Cloud Capabilities and Platform Features

Table 6.1 Important Cloud Platform Capabilities

Capability	Description
Physical or virtual computing platform	The cloud environment consists of some physical or virtual platforms. Virtual platforms have unique capabilities to provide isolated environments for different applications and users.
Massive data storage service, distributed file system	With large data sets, cloud data storage services provide large disk capacity and the service interfaces that allow users to put and get data. The distributed file system offers massive data storage service. It can provide similar interfaces as local file systems.
Massive database storage service	Some distributed file systems are sufficient to provide the underlying storage service application developers need to save data in a more semantic way. Just like DBMS in the traditional software stack, massive database storage services are needed in the cloud.
Massive data processing method and programming model	Cloud infrastructure provides thousands of computing nodes for even a very simple application. Programmers need to be able to harness the power of these machines without considering tedious infrastructure management issues such as handling network failure or scaling the running code to use all the computing facilities provided by the platforms.
Workflow and data query language support	The programming model offers abstraction of the cloud infrastructure. Similar to the SQL language used for database systems, in cloud computing, providers have built some workflow language as well as data query language to support better application logic.
Programming interface and service deployment	Web interfaces or special APIs are required for cloud applications: J2EE, PHP, ASP, or Rails. Cloud applications can use Ajax technologies to improve the user experience while using web browsers to access the functions provided. Each cloud provider opens its programming interface for accessing the data stored in massive storage.
Runtime support	Runtime support is transparent to users and their applications. Support includes distributed monitoring services, a distributed task scheduler, as well as distributed locking and other services. They are critical in running cloud applications.
Support services	Important support services include data and computing services. For example, clouds offer rich data services and interesting data parallel execution models like MapReduce.

PARALLEL AND DISTRIBUTED PROGRAMMING PARADIGMS

- Parallel and distributed program as a parallel program running on a set of computing engines or a distributed computing system.

- Distributed computing system is a set of computational engines connected by a network to achieve a common goal of running a job or an application.
  Ex: A computer cluster or network of workstations is an example of a distributed computing system.

- Parallel computing is the simultaneous use of more than one computational engine (not necessarily connected via a network) to run a job or an application.
  For instance, parallel computing may use either a distributed or a non-distributed computing system such as a multiprocessor platform.

Parallel Computing and Programming Paradigms

Distributed computing system consisting of a set of networked nodes or workers. The system issues for running a typical parallel program in either a parallel or a distributed manner would include the following

- **Partitioning** This is applicable to both computation and data as follows

- **Computation partitioning** This splits a given job or a program into smaller tasks. Partitioning greatly depends on correctly identifying portions of the job or program that can be performed concurrently. In other words, upon identifying parallelism in the structure of the program, it can be divided into parts to be run on different workers. Different parts may process different data or a copy of the same data.

- **Data partitioning** This splits the input or intermediate data into smaller pieces. Similarly, upon identification of parallelism in the input data, it can also be divided into pieces to be processed on different workers. Data pieces may be processed by different parts of a program or a copy of the same program.

- **Mapping** This assigns the either smaller parts of a program or the smaller pieces of data to underlying resources. This process aims to appropriately assign such parts or pieces to be run simultaneously on different workers and is usually handled by resource allocators in the system.

- **Synchronization** Because different workers may perform different tasks, synchronization and coordination among workers is necessary so that race conditions are prevented and data dependency among different workers is properly managed. Multiple accesses to a shared resource by different workers may raise race conditions, whereas data dependency happens when a worker needs the processed data of other workers.

- **Communication** Because data dependency is one of the main reasons for communication among workers, communication is always triggered when the intermediate data is sent to workers.

  - **Scheduling** For a job or program, when the number of computation parts (tasks) or data pieces is more than the number of available workers, a scheduler selects a sequence of tasks or data pieces to be assigned to the workers. It is worth noting that the resource allocator performs the actual mapping of the computation or data pieces to workers, while the scheduler only picks the next part from the queue of unassigned tasks based on a set of rules called the scheduling policy. For multiple jobs or programs, a scheduler selects a sequence of jobs or programs to be run on the distributed computing system.

Because handling the whole data flow of parallel and distributed programming is very time consuming and requires specialized knowledge of programming, dealing with these issues may affect the productivity of the programmer and may even result in affecting the program's time to market.
Therefore, simplicity of writing parallel programs is an important metric for parallel and distributed programming paradigms.

Other motivations behind parallel and distributed programming models are
(1) to improve productivity of programmers,
(2) to decrease programs' time to market,
(3) to leverage underlying resources more efficiently,
(4) to increase system throughput, and
(5) to support higher levels of abstraction

MapReduce, Hadoop, and Dryad are three of the most recently proposed parallel and distributed programming models. They were developed for information retrieval applications but have been shown to be applicable for a variety of important applications

