CLOUD TECHNOLOGIES AND ADVANCEMENTS

UNIT V

HADOOP

- A software library to write and run large user applications on vast data sets in business applications.
- A scalable, economical, efficient, and reliable tool for providing users with easy access of commercial clusters.
- Hadoop offers a software platform that was originally developed by a Yahoo! group.
- Users can easily scale Hadoop to store and process petabytes of data in the web space.

- Also, Hadoop is economical in that it comes with an open source version of MapReduce that minimizes overhead in task spawning and massive data communication.
- It is efficient, as it processes data with a high degree of parallelism across a large number of commodity nodes, and it is reliable in that it automatically keeps multiple data copies to facilitate redeployment of computing tasks upon unexpected system failures.
- Hadoop provides the runtime environment, and developers need to provide only the input data and specify the map and reduce functions that need to be executed.
- Hadoop is an integral part of the Yahoo! cloud infrastructure and supports several business processes of the company.
- Currently, Yahoo! manages the largest Hadoop cluster in the world, which is also available to academic institutions.

WHY USE HADOOP?

- Cheaper
  - Scales to Petabytes or more
- Faster
  - Parallel data processing
- Better
  - Suited for particular types of BigData problems

HADOOP LIBRARY FROM APACHE

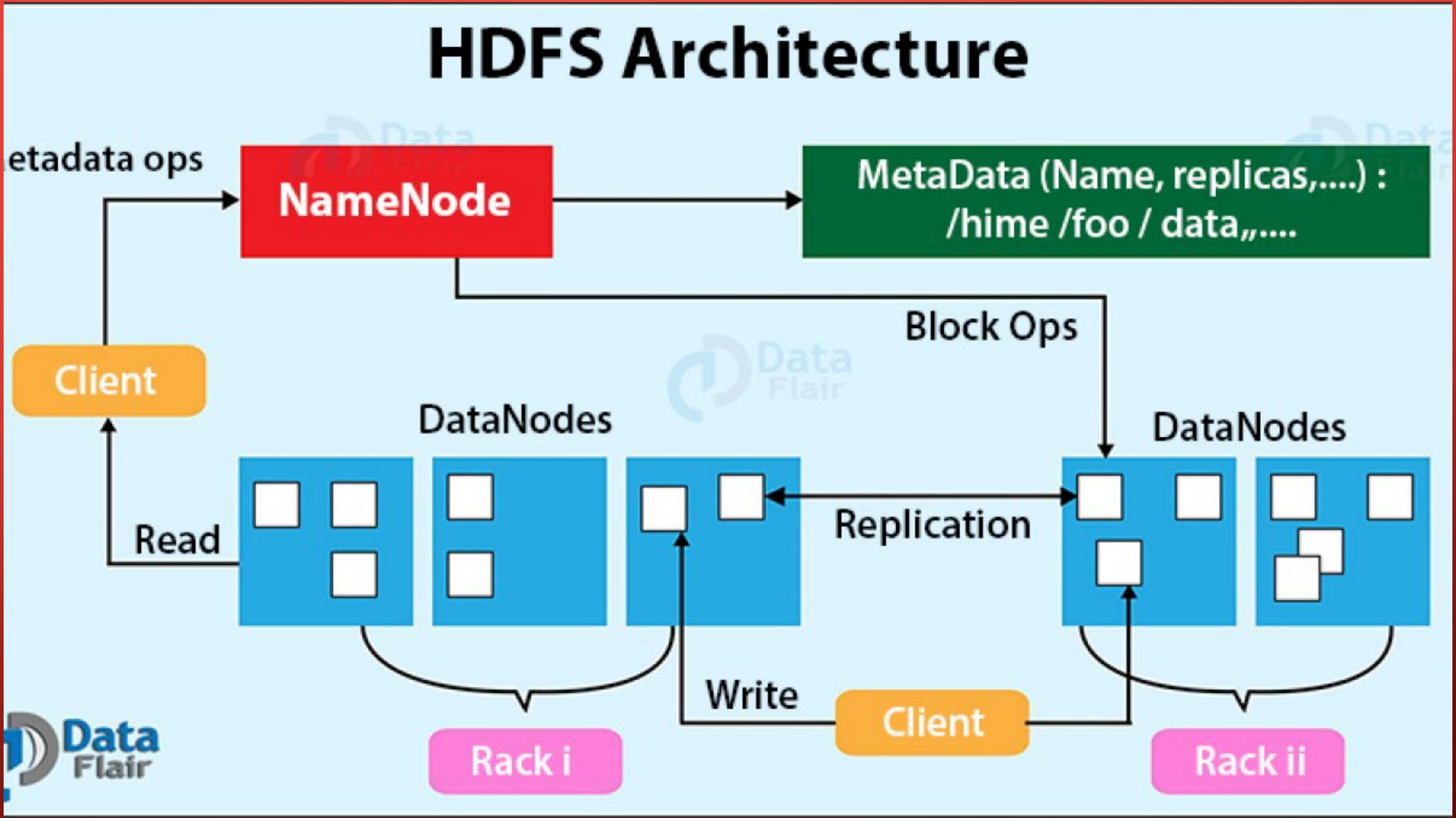
- Hadoop is an open source implementation of MapReduce coded and released in Java (rather than C) by Apache.
- The Hadoop implementation of MapReduce uses the Hadoop Distributed File System (HDFS) as its underlying layer rather than GFS.
- The Hadoop core is divided into two fundamental layers: the MapReduce engine and HDFS.
- The MapReduce engine is the computation engine running on top of HDFS as its data storage manager.
- The following two sections cover the details of these two fundamental

HDFS:

- HDFS is a distributed file system inspired by GFS that organizes files and stores their data on a distributed computing system.

HDFS Architecture:

- HDFS has a master/slave architecture containing a single NameNode as the master and a number of DataNodes as workers (slaves).
- To store a file in this architecture, HDFS splits the file into fixed-size blocks (e.g., 64 MB) and stores them on workers (DataNodes).
- The mapping of blocks to DataNodes is determined by the NameNode.
- The NameNode (master) also manages the file system's metadata and namespace.
- In such systems, the namespace is the area maintaining the metadata, and metadata refers to all the information stored by a file system that is needed for overall



HDFS Architecture
etadata ops
NameNode
MetaData (Name, replicas,....) : /hime /foo / data,,....
Block Ops
Client
DataNodes
DataNodes
Read
Replication
Write
Client
Rack i
Rack ii
Data Flair

- For example, NameNode in the metadata stores all information regarding the location of input splits/blocks in all DataNodes.
- Each DataNode, usually one per node in a cluster, manages the storage attached to the node.
- Each DataNode is responsible for storing and retrieving its file blocks.

HDFS Features:

- Distributed file systems have special requirements, such as performance, scalability, concurrency control, fault tolerance, and security requirements, to operate efficiently.
- However, because HDFS is not a general-purpose file system, as it only executes specific types of applications, it does not need all the requirements of a general distributed file system.
- For example, security has never been supported for HDFS systems.
- The following discussion highlights two important characteristics of HDFS to distinguish it from other generic distributed file systems.

HDFS Fault Tolerance:

- One of the main aspects of HDFS is its fault tolerance characteristic.
- Since Hadoop is designed to be deployed on low-cost hardware by default, a hardware failure in this system is considered to be common rather than an exception.
- Therefore, Hadoop considers the following issues to fulfill reliability requirements of the file system.
- Block replication To reliably store data in HDFS, file blocks are replicated in this system.
- In other words, HDFS stores a file as a set of blocks and each block is replicated and distributed across the whole cluster.
- The replication factor is set by the user and is three by default.

- Replica placement The placement of replicas is another factor to fulfill the desired fault tolerance in HDFS.
- Although storing replicas on different nodes (DataNodes) located in different racks across the whole cluster provides more reliability, it is sometimes ignored as the cost of communication between two nodes in different racks is relatively high in comparison with that of different nodes located in the same rack.
- Therefore, sometimes HDFS compromises its reliability to achieve lower communication costs.
- For example, for the default replication factor of three, HDFS stores one replica in the same node the original data is stored, one replica on a different node but in the same rack, and one replica on a different node in a different rack to provide three copies of the data.

- Heartbeat and Blockreport messages Heartbeats and Blockreports are periodic messages sent to the NameNode by each DataNode in a cluster.
- Receipt of a Heartbeat implies that the DataNode is functioning properly, while each Blockreport contains a list of all blocks on a DataNode.
- The NameNode receives such messages because it is the sole decision maker of all replicas in the system.

HDFS High-Throughput Access to Large Data Sets (Files):

- Because HDFS is primarily designed for batch processing rather than interactive processing, data access throughput in HDFS is more important than latency.
- Also, because applications run on HDFS typically have large data sets, individual files are broken into large blocks (e.g., 64 MB) to allow HDFS to

- This provides two advantages: The list of blocks per file will shrink as the size of individual blocks increases, and by keeping large amounts of data sequentially within a block, HDFS provides fast streaming reads of data.

HDFS Operation:

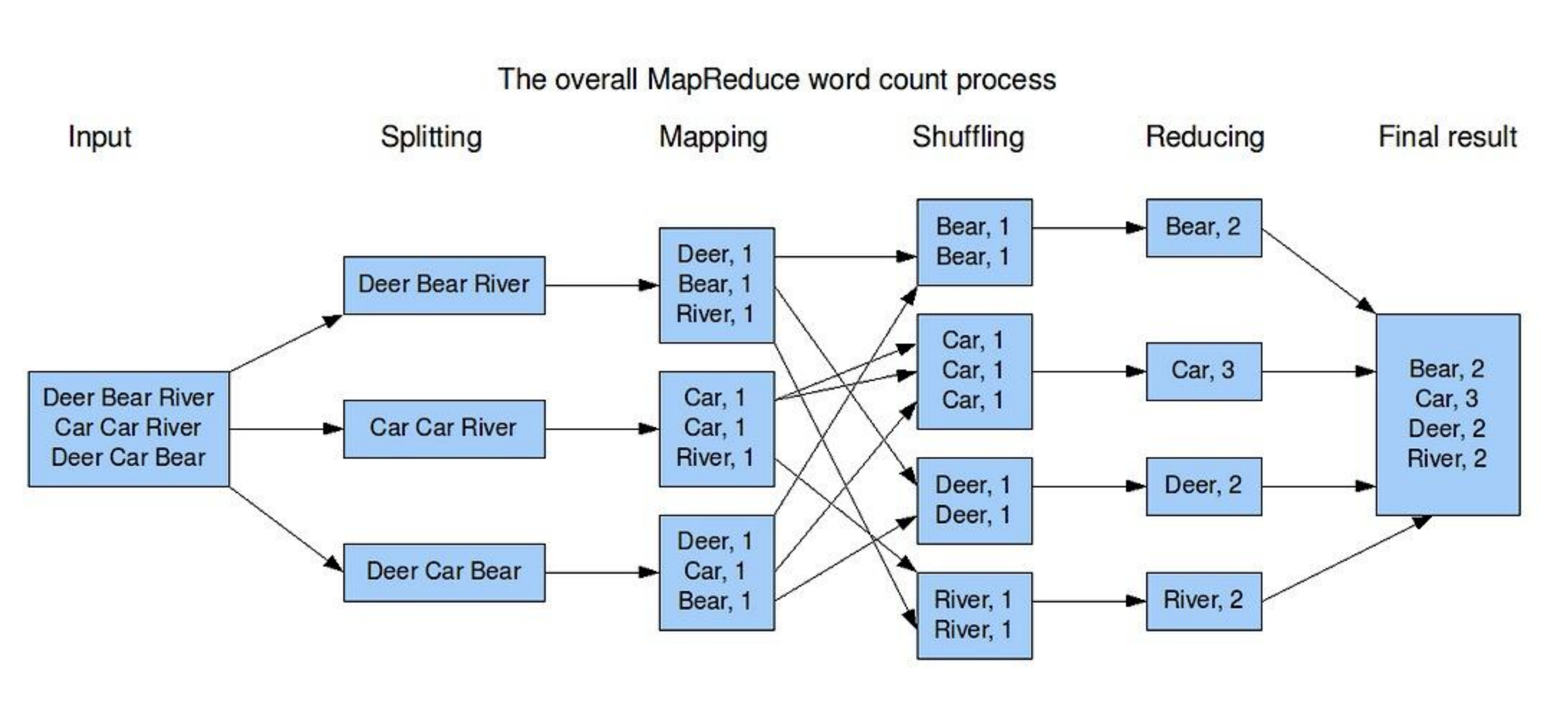
- The control flow of HDFS operations such as write and read can properly highlight roles of the NameNode and DataNodes in the managing operations.
- In this section, the control flow of the main operations of HDFS on files is further described to manifest the interaction between the user, the NameNode, and the DataNodes in such systems.
- **Reading a file** To read a file in HDFS, a user sends an "open" request to the NameNode to get the location of file blocks.
- For each file block, the NameNode returns the address of a set of DataNodes

