Failure is the norm rather than exception
- □ A HDFS instance may consist of thousands of server machines, each storing part of the file system’s data.
- □ Since we have huge number of components and that each component has non-trivial probability of failure means that there is always some component that is non-functional.
- □ Detection of faults and quick, automatic recovery from them is a core architectural goal of HDFS.

Imp

HDFS Architecture

V.V.I.P



Data Characteristics

- Streaming data access
- Applications need streaming access to data
- Batch processing rather than interactive user access.
- Large data sets and files: gigabytes to terabytes size
- High aggregate data bandwidth
- Scale to hundreds of nodes in a cluster
- Tens of millions of files in a single instance
- Write-once-read-many: a file once created, written and closed need not be changed – this assumption simplifies coherency
- A map-reduce application or web-crawler application fits perfectly with this model.

v.v. Imp

Namenode and Datanodes

- Master/slave architecture
- HDFS cluster consists of a single Namenode, a master server that manages the file system namespace and regulates access to files by clients.
- There are a number of DataNodes usually one per node in a cluster.
- The DataNodes manage storage attached to the nodes that they run on.
- HDFS exposes a file system namespace and allows user data to be stored in files.
- A file is split into one or more blocks and set of blocks are stored in DataNodes.
- DataNodes: serves read, write requests, performs block creation, deletion, and replication upon instruction from Namenode.

V.V.Imp

File System Namespace

- Hierarchical file system with directories and files
- Create, remove, move, rename etc.
- Namenode maintains the file system
- Any meta information changes to the file system recorded by the Namenode.
- An application can specify the number of replicas of the file needed: replication factor of the file. This information is stored in the Namenode.

abstract

V.V.IP

Data Replication

- HDFS is designed to store very large files across machines in a large cluster.
- Each file is a sequence of blocks. V.V.I.P
- All blocks in the file except the last are of the same size.
- Blocks are replicated for fault tolerance.
- Block size and replicas are configurable per file.
- The Namenode receives a Heartbeat and a BlockReport from each DataNode in the cluster. V.V.I.P
- BlockReport contains all the blocks on a Datanode.

Conclusion

- ❑ It is rather impossible to meet both Availability and Consistency
- ❑ Distributed File Systems are heavily employed in organizational computing, and their performance has been the subject of much tuning
- ❑ Current state-of-the-art distributed file systems, provide good performance across both local and wide-area networks

Bibliography

1. Distributed Systems: Concepts and design Fifth Edition – George Coulouris, Jean Dollimore, Tim Kindberg, Gordon Blair

2. Distributed Systems: Principles and Paradigms Second Edition – Andrew S.Tanenbaum, Maarten Van Steen

3. An overview of the Andrew File System – John H Howard, CMU

4. The Google File System – Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung.

5 Dynamo: Amazon’s Highly Available Key-value Store - Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall and Werner Vogels

6 The Hadoop Distributed File System: Architecture and Design – Dhruba Borthakur