MapReduce

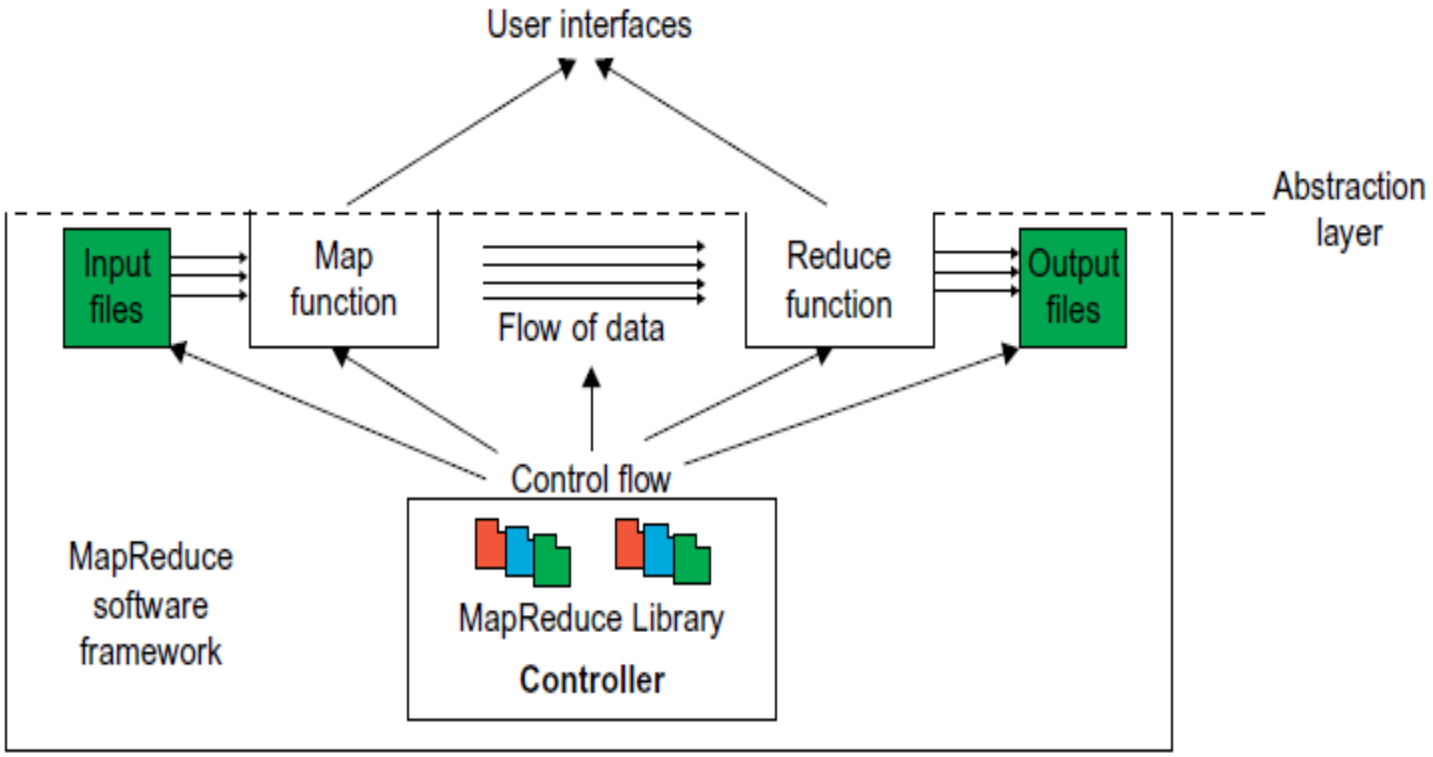
MapReduce, is a software framework which supports parallel and distributed computing on large data sets

- This software framework abstracts the data flow of running a parallel program on a distributed computing system by providing users with two interfaces in the form of two functions: **Map and Reduce.**

- Users can override these two functions to interact with and manipulate the data flow of running their programs.

- Framework hides the implementation of all data flow steps such as data partitioning, mapping, synchronization, communication, and scheduling.

- In this framework the “value” part of the data, (key, value), is the actual data, and the “key” part is only used by the MapReduce controller to control the data flow



MapReduce framework: Input data flows through the Map and Reduce functions to generate the output result under the control flow using MapReduce software library.

Special user interfaces are used to access the Map and Reduce resources.

- Therefore, the user overrides the Map and Reduce functions first and then invokes the provided MapReduce (Spec, & Results) function from the library to start the flow of data.

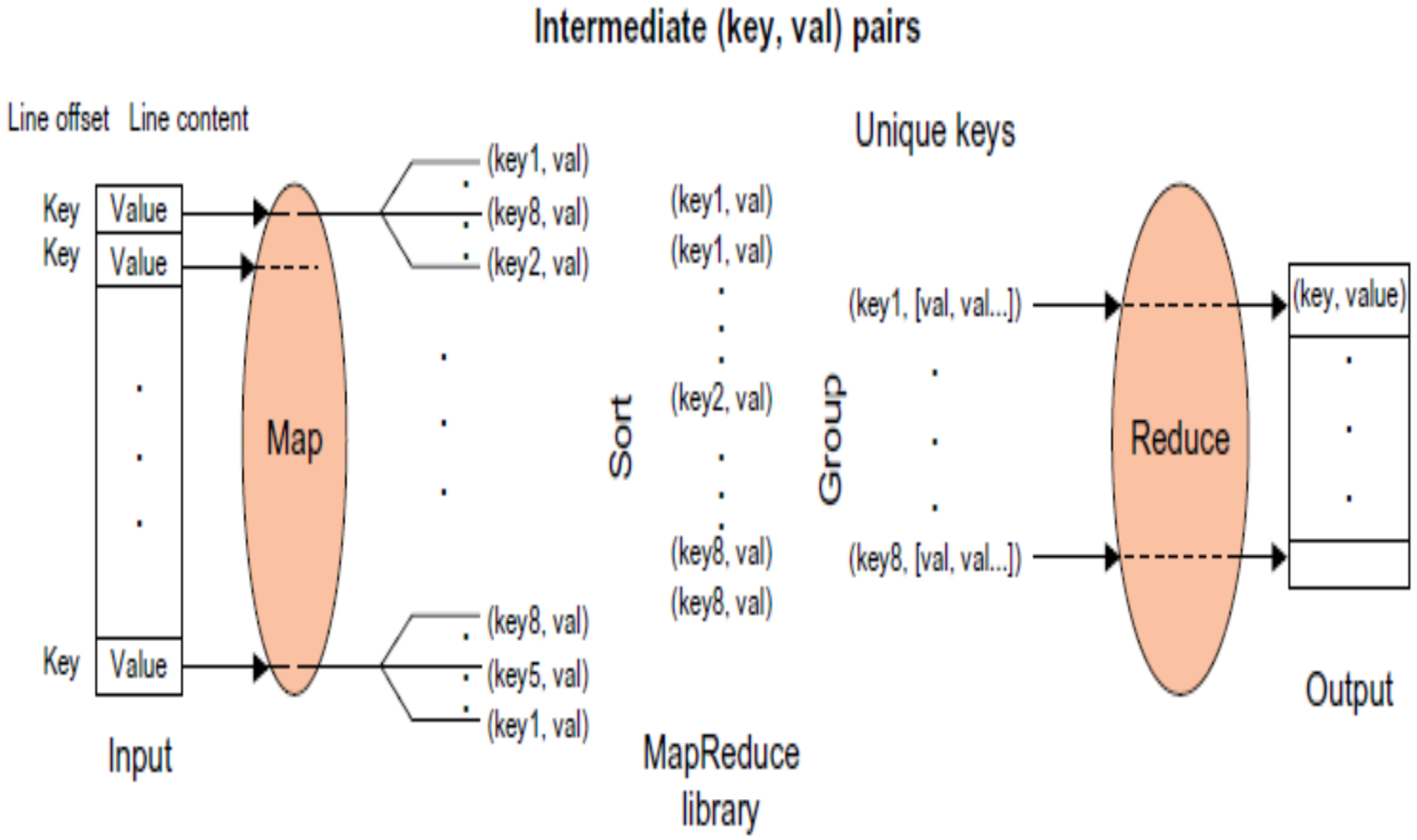
- The MapReduce function, MapReduce (Spec, & Results), takes an important parameter which is a specification object, the Spec.