- The number of addresses depends on the number of block replicas.
- Upon receiving such information, the user calls the read function to connect to the closest DataNode containing the first block of the file.
- After the first block is streamed from the respective DataNode to the user, the established connection is terminated and the same process is repeated for all blocks of the requested file until the whole file is streamed to the user.
- Writing to a file To write a file in HDFS, a user sends a “create” request to the NameNode to create a new file in the file system namespace.
- If the file does not exist, the NameNode notifies the user and allows him to start writing data to the file by calling the write function.
- The first block of the file is written to an internal queue termed the data queue while a data streamer monitors its writing into a DataNode.

- Since each file block needs to be replicated by a predefined factor, the data streamer first sends a request to the NameNode to get a list of suitable DataNodes to store replicas of the first block.
- The steamer then stores the block in the first allocated DataNode.
- Afterwards, the block is forwarded to the second DataNode by the first DataNode.
- The process continues until all allocated DataNodes receive a replica of the first block from the previous DataNode.
- Once this replication process is finalized, the same process starts for the second block and continues until all blocks of the file are stored and replicated on the file system.

MAP REDUCE

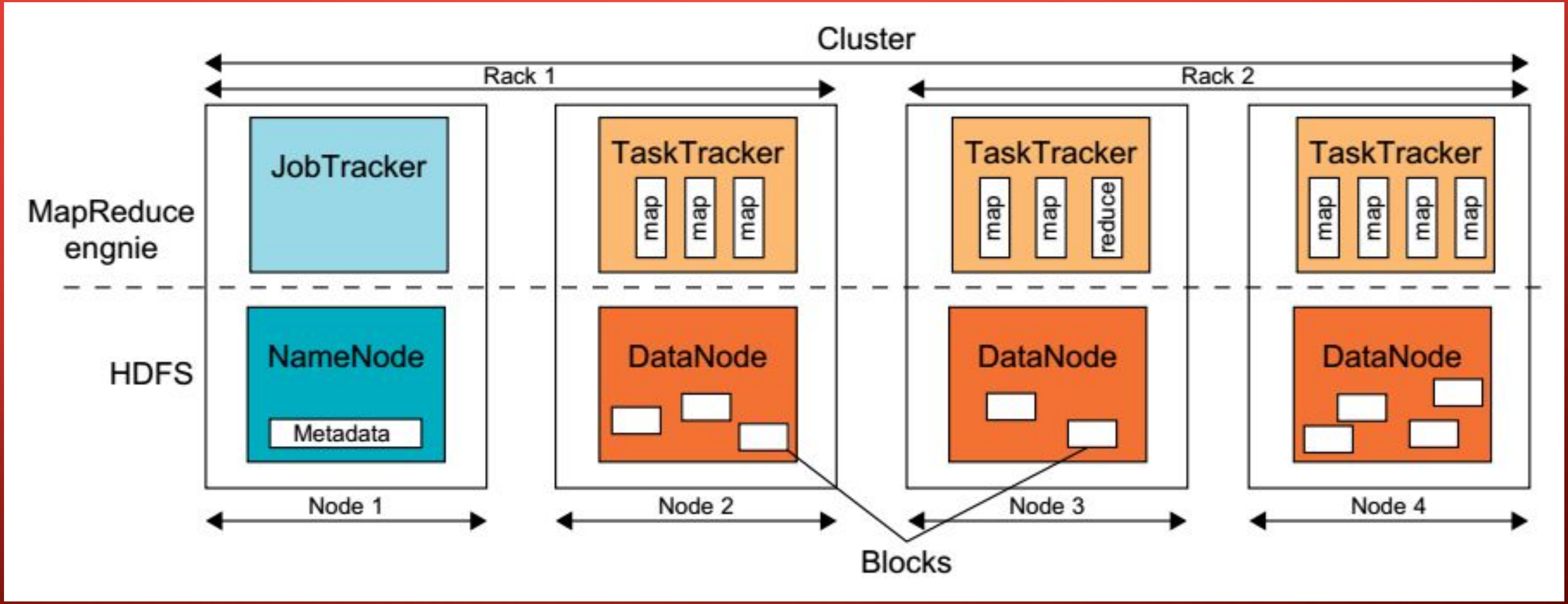
- This is a web programming model for scalable data processing on large clusters over large data sets.
- The model is applied mainly in web-scale search and cloud computing applications.
- The user specifies a Map function to generate a set of intermediate key/value pairs.
- Then the user applies a Reduce function to merge all intermediate values with the same intermediate key.
- MapReduce is highly scalable to explore high degrees of parallelism at different job levels.
- A typical MapReduce computation process can handle terabytes of data on tens of thousands or more client machines.
- Hundreds of MapReduce programs can be executed simultaneously; in fact, thousands of MapReduce jobs are executed on Google's clusters every day.


The overall MapReduce word count process
Input
Splitting
Mapping
Shuffling
Reducing
Final result
Deer Bear River
Car Car River
Deer Car Bear
Deer Bear River
Car Car River
Deer Car Bear
Deer, 1
Bear, 1
River, 1
Car, 1
Car, 1
River, 1
Deer, 1
Car, 1
Bear, 1
Bear, 1
Bear, 1
Car, 1
Car, 1
Car, 1
Deer, 1
Deer, 1
River, 1
River, 1
Bear, 2
Car, 3
Deer, 2
River, 2
Bear, 2
Car, 3
Deer, 2
River, 2

ARCHITECTURE OF MAPREDUCE IN HADOOP

- The topmost layer of Hadoop is the MapReduce engine that manages the data flow and control flow of MapReduce jobs over distributed computing systems.
- The following figure shows the MapReduce engine architecture cooperating with HDFS.
- Similar to HDFS, the MapReduce engine also has a master/slave architecture consisting of a single JobTracker as the master and a number of TaskTrackers as the slaves (workers).

HDFS and MapReduce architecture in Hadoop where boxes with different shadings refer to different functional nodes applied to different blocks of data.

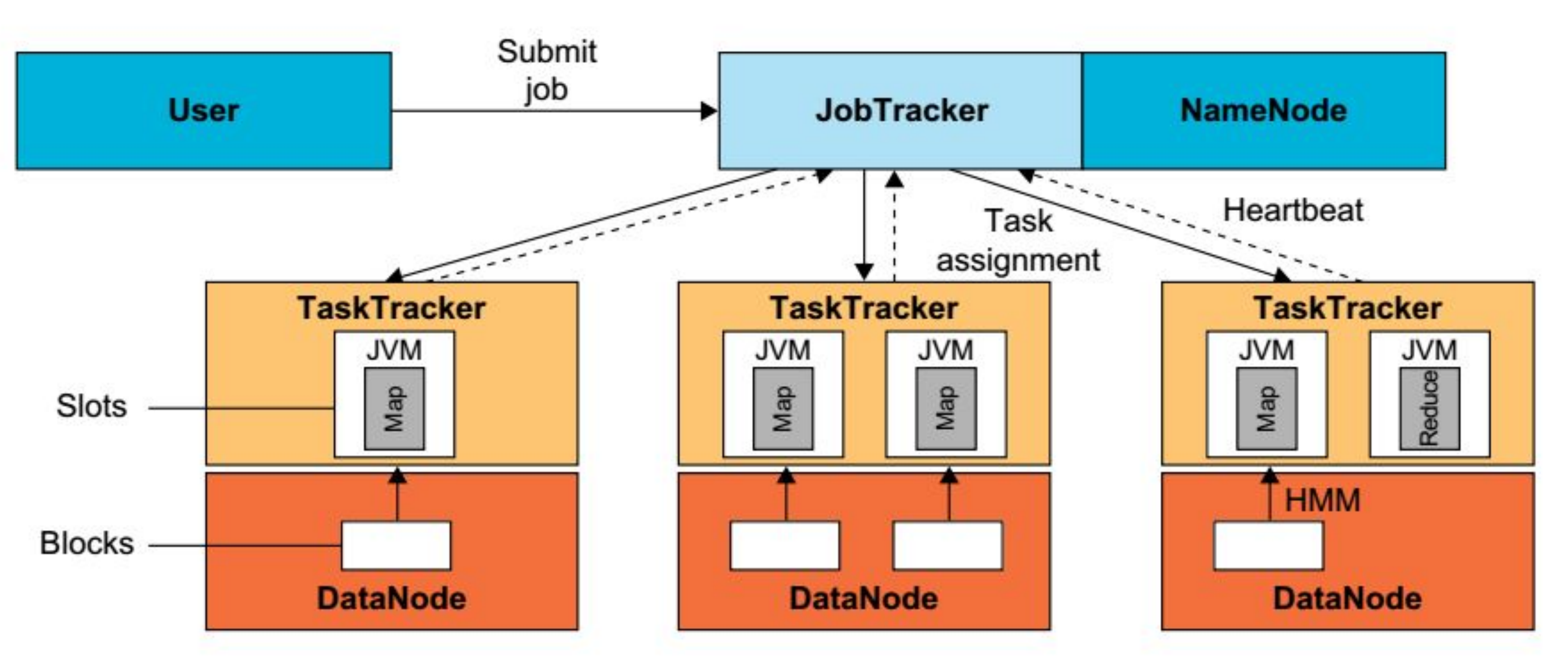


- The JobTracker manages the MapReduce job over a cluster and is responsible for monitoring jobs and assigning tasks to TaskTrackers.
- The TaskTracker manages the execution of the map and/or reduce tasks on a single computation node in the cluster.
- Each TaskTracker node has a number of simultaneous execution slots, each executing either a map or a reduce task.
- Slots are defined as the number of simultaneous threads supported by CPUs of the TaskTracker node.
- For example, a TaskTracker node with N CPUs, each supporting M threads, has M * N simultaneous execution slots.
- It is worth noting that each data block is processed by one map task running on a single slot.
- Therefore, there is a one-to-one correspondence between map tasks in a

RUNNING A JOB IN HADOOP

- Three components contribute in running a job in this system: a user node, a JobTracker, and several TaskTrackers.
- The data flow starts by calling the runJob(conf) function inside a user program running on the user node, in which conf is an object containing some tuning parameters for the MapReduce framework and HDFS.
- The runJob(conf) function and conf are comparable to the MapReduce(Spec, &Results) function and Spec in the first implementation of MapReduce by Google.
- The following figure depicts the data flow of running a MapReduce job in Hadoop.

Data flow in running a MapReduce job at various task trackers using the Hadoop library



- Job Submission Each job is submitted from a user node to the JobTracker node that might be situated in a different node within the cluster through the following procedure:
    - A user node asks for a new job ID from the JobTracker and computes input file splits.
    - The user node copies some resources, such as the job's JAR file, configuration file, and computed input splits, to the JobTracker's file system.
    - The user node submits the job to the JobTracker by calling the submitJob() function.
- Task assignment The JobTracker creates one map task for each computed input split by the user node and assigns the map tasks to the execution slots of the TaskTrackers.
- The JobTracker considers the localization of the data when assigning the map tasks to the TaskTrackers.
- The JobTracker also creates reduce tasks and assigns them to the TaskTrackers.

- Task execution The control flow to execute a task (either map or reduce) starts inside the TaskTracker by copying the job JAR file to its file system.
- Instructions inside the job JAR file are executed after launching a Java Virtual Machine (JVM) to run its map or reduce task.
- Task running check A task running check is performed by receiving periodic heartbeat messages to the JobTracker from the TaskTrackers.
- Each heartbeat notifies the JobTracker that the sending TaskTracker is alive, and whether the sending TaskTracker is ready to run a new task.

LIMITATIONS OF MAPREDUCE

- Batch processing, not interactive
- Designed for a specific problem domain
- MapReduce programming paradigm not commonly understood (functional)
- Lack of trained support professionals
- API / security model are moving targets

VIRTUAL BOX

What is a Virtual Machine?

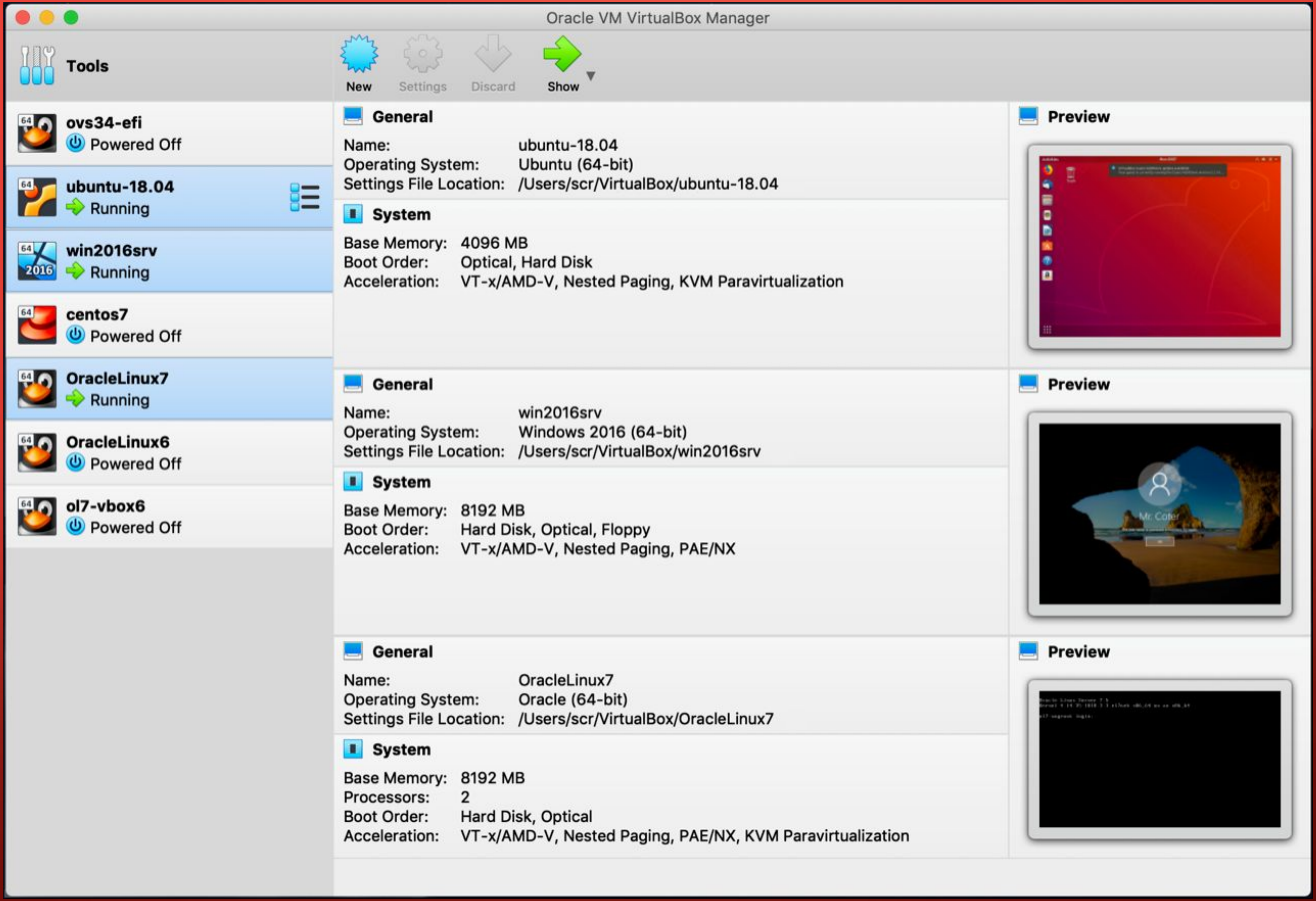
- A software implementation of a machine (i.e. a computer) that executes programs like a physical machine.
- A Virtual Machine can run a completely isolated guest operating system installation within a normal host operating system.

What would you use a Virtual Machine for?

- Testing out a Operating System without erasing or dual booting your existing OS.
- Running software that is not compatible with your current OS.
- Creating a secure environment for web browsing.

VirtualBox

- VirtualBox is virtualization software for the x86 architecture (Intel).
- Cross platform support
- –Will run on Windows, Linux and Macintosh
- Integration features with Linux, Windows, Mac Guest OSs
- Support for Virtualization Technology built in most modern CPUs
- Support for virtualization of base I/O ports, optical drives, removable storage and network controllers.
- Support VPN and Remote Desktop functions as well as Java based remote management through a web interface.
- Supports translation of OpenGL and Direct3D extensions to the guest OS


Oracle VM VirtualBox Manager
Tools
New
Settings
Discard
Show
ovs34-efi
Powered Off
ubuntu-18.04
Running
win2016srv
Running
centos7
Powered Off
OracleLinux7
Running
OracleLinux6
Powered Off
ol7-vbox6
Powered Off
General
Name: ubuntu-18.04
Operating System: Ubuntu (64-bit)
Settings File Location: /Users/scr/VirtualBox/ubuntu-18.04
System
Base Memory: 4096 MB
Boot Order: Optical, Hard Disk
Acceleration: VT-x/AMD-V, Nested Paging, KVM Paravirtualization
Preview
General
Name: win2016srv
Operating System: Windows 2016 (64-bit)
Settings File Location: /Users/scr/VirtualBox/win2016srv
System
Base Memory: 8192 MB
Boot Order: Hard Disk, Optical, Floppy
Acceleration: VT-x/AMD-V, Nested Paging, PAE/NX
Preview
General
Name: OracleLinux7
Operating System: Oracle (64-bit)
Settings File Location: /Users/scr/VirtualBox/OracleLinux7
System
Base Memory: 8192 MB
Processors: 2
Boot Order: Hard Disk, Optical
Acceleration: VT-x/AMD-V, Nested Paging, PAE/NX, KVM Paravirtualization
Preview

GAE – GOOGLE APP ENGINE

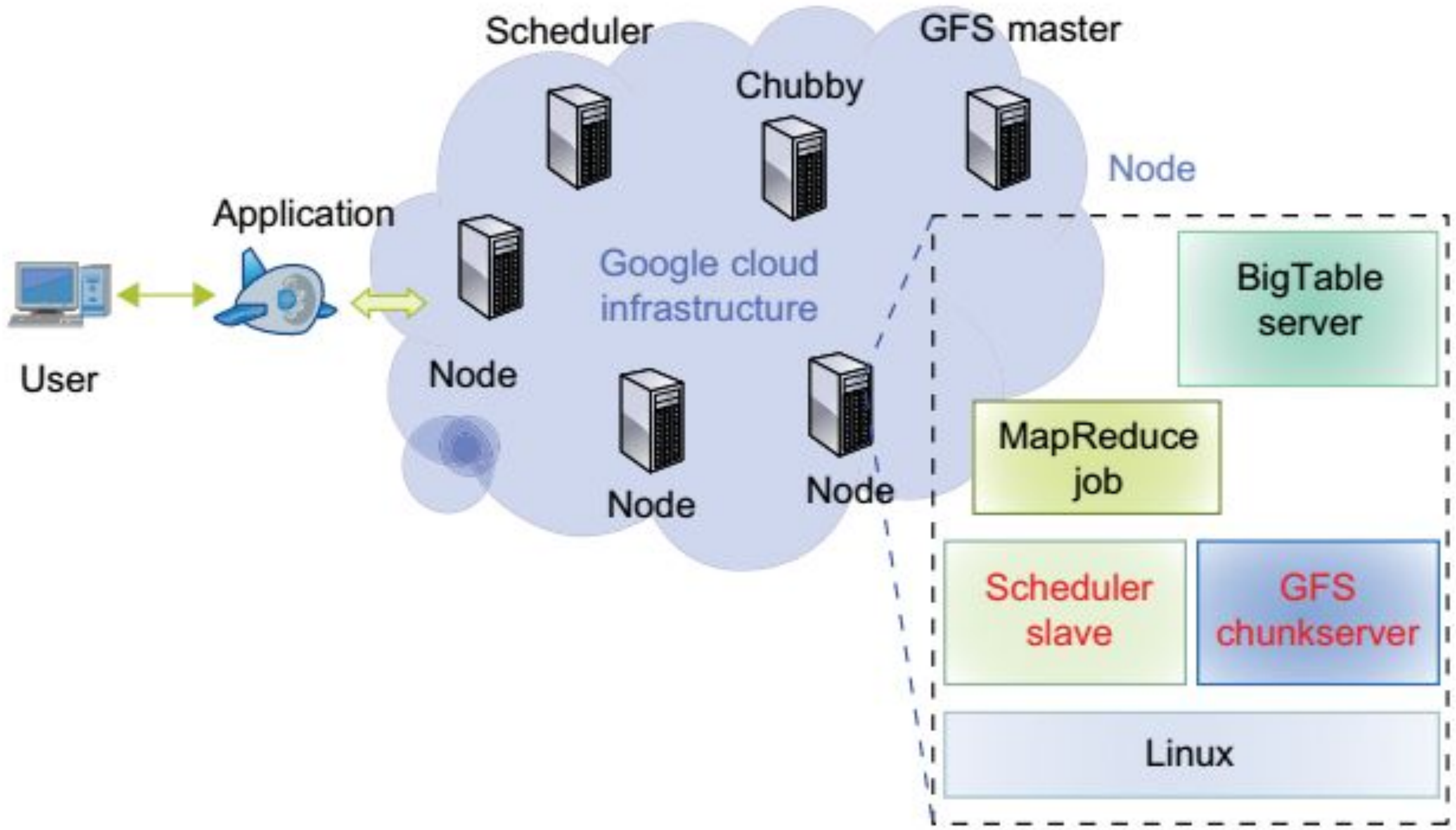
- Google has the world's largest search engine facilities.
- The company has extensive experience in massive data processing that has led to new insights into data-center design and novel programming models that scale to incredible sizes.
- The Google platform is based on its search engine expertise, but as discussed earlier with MapReduce, this infrastructure is applicable to many other areas.
- Google has hundreds of data centers and has installed more than 4,60,000 servers worldwide.
- For example, 200 Google data centers are used at one time for a number of cloud applications.
- Data items are stored in text, images, and video and are replicated to tolerate faults or failures.
- Here we discuss Google's App Engine (GAE) which offers a PaaS platform