- This object is first initialized inside the user's program, and then the user writes code to fill it with the names of input and output files, as well as other optional tuning parameters.

- This object is also filled with the name of the Map and Reduce functions to identify these user defined functions to the MapReduce library

The overall structure of a user's program containing the Map, Reduce, and the Main functions is given below.

```
Map Function (... . )
{
... ...
}
Reduce Function (... . )
{
... ...
}
Main Function (... . )
{
Initialize Spec object
... ...
MapReduce (Spec, & Results)
}
```


Intermediate (key, val) pairs
Line offset
Line content
Key
Value
Key
Value
Map
(key1, val)
(key8, val)
(key2, val)
(key8, val)
(key5, val)
(key1, val)
Input
Sort
(key1, val)
(key1, val)
(key2, val)
(key8, val)
(key8, val)
MapReduce library
Unique keys
Group
(key1, [val, val...])
(key8, [val, val...])
Reduce
(key, value)
Output

- The input data to the Map function is in the form of a (key, value) pair. For example, the key is the line offset within the input file and the value is the content of the line.
- The output data from the Map function is structured as (key, value) pairs called intermediate (key, value) pairs.
- In other words, the user-defined Map function processes each input (key, value) pair and produces a number of (zero, one, or more) intermediate (key, value) pairs. Here, the goal is to process all input (key, value) pairs to the Map function in parallel
    - In turn, the Reduce function receives the intermediate (key, value) pairs in the form of a group of intermediate values associated with one intermediate key, (key, [set of values]).
    - MapReduce framework forms these groups by first sorting the intermediate (key, value) pairs and then grouping values with the same key.
    - It should be noted that the data is sorted to simplify the grouping process. The Reduce function processes each (key, [set of values]) group and produces a set of (key, value) pairs as output.