GOOGLE CLOUD INFRASTRUCTURE

- Google has pioneered cloud development by leveraging the large number of data centers it operates.
- For example, Google pioneered cloud services in Gmail, Google Docs, and Google Earth, among other applications.
- These applications can support a large number of users simultaneously with HA.
- Notable technology achievements include the Google File System (GFS), MapReduce, BigTable, and Chubby.
- In 2008, Google announced the GAE web application platform which is becoming a common platform for many small cloud service providers.
- This platform specializes in supporting scalable (elastic) web applications.

GAE ARCHITECTURE

- The following fig shows the major building blocks of the Google cloud platform which has been used to deliver the cloud services highlighted earlier.
- GFS is used for storing large amounts of data.
- MapReduce is for use in application program development.
- Chubby is used for distributed application lock services.
- BigTable offers a storage service for accessing structured data.
- Users can interact with Google applications via the web interface provided by each application.
- Third-party application providers can use GAE to build cloud applications for providing services.
- The applications all run in data centers under tight management by Google engineers.


Scheduler
Chubby
GFS master
Node
Application
Google cloud
infrastructure
User
Node
Node
Node
BigTable
server
MapReduce
job
Scheduler
slave
GFS
chunkserver
Linux

- Google is one of the larger cloud application providers, although its fundamental service program is private and outside people cannot use the Google infrastructure to build their own service.
- The building blocks of Google's cloud computing application include the Google File System for storing large amounts of data, the MapReduce programming framework for application developers, Chubby for distributed application lock services, and BigTable as a storage service for accessing structural or semi structural data.
- With these building blocks, Google has built many cloud applications.
- A typical cluster configuration can run the Google File System, MapReduce jobs, and BigTable servers for structure data.
- Extra services such as Chubby for distributed locks can also run in the clusters.

- GAE runs the user program on Google's infrastructure.
- As it is a platform running third-party programs, application developers now do not need to worry about the maintenance of servers.
- GAE can be thought of as the combination of several software components.
- The frontend is an application framework which is similar to other web application frameworks such as ASP, J2EE, and JSP.
- At the time of this writing, GAE supports Python and Java programming environments.
- The applications can run similar to web application containers.
- The frontend can be used as the dynamic web serving infrastructure which can provide the full support of common technologies.

FUNCTIONAL MODULES OF GAE

- The GAE platform comprises the following five major components.
- The GAE is not an infrastructure platform, but rather an application development platform for users.
- We describe the component functionalities separately.

a) The datastore offers object-oriented, distributed, structured data storage services based on BigTable techniques. The datastore secures data management operations.

b) The application runtime environment offers a platform for scalable web programming and execution. It supports two development languages: Python and Java.

c) The software development kit (SDK) is used for local application development. The SDK allows users to execute test runs of local applications and upload application code.

d) The administration console is used for easy management of user application development cycles, instead of for physical resource management.

e) The GAE web service infrastructure provides special interfaces to guarantee flexible use and management of storage and network resources by GAE.

- Google offers essentially free GAE services to all Gmail account owners.
- You can register for a GAE account or use your Gmail account name to sign up for the service.
- The service is free within a quota.
- If you exceed the quota, the page instructs you on how to pay for the service.
- Then you download the SDK and read the Python or Java guide to get started.
- Note that GAE only accepts Python, Ruby, and Java programming languages.
- The platform does not provide any IaaS services, unlike Amazon, which offers Iaas and PaaS.
- This model allows the user to deploy user-built applications on top of the cloud infrastructure that are built using the programming languages and software tools supported by the provider (e.g., Java, Python). Azure does this similarly for .NET.
- The user does not manage the underlying cloud infrastructure.
- The cloud provider facilitates support of application development, testing, and

GAE APPLICATIONS

- Well-known GAE applications include the Google Search Engine, Google Docs, Google Earth, and Gmail.
- These applications can support large numbers of users simultaneously.
- Users can interact with Google applications via the web interface provided by each application.
- Third-party application providers can use GAE to build cloud applications for providing services.
- The applications are all run in the Google data centers.
- Inside each data center, there might be thousands of server nodes to form different clusters.

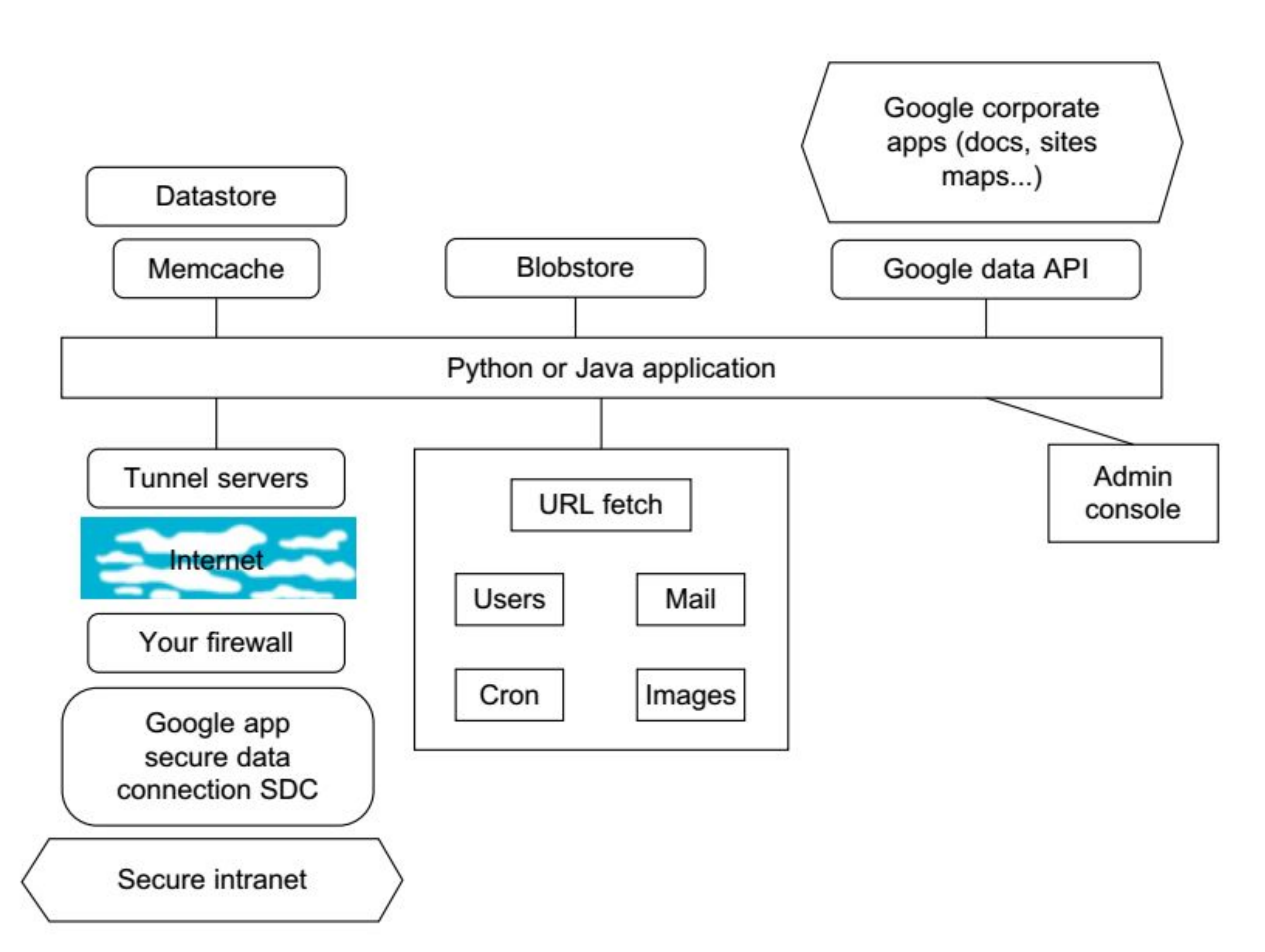
GAE APPLICATIONS

- GAE supports many web applications.
- One is a storage service to store application-specific data in the Google infrastructure.
- The data can be persistently stored in the backend storage server while still providing the facility for queries, sorting, and even transactions similar to traditional database systems.
- GAE also provides Google-specific services, such as the Gmail account service (which is the login service, that is, applications can use the Gmail account directly).
- This can eliminate the tedious work of building customized user management components in web applications.
- Thus, web applications built on top of GAE can use the APIs authenticating users and sending e-mail using Google accounts.

PROGRAMMING ENVIRONMENT FOR GOOGLE APP ENGINE

- The following figure summarizes some key features of GAE programming model for two supported languages: Java and Python.
- A client environment that includes an Eclipse plug-in for Java allows you to debug your GAE on your local machine.
- Also, the GWT Google Web Toolkit is available for Java web application developers.
- Developers can use this, or any other language using a JVMbased interpreter or compiler, such as JavaScript or Ruby.
- Python is often used with frameworks such as Django and CherryPy, but Google also supplies a built in webapp Python environment.

Programming environment for Google AppEngine



- There are several powerful constructs for storing and accessing data.
- The data store is a NOSQL data management system for entities that can be, at most, 1 MB in size and are labeled by a set of schema-less properties.
- Queries can retrieve entities of a given kind filtered and sorted by the values of the properties.
- Java offers Java Data Object (JDO) and Java Persistence API (JPA) interfaces implemented by the open source Data Nucleus Access platform, while Python has a SQL-like query language called GQL.
- The data store is strongly consistent and uses optimistic concurrency control.
- An update of an entity occurs in a transaction that is retried a fixed number of times if other processes are trying to update the same entity simultaneously.
- Your application can execute multiple data store operations in a single transaction which either all succeed or all fail together.

- The data store implements transactions across its distributed network using "entity groups."
- A transaction manipulates entities within a single group.
- Entities of the same group are stored together for efficient execution of transactions.
- Your GAE application can assign entities to groups when the entities are created.
- The performance of the data store can be enhanced by in-memory caching using the memcache, which can also be used independently of the data store.
- Recently, Google added the blobstore which is suitable for large files as its size limit is 2 GB.
- There are several mechanisms for incorporating external resources.
- The Google SDC Secure Data Connection can tunnel through the Internet and

- The URL Fetch operation provides the ability for applications to fetch resources and communicate with other hosts over the Internet using HTTP and HTTPS requests.
- There is a specialized mail mechanism to send e-mail from your GAE application.
- Applications can access resources on the Internet, such as web services or other data, using GAE’s URL fetch service.
- The URL fetch service retrieves web resources using the same highspeed Google infrastructure that retrieves web pages for many other Google products.
- There are dozens of Google “corporate” facilities including maps, sites, groups, calendar, docs, and YouTube, among others.
- These support the Google Data API which can be used inside GAE.

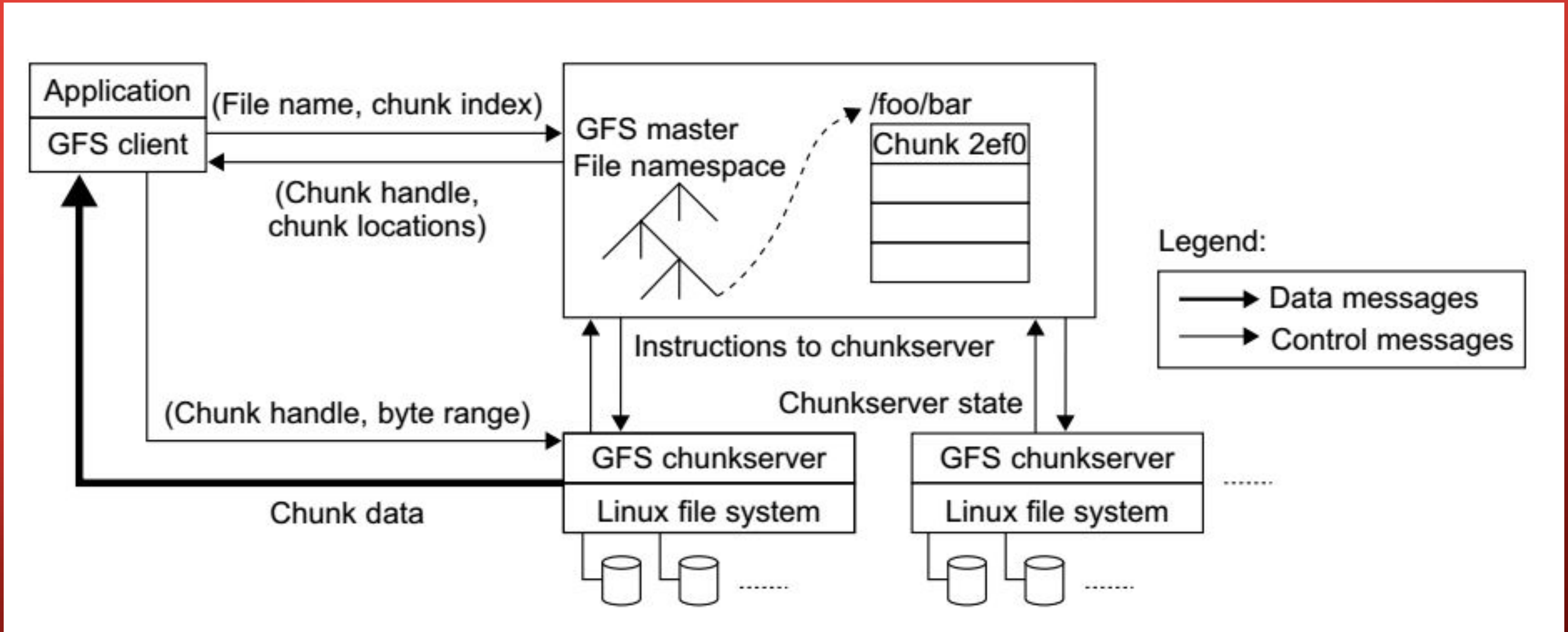
- An application can use Google Accounts for user authentication.
- Google Accounts handles user account creation and sign-in, and a user that already has a Google account (such as a Gmail account) can use that account with your app.
- GAE provides the ability to manipulate image data using a dedicated Images service which can resize, rotate, flip, crop, and enhance images.
- An application can perform tasks outside of responding to web requests.
- Your application can perform these tasks on a schedule that you configure, such as on a daily or hourly basis using "cron jobs," handled by the Cron service.
- Alternatively, the application can perform tasks added to a queue by the application itself, such as a background task created while handling a request.
- A GAE application is configured to consume resources up to certain limits or quotas.
- With quotas, GAE ensures that your application won't exceed your budget, and that other applications running on GAE won't impact the performance of your app.

GOOGLE FILE SYSTEM (GFS)

- GFS was built primarily as the fundamental storage service for Google's search engine.
- As the size of the web data that was crawled and saved was quite substantial, Google needed a distributed file system to redundantly store massive amounts of data on cheap and unreliable computers.
- None of the traditional distributed file systems can provide such functions and hold such large amounts of data.
- In addition, GFS was designed for Google applications, and Google applications were built for GFS.

- In traditional file system design, such a philosophy is not attractive, as there should be a clear interface between applications and the file system, such as a POSIX interface.
- There are several assumptions concerning GFS.
- One is related to the characteristic of the cloud computing hardware infrastructure (i.e., the high component failure rate).
- As servers are composed of inexpensive commodity components, it is the norm rather than the exception that concurrent failures will occur all the time.
- Another concerns the file size in GFS.
- GFS typically will hold a large number of huge files, each 100 MB or larger, with files that are multiple GB in size quite common.
- Thus, Google has chosen its file data block size to be 64 MB instead of the 4 KB in typical traditional file systems.
- Reliability is achieved by using replications (i.e., each chunk or data block of a file

Architecture of Google File System (GFS)

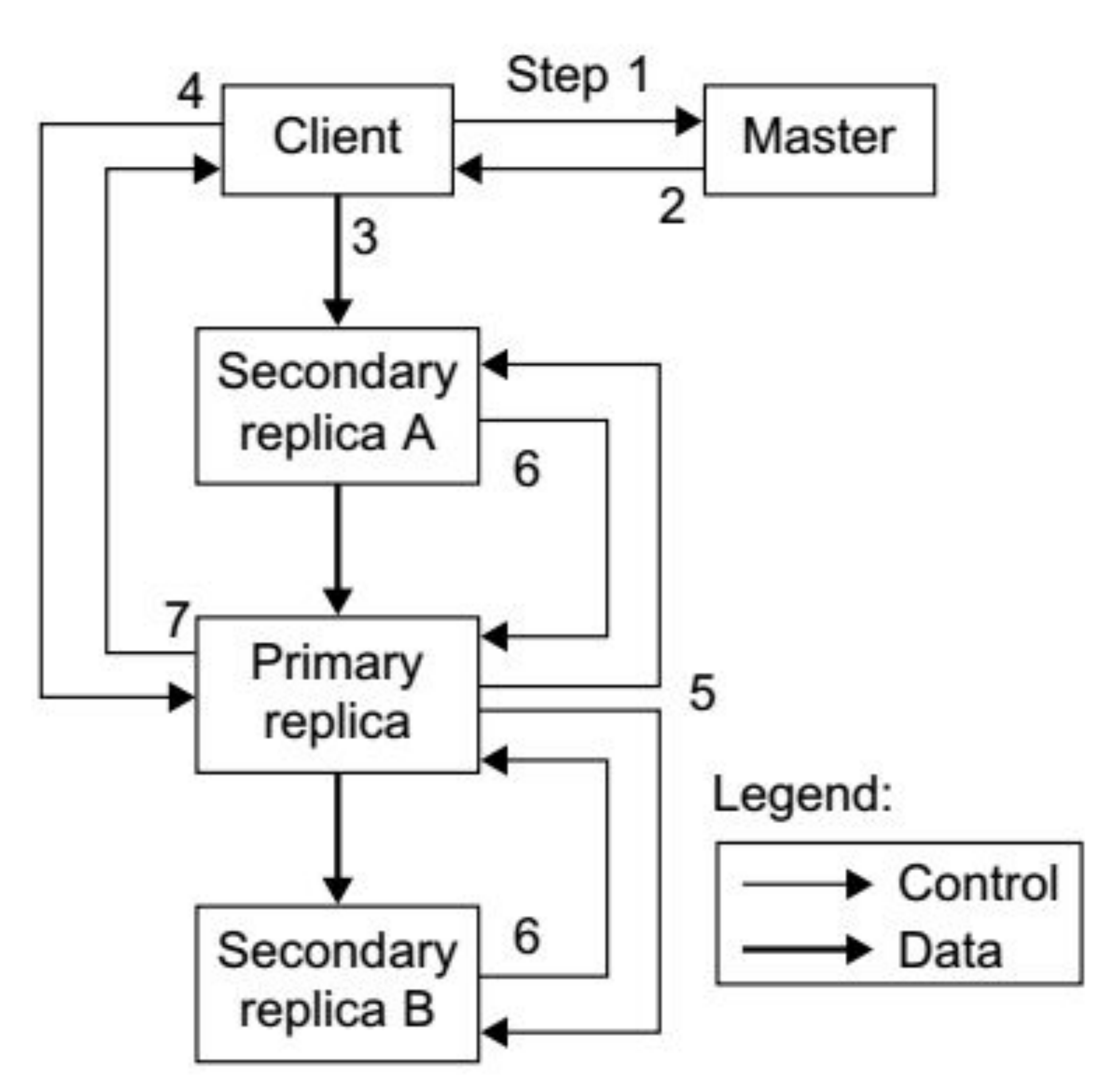


- In GFS architecture, it is quite obvious that there is a single master in the whole cluster.
- Other nodes act as the chunk servers for storing data, while the single master stores the metadata.
- The file system namespace and locking facilities are managed by the master.
- The master periodically communicates with the chunk servers to collect management information as well as give instructions to the chunk servers to do work such as load balancing or fail recovery.
- The master has enough information to keep the whole cluster in a healthy state.
- With a single master, many complicated distributed algorithms can be avoided and the design of the system can be simplified.