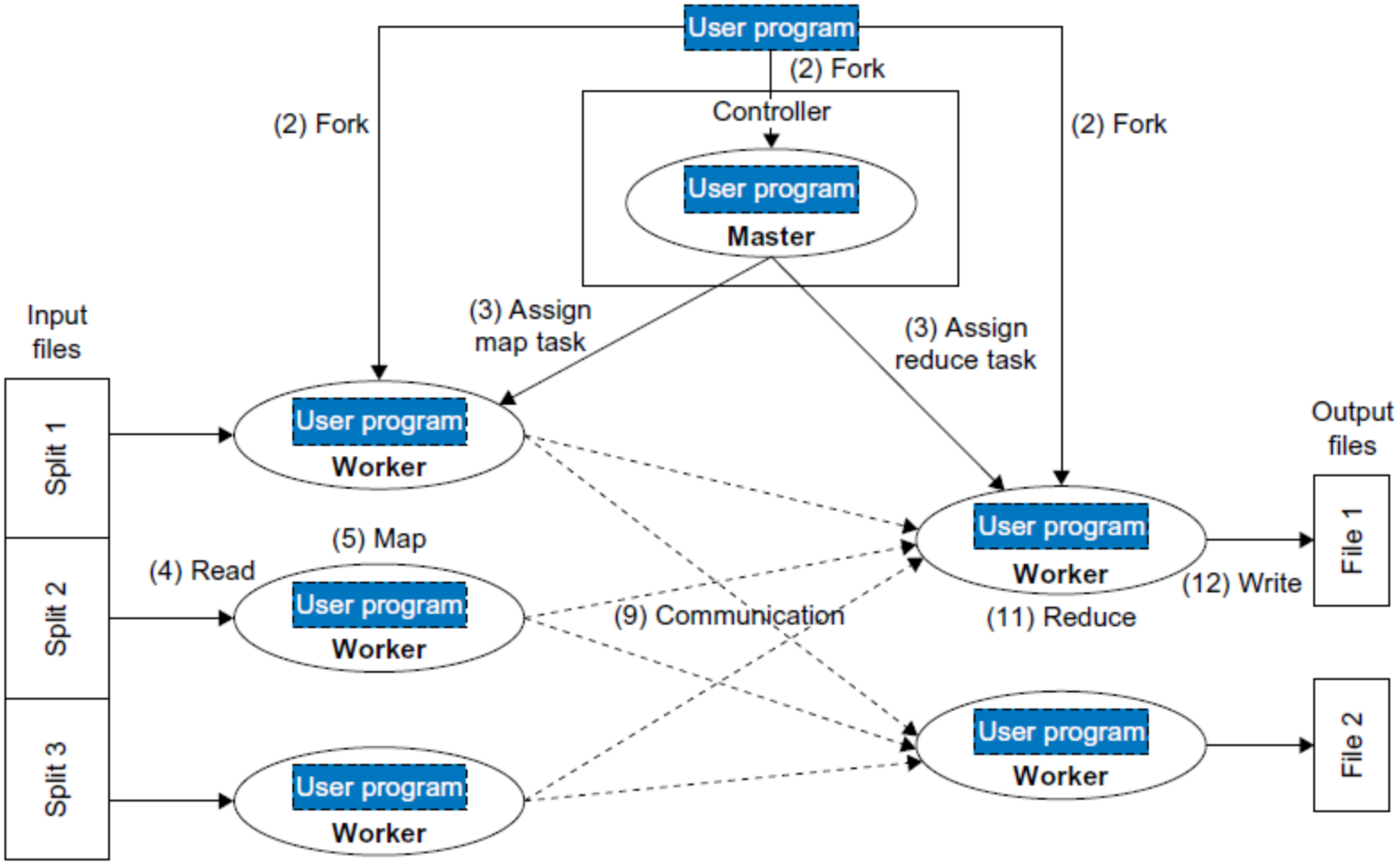


FIGURE 6.6

Control flow implementation of the MapReduce functionalities in Map workers and Reduce workers (running user programs) from input files to the output files under the control of the master user program.

Strategy to Solve MapReduce Problems

Problem 1: Counting the number of occurrences of each word in a collection of documents
Solution: unique “key”: each word, intermediate “value”: number of occurrences

Problem 2: Counting the number of occurrences of words having the same size, or the same
number of letters, in a collection of documents
Solution: unique “key”: each word, intermediate “value”: size of the word

Problem 3: Counting the number of occurrences of anagrams in a collection of documents.
Anagrams are words with the same set of letters but in a different order (e.g., the words “listen” and “silent”).
Solution: unique “key”: alphabetically sorted sequence of letters for each word (e.g., “eilnst”),
intermediate “value”: number of occurrences

The MapReduce software framework was first proposed and implemented by Google. The first implementation was coded in C. The implementation takes advantage of GFS [53] as the underlying layer. MapReduce could perfectly adapt itself to GFS. GFS is a distributed file system where files are divided into fixed-size blocks (chunks) and blocks are distributed and stored on cluster nodes.

Twister Iterative MapReduce

It is important to understand the performance of different runtimes and, in particular, to compare MPI(The message passing interface (**MPI**) is a standardized means of exchanging messages between multiple computers running a parallel program across distributed memory.) and MapReduce.

The two major sources of parallel overhead are load imbalance and communication (which is equivalent to synchronization overhead as communication synchronizes parallel units.

The communication overhead in MapReduce can be quite high, for two reasons:

- MapReduce reads and writes via files, whereas MPI transfers information directly between nodes over the network.

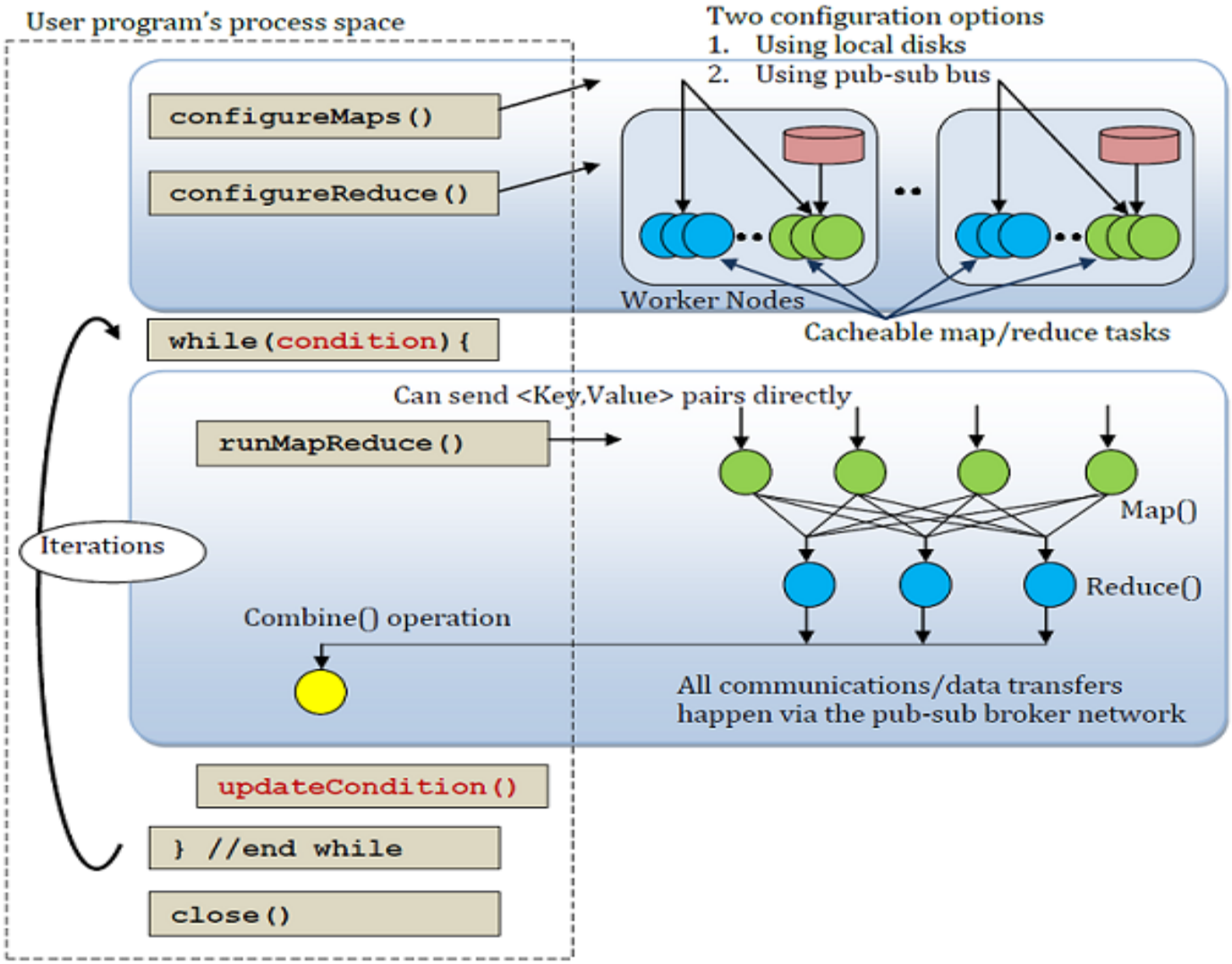
- MPI does not transfer all data from node to node, but just the amount needed to update information. We can call the MPI flow δ flow and the MapReduce flow full data flow.

The same phenomenon is seen in all “classic parallel” loosely synchronous applications which typically exhibit an iteration structure over compute phases followed by communication phases. We can address the performance issues with two important changes:

1. Stream information between steps without writing intermediate steps to disk.
2. Use long-running threads or processors to communicate the δ (between iterations) flow.

***Twister* is a lightweight MapReduce** runtime we have developed by incorporating these enhancements.
Twister provides the following features to support MapReduce computations

- Distinction on static and variable data
- Configurable long running (cacheable) map/reduce tasks
- Pub/sub messaging based communication/data transfers
- Efficient support for Iterative MapReduce computations (extremely faster than Hadoop or Dryad/DryadLINQ)
- Combine phase to collect all reduce outputs
- Data access via local disks
- Lightweight (~5600 lines of Java code)
- Support for typical MapReduce computations
- Tools to manage data


User program's process space
Two configuration options
1. Using local disks
2. Using pub-sub bus
configureMaps()
configureReduce()
Worker Nodes
Cacheable map/reduce tasks
while(condition){
Can send <Key,Value> pairs directly
runMapReduce()
Map()
Reduce()
Iterations
Combine() operation
All communications/data transfers
happen via the pub-sub broker network
updateCondition()
} //end while
close()

Hadoop Library from Apache

The Hadoop Distributed File System (HDFS) is a distributed file system designed to run on commodity hardware. It has many similarities with existing distributed file systems. However, the differences from other distributed file systems are significant. HDFS is highly fault-tolerant and is designed to be deployed on low-cost hardware. HDFS provides high throughput access to application data and is suitable for applications that have large data sets. HDFS relaxes a few POSIX requirements to enable streaming access to file system data.

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

Features of Hadoop

- Hardware failure is the norm rather than the exception. An HDFS instance may consist of hundreds or thousands of server machines, each storing part of the file system's data. The fact that there are a huge number of components and that each component has a non-trivial probability of failure means that some component of HDFS is always non-functional. Therefore, detection of faults and quick, automatic recovery from them is a core architectural goal of HDFS.
- Applications that run on HDFS need streaming access to their data sets. They are not general purpose applications that typically run on general purpose file systems. HDFS is designed more for batch processing rather than interactive use by users. The emphasis is on high throughput of data access rather than low latency of data access.
- Applications that run on HDFS have large data sets. A typical file in HDFS is gigabytes to terabytes in size. Thus, HDFS is tuned to support large files. It should provide high aggregate data bandwidth and scale to hundreds of nodes in a single cluster. It should support tens of millions of files in a single instance.
- HDFS applications need a write-once-read-many access model for files. A file once created, written, and closed need not be changed except for appends and truncates.