- A single master coordinates access as well as keeps the metadata.
- This decision simplified the design and management of the whole cluster.
- Developers do not need to consider many difficult issues in distributed systems, such as distributed consensus.
- There is no data cache in GFS as large streaming reads and writes represent neither time nor space locality.
- GFS provides a similar, but not identical, POSIX file system accessing interface.
- The distinct difference is that the application can even see the physical location of file blocks.
- Such a scheme can improve the upper-layer applications.
- The customized API can simplify the problem and focus on Google

DATA MUTATION SEQUENCE IN GFS

- The following fig shows the data mutation (write, append operations) in GFS.
- Data blocks must be created for all replicas.
- The goal is to minimize involvement of the master.
- The mutation takes the following steps:


Step 1
Client
Master
2
3
4
Secondary
replica A
6
7
Primary
replica
5
Secondary
replica B
6
Legend:
Control
Data

1. The client asks the master which chunk server holds the current lease for the chunk and the locations of the other replicas. If no one has a lease, the master grants one to a replica it chooses (not shown).

2. The master replies with the identity of the primary and the locations of the other (secondary) replicas. The client caches this data for future mutations. It needs to contact the master again only when the primary becomes unreachable or replies that it no longer holds a lease.

3. The client pushes the data to all the replicas. A client can do so in any order. Each chunk server will store the data in an internal LRU buffer cache until the data is used or aged out. By decoupling the data flow from the control flow, we can improve performance by scheduling the expensive data flow based on the network topology regardless of which chunk server is the primary.

4. Once all the replicas have acknowledged receiving the data, the client sends a write request to the primary. The request identifies the data pushed earlier to all the replicas. The primary assigns consecutive serial numbers to all the mutations it receives, possibly from multiple clients, which provides the necessary serialization. It applies the mutation to its own local state in serial order.

5. The primary forwards the write request to all secondary replicas. Each secondary replica applies mutations in the same serial number order assigned by the primary.

6. The secondaries all reply to the primary indicating that they have completed the operation.

7. The primary replies to the client. Any errors encountered at any replicas are reported to the client. In case of errors, the write corrects at the primary and an arbitrary subset of the secondary replicas. The client request is considered to have failed, and the modified region is left in an inconsistent state. Our client code handles such errors by retrying the failed mutation. It will make a few attempts at steps 3 through 7 before falling back to a retry from the beginning of the write.

- GFS was designed for high fault tolerance and adopted some methods to achieve this goal. Master and chunk servers can be restarted in a few seconds, and with such a fast recovery capability, the window of time in which the data is unavailable can be greatly reduced.

BIGTABLE, GOOGLE'S NOSQL SYSTEM

- BigTable - Innovative Google technology
- BigTable was designed to provide a service for storing and retrieving structured and semi-structured data.
- BigTable applications include storage of web pages, per-user data, and geographic locations.
- Here we use web pages to represent URLs and their associated data, such as contents, crawled metadata, links, anchors, and page rank values.
- Per-user data has information for a specific user and includes such data as user preference settings, recent queries/search results, and the user's e-mails.

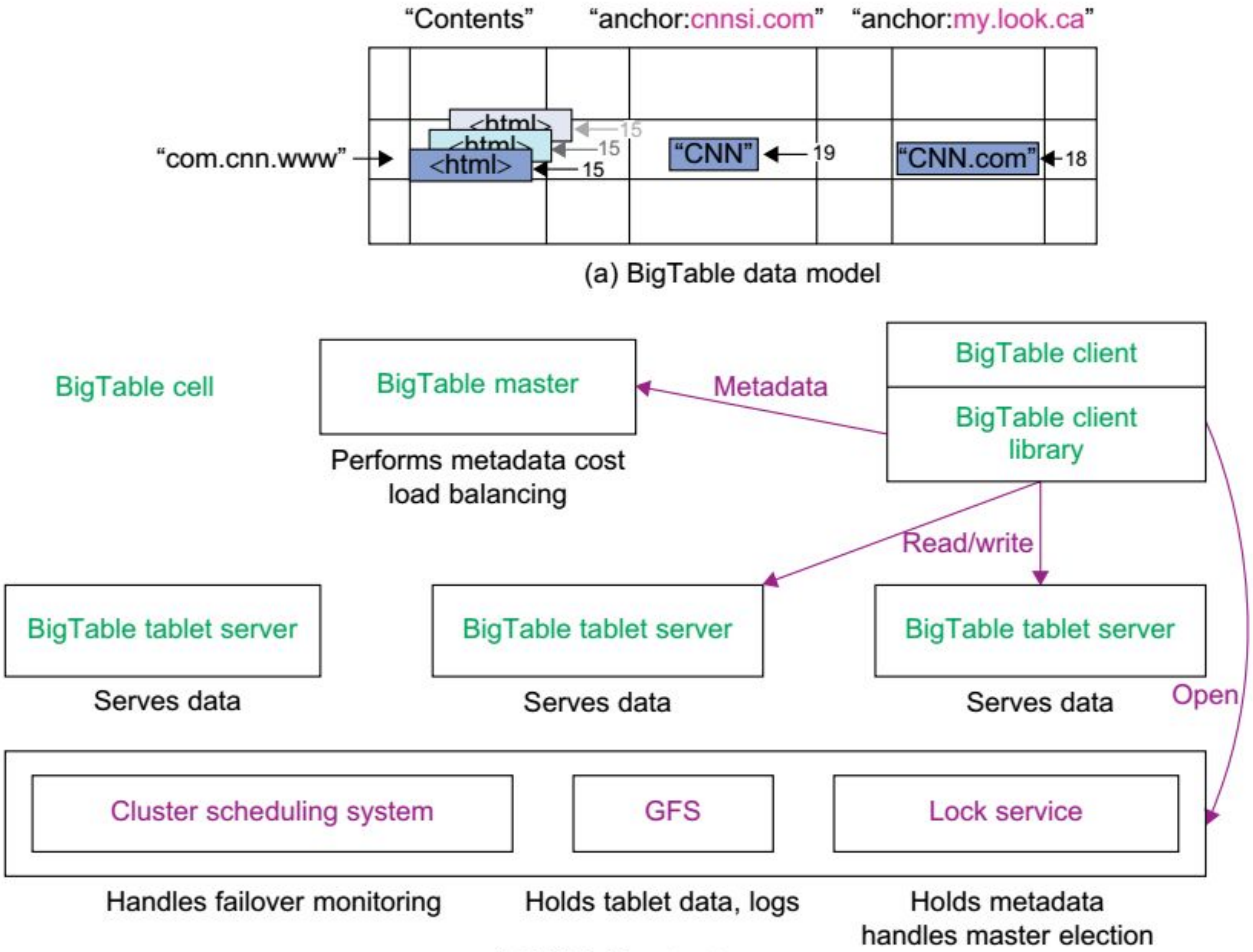
- Geographic locations are used in Google's well-known Google Earth software.
- Geographic locations include physical entities (shops, restaurants, etc.), roads, satellite image data, and user annotations.
- The scale of such data is incredibly large.
- There will be billions of URLs, and each URL can have many versions, with an average page size of about 20 KB per version.
- The user scale is also huge.
- There are hundreds of millions of users and there will be thousands of queries per second.
- The same scale occurs in the geographic data, which might consume more than 100 TB of disk space.

- It is not possible to solve such a large scale of structured or semistructured data using a commercial database system.
- This is one reason to rebuild the data management system; the resultant system can be applied across many projects for a low incremental cost.
- The other motivation for rebuilding the data management system is performance.
- Low-level storage optimizations help increase performance significantly, which is much harder to do when running on top of a traditional database layer.
- The design and implementation of the BigTable system has the following goals.
- The applications want asynchronous processes to be continuously updating different pieces of data and want access to the most current data at all times.
- The database needs to support very high read/write rates and the scale might be

- Also, the database needs to support efficient scans over all or interesting subsets of data, as well as efficient joins of large one-to-one and one-to-many data sets.
- The application may need to examine data changes over time (e.g., contents of a web page over multiple crawls).
- Thus, BigTable can be viewed as a distributed multilevel map.
- It provides a fault-tolerant and persistent database as in a storage service.
- The BigTable system is scalable, which means the system has thousands of servers, terabytes of in-memory data, petabytes of disk-based data, millions of reads/writes per second, and efficient scans.
- Also, BigTable is a self-managing system (i.e., servers can be added/removed dynamically and it features automatic load balancing).
- Design/initial implementation of BigTable began at the beginning of 2004.
- BigTable is used in many projects, including Google Search, Orkut, and Google Maps/Google Earth, among others.

- The BigTable system is built on top of an existing Google cloud infrastructure.
- BigTable uses the following building blocks:

1. GFS: stores persistent state
2. Scheduler: schedules jobs involved in BigTable serving
3. Lock service: master election, location bootstrapping
4. MapReduce: often used to read/write BigTable data

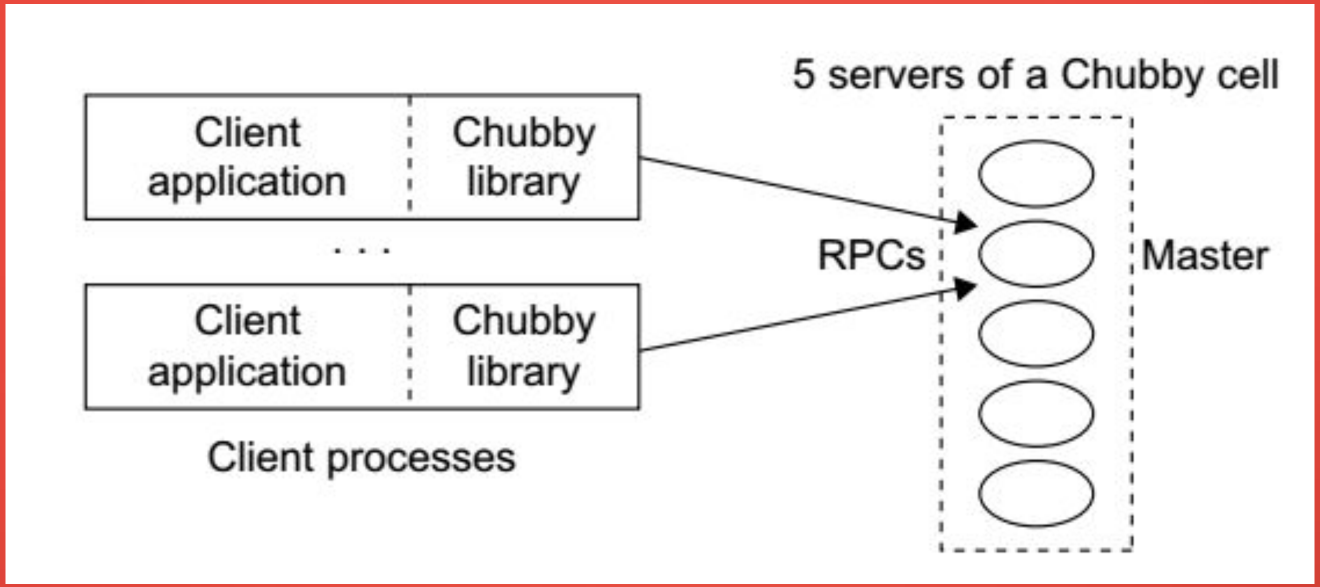


(a) BigTable data model

(b) BigTable structure

CHUBBY, GOOGLE'S DISTRIBUTED LOCK SERVICE

- Chubby is intended to provide a coarse-grained locking service.
- It can store small files inside Chubby storage which provides a simple namespace as a file system tree.
- The files stored in Chubby are quite small compared to the huge files in GFS.
- Based on the Paxos agreement protocol, the Chubby system can be quite reliable despite the failure of any member node.
- The following Figure shows the overall architecture of the Chubby system.