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

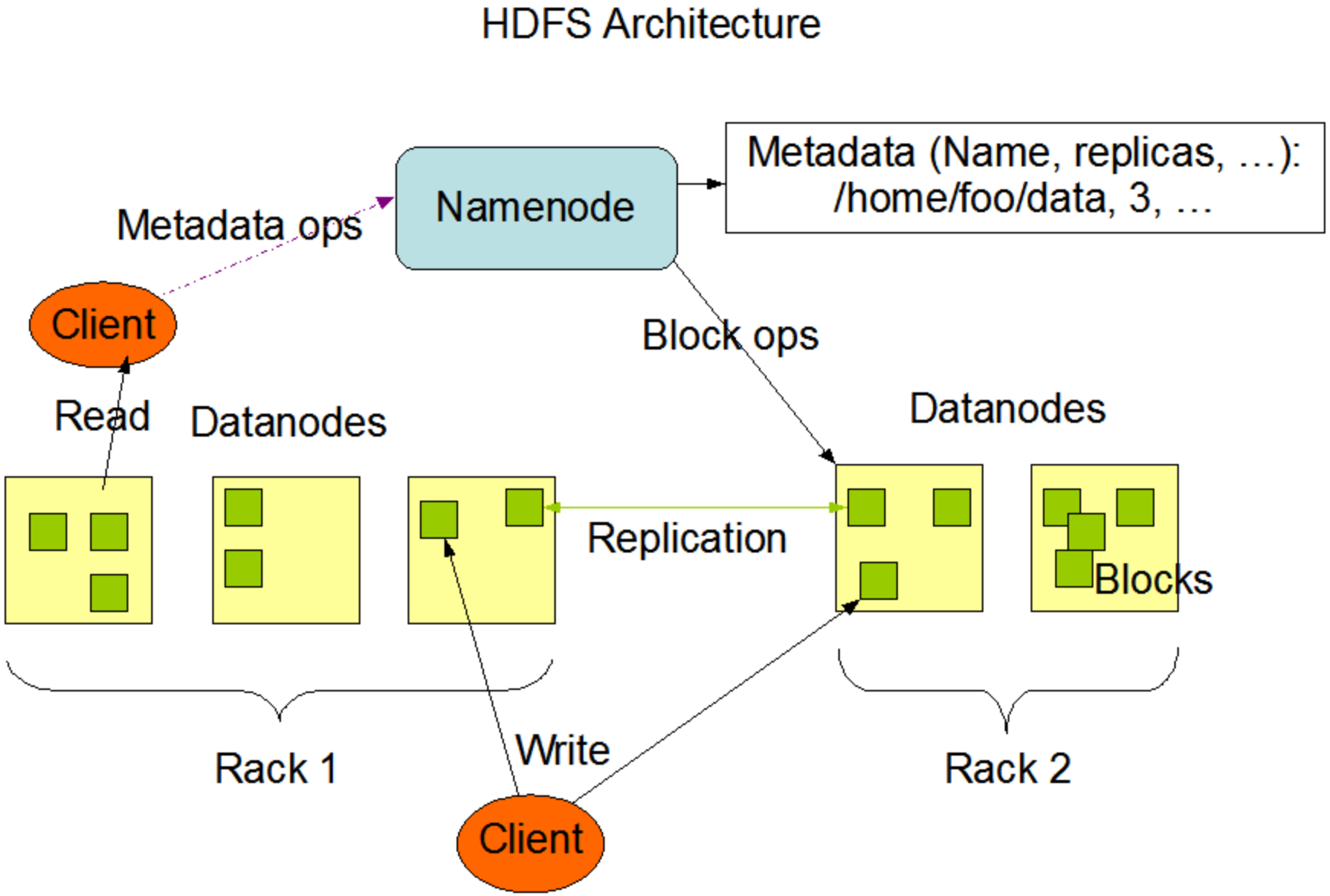
- HDFS has a master/slave architecture. An HDFS cluster consists of a single NameNode, a master server that manages the file system namespace and regulates access to files by clients.
- In addition, there are a number of DataNodes as workers (slaves). usually one per node in the cluster, which manage storage attached to the nodes that they run on.
- HDFS exposes a file system namespace and allows user data to be stored in files
- 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

- The NameNode executes file system namespace operations like opening, closing, and renaming files and directories. It also determines the mapping of blocks to DataNodes.

- The DataNodes are responsible for serving read and write requests from the file system's clients. The DataNodes also perform block creation, deletion, and replication upon instruction from the NameNode.

- The NameNode and DataNode are pieces of software designed to run on commodity machines. These machines typically run a GNU/Linux operating system (OS). HDFS is built using the Java language; any machine that supports Java can run the NameNode or the DataNode software.

- Usage of the highly portable Java language means that HDFS can be deployed on a wide range of machines. A typical deployment has a dedicated machine that runs only the NameNode software. Each of the other machines in the cluster runs one instance of the DataNode software.


HDFS Architecture
Namenode
Metadata (Name, replicas, …):
/home/foo/data, 3, …
Metadata ops
Client
Block ops
Read
Datanodes
Datanodes
Replication
Blocks
Rack 1
Write
Rack 2
Client

The existence of a single NameNode in a cluster greatly simplifies the architecture of the system. The NameNode is the arbitrator and repository for all HDFS metadata. The system is designed in such a way that user data never flows through the NameNode

- The system having the namenode acts as the master server and it does the following tasks −
- Manages the file system namespace.
- Regulates client's access to files.
- It also executes file system operations such as renaming, closing, and opening files and directories.

HDFS is designed to reliably store very large files across machines in a large cluster. It stores each file as a sequence of blocks. The blocks of a file are replicated for fault tolerance. The block size and replication factor are configurable per file.

All blocks in a file except the last block are the same size, while users can start a new block without filling out the last block to the configured block size after the support for variable length block was added to append and hsync.

- An application can specify the number of replicas of a file. The replication factor can be specified at file creation time and can be changed later. Files in HDFS are write-once (except for appends and truncates) and have strictly one writer at any time.

- The NameNode makes all decisions regarding replication of blocks. It periodically receives a Heartbeat and a Blockreport from each of the DataNodes in the cluster.
- Receipt of a Heartbeat implies that the DataNode is functioning properly. A Blockreport contains a list of all blocks on a DataNode.

On startup, the NameNode enters a special state called Safemode. Replication of data blocks does not occur when the NameNode is in the Safemode state. The NameNode receives Heartbeat and Blockreport messages from the DataNodes.
A Blockreport contains the list of data blocks that a DataNode is hosting. Each block has a specified minimum number of replicas.

A block is considered safely replicated when the minimum number of replicas of that data block has checked in with the NameNode. After a configurable percentage of safely replicated data blocks checks in with the NameNode (plus an additional 30 seconds), the NameNode exits the Safemode state.

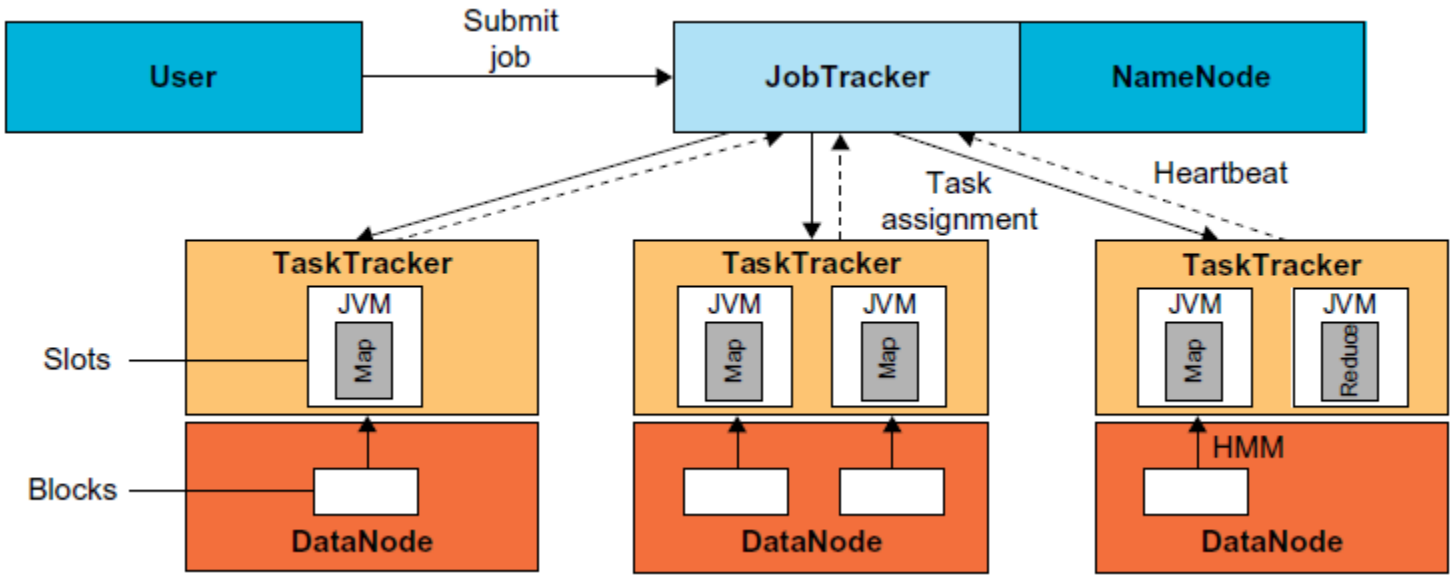
It then determines the list of data blocks (if any) that still have fewer than the specified number of replicas. The NameNode then replicates these blocks to other DataNodes.

Running 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.



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. The number of reduce tasks is predetermined by the user, and there is no locality consideration in assigning them.

• 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.

Table 6.10 Application Classification for Parallel and Distributed Systems

Category	Class	Description	Machine Architecture
1	Synchronous	The problem class can be implemented with instruction-level lockstep operation as in SIMD architectures.	SIMD
2	Loosely synchronous (BSP or bulk synchronous processing)	These problems exhibit iterative compute-communication stages with independent compute (map) operations for each CPU that are synchronized with a communication step. This problem class covers many successful MPI applications including partial differential equation solutions and particle dynamics applications.	MIMD on MPP (massively parallel processor)
3	Asynchronous	Illustrated by Compute Chess and Integer Programming; combinatorial search is often supported by dynamic threads. This is rarely important in scientific computing, but it is at the heart of operating systems and concurrency in consumer applications such as Microsoft Word.	Shared memory
4	Pleasingly parallel	Each component is independent. In 1988, Fox estimated this at 20 percent of the total number of applications, but that percentage has grown with the use of grids and data analysis applications including, for example, the Large Hadron Collider analysis for particle physics.	Grids moving to clouds
5	Metaproblems	These are coarse-grained (asynchronous or data flow) combinations of categories 1-4 and 6. This area has also grown in importance and is well supported by grids and described by workflow in Section 3.5.	Grids of clusters
6	MapReduce++ (Twister)	This describes file (database) to file (database) operations which have three subcategories (see also Table 6.11): 6a) Pleasingly Parallel Map Only (similar to category 4) 6b) Map followed by reductions 6c) Iterative "Map followed by reductions" (extension of current technologies that supports linear algebra and data mining)	Data-intensive clouds a) Master-worker or MapReduce b) MapReduce c) Twister

Google App Engine

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 JVM based 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.

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.

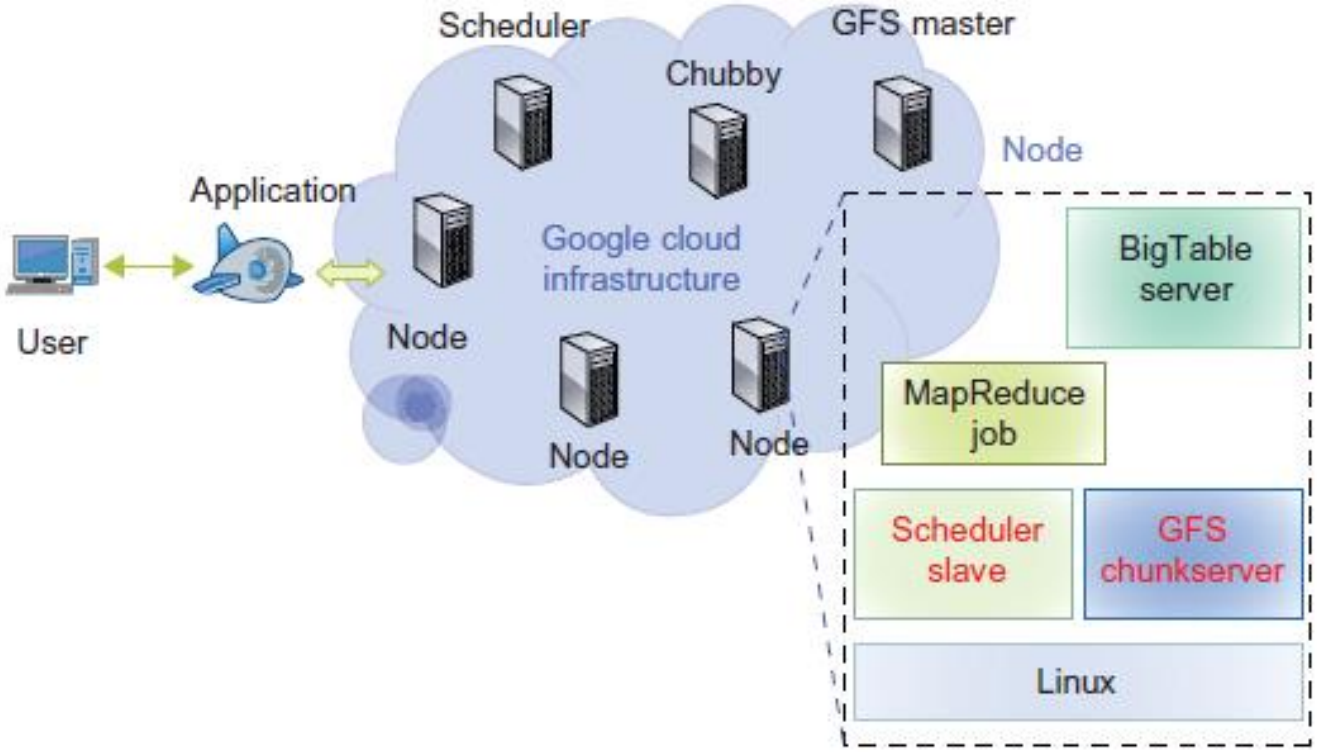
Java offers Java Data Object (JDO) and Java Persistence API (JPA) interfaces implemented by the open source Data Nucleus Access platform, while Python has a SQL-like query language called GQL.