- Each Chubby cell has five servers inside.
- Each server in the cell has the same file system namespace.
- Clients use the Chubby library to talk to the servers in the cell.
- Client applications can perform various file operations on any server in the Chubby cell.
- Servers run the Paxos protocol to make the whole file system reliable and consistent.
- Chubby has become Google's primary internal name service.

OPEN STACK

- OpenStack was been introduced by Rackspace and NASA in July 2010.
- The project is building an open source community spanning technologists, developers, researchers, and industry to share resources and technologies with the goal of creating a massively scalable and secure cloud infrastructure.
- Currently, OpenStack focuses on the development of two aspects of cloud computing to address compute and storage aspects with the OpenStack Compute and OpenStack Storage solutions.

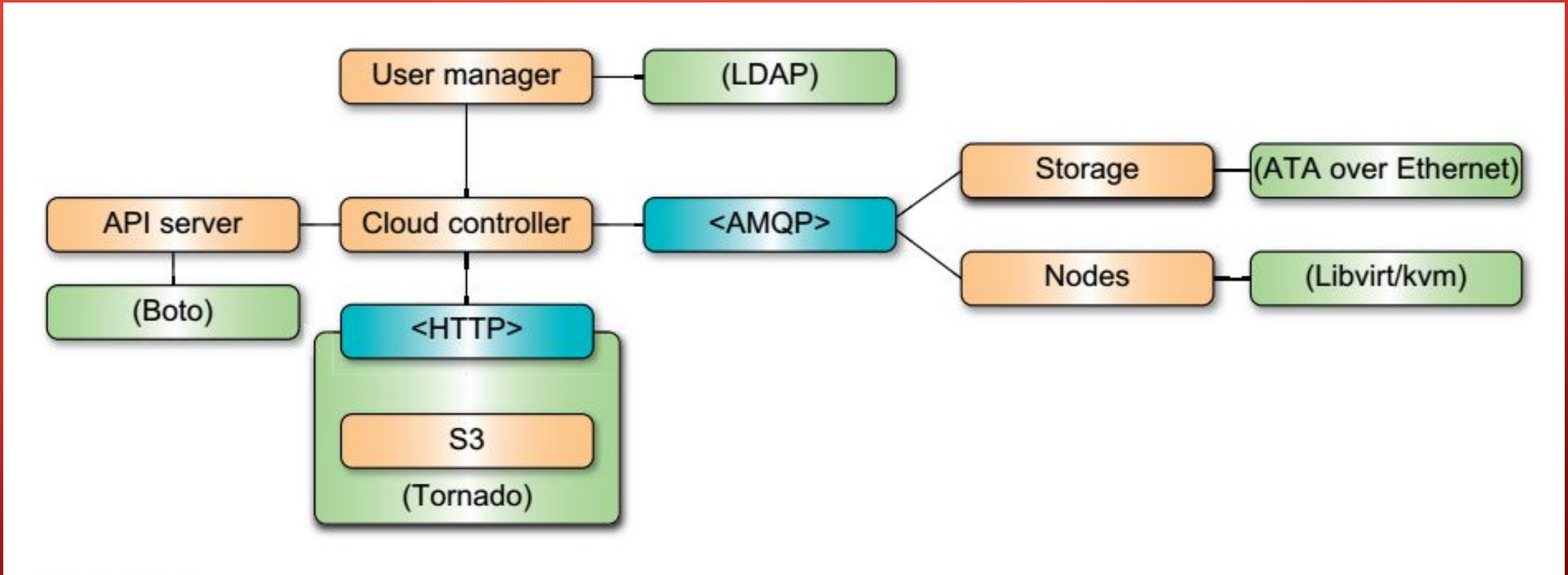
- “OpenStack Compute is the internal fabric of the cloud creating and managing large groups of virtual private servers” and “OpenStack Object Storage is software for creating redundant, scalable object storage using clusters of commodity servers to store terabytes or even petabytes of data.”
- Recently, an image repository was prototyped.
- The image repository contains an image registration and discovery service and an image delivery service.
- Together they deliver images to the compute service while obtaining them from the storage service.
- This development gives an indication that the project is striving to integrate more services into its portfolio.

OPENSTACK COMPUTE

- As part of its computing support efforts, OpenStack is developing a cloud computing fabric controller, a component of an IaaS system, known as Nova.
- The architecture for Nova is built on the concepts of shared-nothing and messaging-based information exchange.
- Hence, most communication in Nova is facilitated by message queues.
- To prevent blocking components while waiting for a response from others, deferred objects are introduced.
- Such objects include callbacks that get triggered when a response is received.

- To achieve the shared-nothing paradigm, the overall system state is kept in a distributed data system.
- State updates are made consistent through atomic transactions.
- Nova it implemented in Python while utilizing a number of externally supported libraries and components.
- This includes boto, an Amazon API provided in Python, and Tornado, a fast HTTP server used to implement the S3 capabilities in OpenStack.
- The following figure shows the main architecture of Open Stack Compute.
- In this architecture, the API Server receives HTTP requests from boto, converts the commands to and from the API format, and forwards the requests to the cloud controller.
- The cloud controller maintains the global state of the system, ensures authorization while interacting with the User Manager via Lightweight Directory Access Protocol (LDAP) , interacts with the S3 service, and manages nodes, as well as storage

OpenStack Nova system architecture



OPENSTACK STORAGE

- The OpenStack storage solution is built around a number of interacting components and concepts, including a proxy server, a ring, an object server, a container server, an account server, replication, updaters, and auditors.
- The role of the proxy server is to enable lookups to the accounts, containers, or objects in OpenStack storage rings and route the requests.
- Thus, any object is streamed to or from an object server directly through the proxy server to or from the user.
- A ring represents a mapping between the names of entities stored on disk and their physical locations.

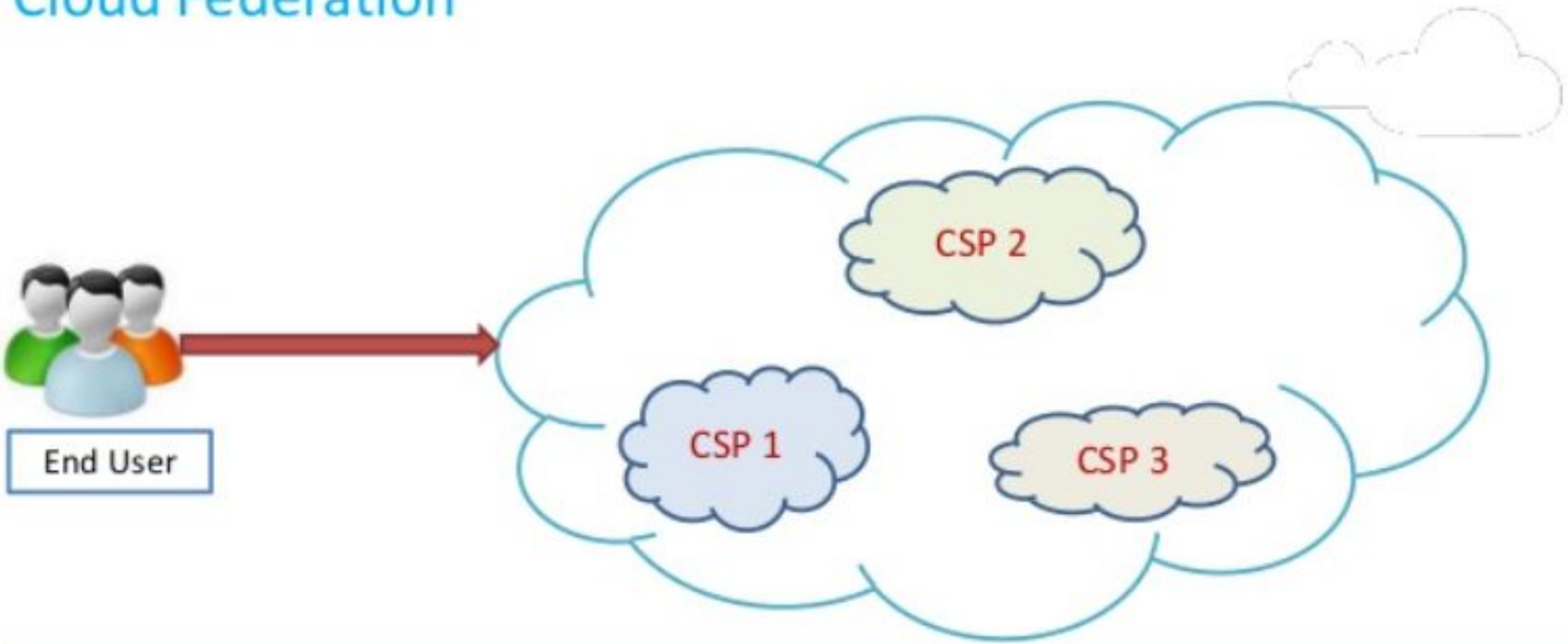
- Separate rings for accounts, containers, and objects exist.
- A ring includes the concept of using zones, devices, partitions, and replicas.
- Hence, it allows the system to deal with failures, and isolation of zones representing a drive, a server, a cabinet, a switch, or even a data center.
- Weights can be used to balance the distribution of partitions on drives across the cluster, allowing users to support heterogeneous storage resources.
- According to the documentation, “the Object Server is a very simple blob storage server that can store, retrieve and delete objects stored on local devices.”
- Objects are stored as binary files with metadata stored in the file’s extended attributes.
- This requires that the underlying file system is built around object servers, which is often not the case for standard Linux installations.

FEDERATION IN THE CLOUD

- **Definition:** Cloud federation is the practice of interconnecting the cloud computing environments of two or more service providers for the purpose of load balancing traffic and accommodating spikes in demand.
- Cloud federation requires one provider to wholesale or rent computing resources to another cloud provider.
- Those resources become a temporary or permanent extension of the buyer's cloud computing environment, depending on the specific federation agreement between providers.

- Cloud federation offers two substantial benefits to cloud providers.
- First, it allows providers to earn revenue from computing resources that would otherwise be idle or underutilized.
- Second, cloud federation enables cloud providers to expand their geographic footprints and accommodate sudden spikes in demand without having to build new points-of-presence (POPs).
- Service providers strive to make all aspects of cloud federation from cloud provisioning to billing support systems (BSS) and customer support transparent to customers.
- When federating cloud services with a partner, cloud providers will also establish extensions of their customer-facing service-level agreements (SLAs) into their partner provider's data centers.

Cloud Federation

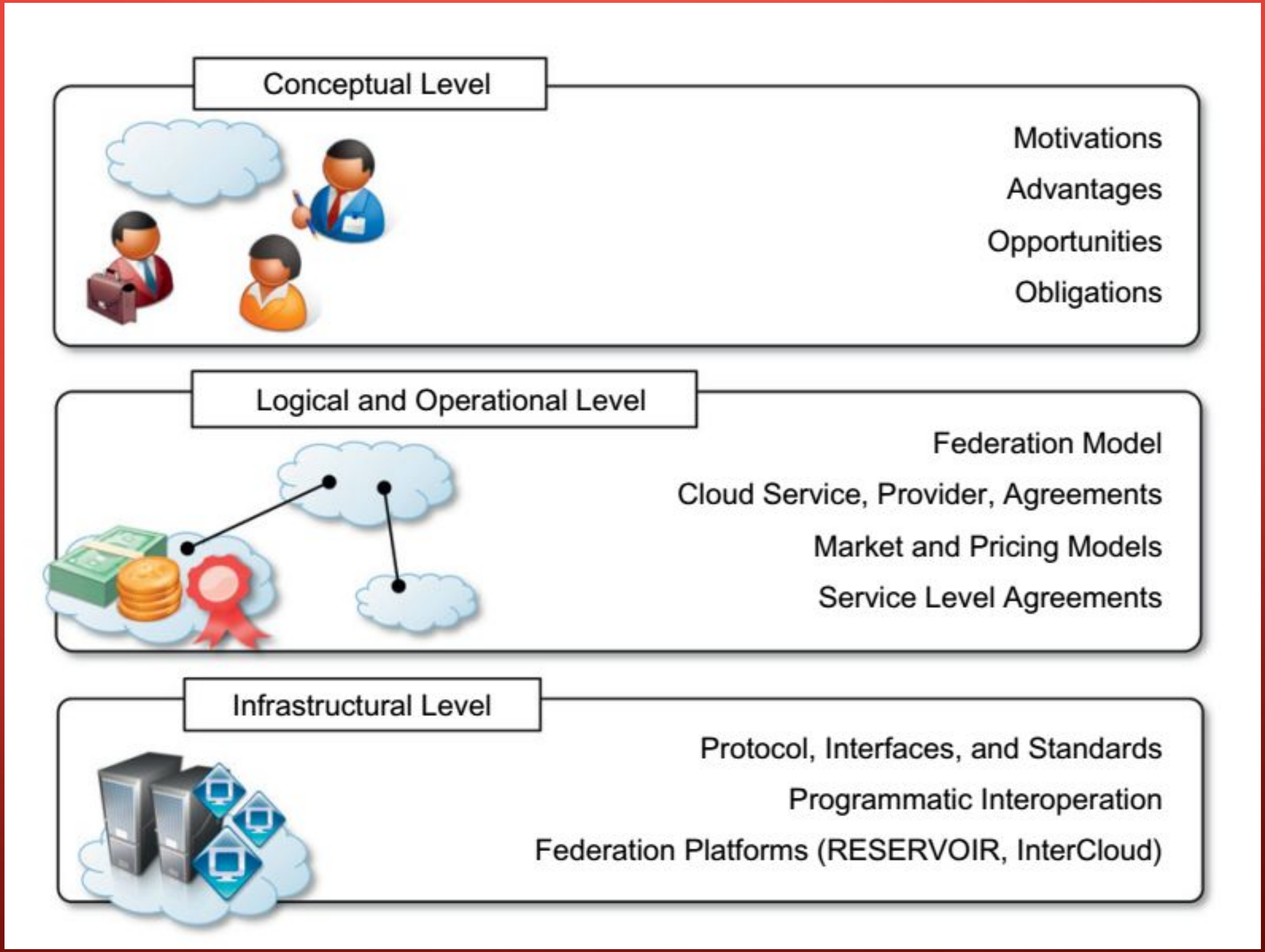


Cloud Federation is the concept of bringing different services offered by various Providers under a single platform .

FOUR LEVELS OF FEDERATION

- Creating a cloud federation involves research and development at different levels: conceptual, logical and operational, and infrastructural.
- The following figure provides a comprehensive view of the challenges faced in designing and implementing an organizational structure that coordinates together cloud services that belong to different administrative domains and makes them operate within a context of a single unified service middleware.
- Each cloud federation level presents different challenges and operates at a different layer of the IT stack.
- It then requires the use of different approaches and technologies.
- Taken together, the solutions to the challenges faced at each of these levels constitute a reference model for a cloud federation.

Cloud federation reference stack



CONCEPTUAL LEVEL

- The conceptual level addresses the challenges in presenting a cloud federation as a favorable solution with respect to the use of services leased by single cloud providers.
- In this level it is important to clearly identify the advantages for either service providers or service consumers in joining a federation and to delineate the new opportunities that a federated environment creates with respect to the single-provider solution. Elements of concern at this level are:
  - Motivations for cloud providers to join a federation
  - Motivations for service consumers to leverage a federation
  - Advantages for providers in leasing their services to other providers
  - Obligations of providers once they have joined the federation
  - Trust agreements between providers

The functional requirements include:

- Supplying low-latency access to customers, regardless of their location
- Handling bursts in demand
- Scaling existing applications and services beyond the capabilities of the owned infrastructure
- Make revenue from unused capacity

The motivations for joining a cloud federation also include nonfunctional requirements. The most relevant are the following

- Meeting compulsory regulations about the location of data
- Containing transient spikes in operational costs
- Disaster recovery

LOGICAL AND OPERATIONAL LEVEL

- The logical and operational level of a federated cloud identifies and addresses the challenges in devising a framework that enables the aggregation of providers that belong to different administrative domains within a context of a single overlay infrastructure, which is the cloud federation.
- At this level, policies and rules for interoperation are defined.
- Moreover, this is the layer at which decisions are made as to how and when to lease a service to or to leverage a service from another provider.
- The logical component defines a context in which agreements among providers are settled and services are negotiated, whereas the operational component characterizes and shapes the dynamic behavior of the federation as a result of the single providers' choices.

- This is the level where MOCC is implemented and realized.

It is important at this level to address the following challenges:

- How should a federation be represented?
- How should we model and represent a cloud service, a cloud provider, or an agreement?
- How should we define the rules and policies that allow providers to join a federation?
- What are the mechanisms in place for settling agreements among providers?
- What are providers responsibilities with respect to each other?
- When should providers and consumers take advantage of the federation?
- Which kinds of services are more likely to be leased or bought?

- The need for SLAs is an accepted fact in both academy and industry, since SLAs define more clearly what is leased or bought between different providers. Moreover, SLAs allow us to assess whether the services traded are delivered according to the expected quality profile.
- It is then possible to specify policies that regulate the transactions among providers and establish penalties in case of degraded service delivery.
- This is particularly important because it increases the level of trust that each party puts in cloud federation.
- SLAs have been in use since 1980 and originated in the telecommunications domain to define the QoS attached to a contract between a consumer and a network provider.
- From there on, they have been used in several fields, including Web services, grid computing, and cloud computing.
- The specific nature of SLA varies from domain to domain, but it can be generally defined as “an explicit statement of expectations and obligations that exist in a business relationship between two organizations: the service provider and the service consumer.

- SLAs define the provider's performance delivery ability, the consumer's performance profile, and the means to monitor and measure the delivered performance.
- An implementation of an SLA should specify
- **Purpose.** Objectives to achieve by using a SLA.
- **Restrictions.** Necessary steps or actions that need to be taken to ensure that the requested level of service is delivered.
- **Validity period.** Period of time during which the SLA is valid.
- **Scope.** Services that will be delivered to the consumer and services that are outside the SLA.
- **Parties.** Any involved organizations or individual and their roles (e.g. provider, consumer).
- **Service-level objectives (SLOs).** Levels of services on which both parties agree. These are expressed by means of service-level indicators such as availability, performance, and reliability.
- **Penalties.** The penalties that will occur if the delivered service does not achieve the defined SLOs.
- **Optional services.** Services that are not mandatory but might be required.
- **Administration.** Processes that are used to guarantee that SLOs are achieved and the related

- Parties that are willing to operate under an SLA engage a multistep process that generally involves the discovery or creation of the SLA, its operational phase, and its termination once its validity is over or there has been a violation of the contract.
- A more articulated process has been detailed by the Sun Internet Data Center Group, which includes six steps for the SLA life cycle:

1. Discover service provider.
2. Define SLA.
3. Establish agreement.
4. Monitor SLA violation.
5. Terminate SLA.
6. Enforce penalties for SLA violation.

- Currently, a very rudimentary level of SLA management is present and the interoperability among different cloud providers is mostly characterized by ad hoc aggregation.
- Within industry, SLAs are still unilateral arrangements that are imposed by service providers and that the user can only accept rather than negotiate.

INFRASTRUCTURAL LEVEL

- The infrastructural level addresses the technical challenges involved in enabling heterogeneous cloud computing systems to interoperate seamlessly.
- It deals with the technology barriers that keep separate cloud computing systems belonging to different administrative domains.
- By having standardized protocols and interfaces, these barriers can be overcome.
- In other words, this level for the federation is what the TCP/IP stack is for the Internet: a model and a reference implementation of the technologies enabling the interoperation of systems.
- Services for interoperation and interface may also find implementation at the SaaS level, especially for the realization of negotiations and of federated clouds.

At this level it is important to address the following issues:

- What kind of standards should be used?
- How should design interfaces and protocols be designed for interoperation?
- Which are the technologies to use for interoperation?
- How can we realize a software system, design platform components, and services enabling interoperability?
- Interoperation and composition among different cloud computing vendors is possible only by means of open standards and interfaces.
- Moreover, interfaces and protocols change considerably at each layer of the Cloud Computing Reference Model.
- As the more mature layer, the IaaS layer has evolved more in this sense.
- Almost every IaaS provider exposes Web interfaces for packaging virtual machine templates, launching, monitoring, and terminating virtual instances.
- Even though not standardized, these interfaces leverage the Web services and are quite

- The use of a common technology simplifies the interoperation among vendors, since a minimum amount of code is required to enable such interoperation.
- These APIs allow for defining an abstraction layer that uniformly accesses the services of several IaaS vendors.
- To fully support the vision of a cloud federation, more sophisticated capabilities need to be implemented.
- For instance, the possibility of dynamically moving virtual machine instances among different providers is essential to supporting dynamic load balancing among different IaaS vendors.
- In this direction, the Open Virtualization Format (OVF) aims to be a solution for this problem and may eventually be successful, since it has been endorsed by several cloud computing vendors.
- If we consider the PaaS layer, the interoperations become even harder since each cloud computing vendor provides its own runtime environment, which might differ from others in terms of implementation language, abstractions used to develop applications, and purpose.

- An interesting case for interoperability at the SaaS layer is provided by online office automation solutions such as Google Documents, Zoho Office, and others; several of them provide the capability to export and import documents to and from different formats, thus simplifying the exchange of data.
- Alternatively, composition seems to be a more attractive opportunity; different SaaS services can be glued together to provide users with more complex applications.
- Composition is also important in considering interoperability across cloud computing platforms operating at different layers.
- Even in this case it is important to note that, currently, cloud computing providers operating at one layer often implement on their own infrastructure any lower layer that is required to provide the service to the end user, and they are not willing to open their stack of technologies to support interoperation.

- The vision proposed by a federated environment of cloud service vendors still poses a lot of challenges at each level, especially the logical and infrastructural level, where appropriate system organizations need to be designed and effective technologies need to be deployed.
- Considerable research effort has been carried out on the logical and operational level, and initial implementations and drafts of interoperable technologies are now developed especially for the IaaS market segment.