- 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

With these building blocks, Google has built many cloud applications.
Figure shows the overall architecture of the Google cloud infrastructure.

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.



- 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. Inside each data center, there are thousands of servers forming different 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 GAE is not an infrastructure platform, but rather an application development platform for users.

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.

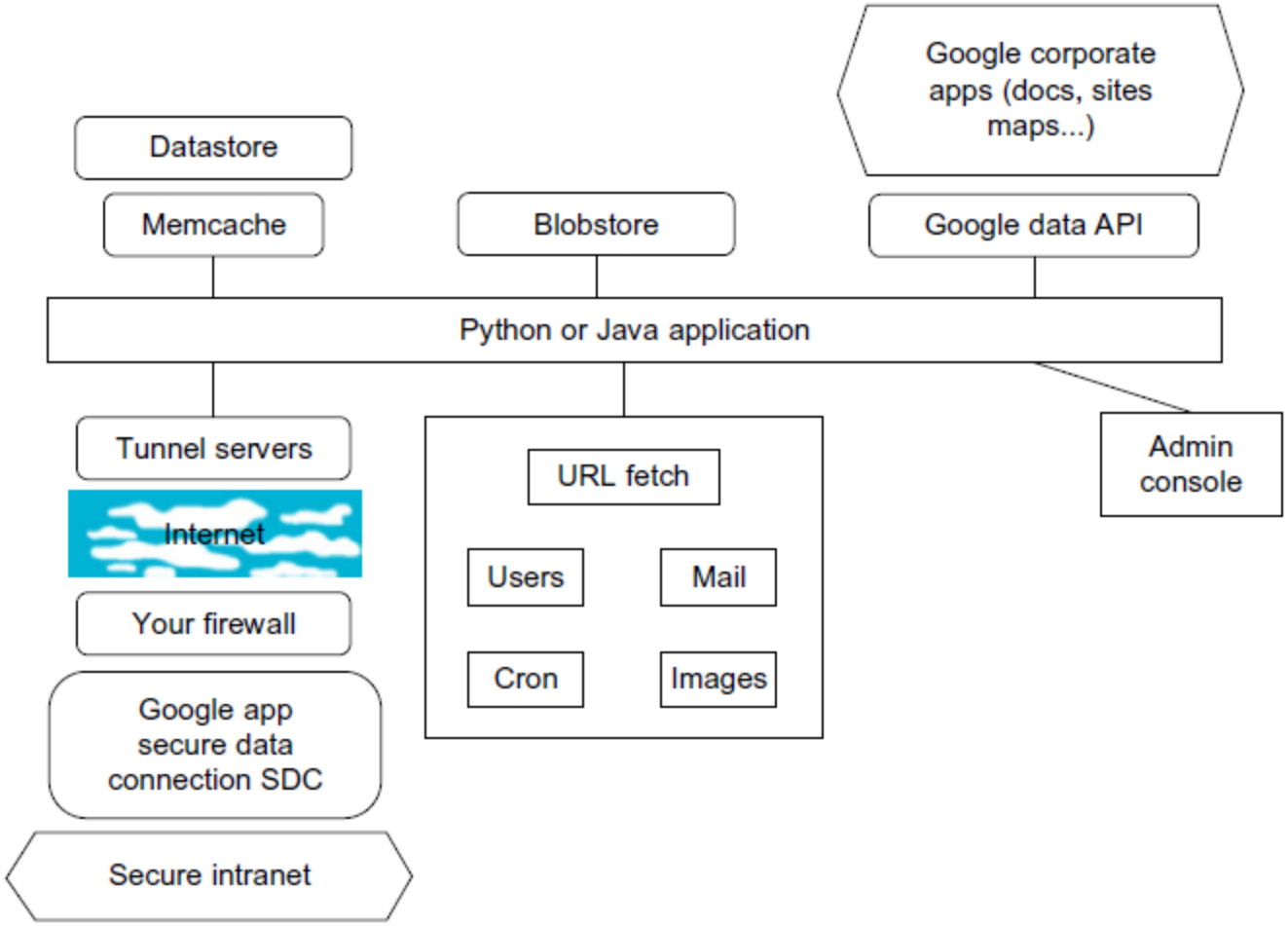


FIGURE 6.17

Programming environment for Google AppEngine.

Amazon Web Services (AWS)

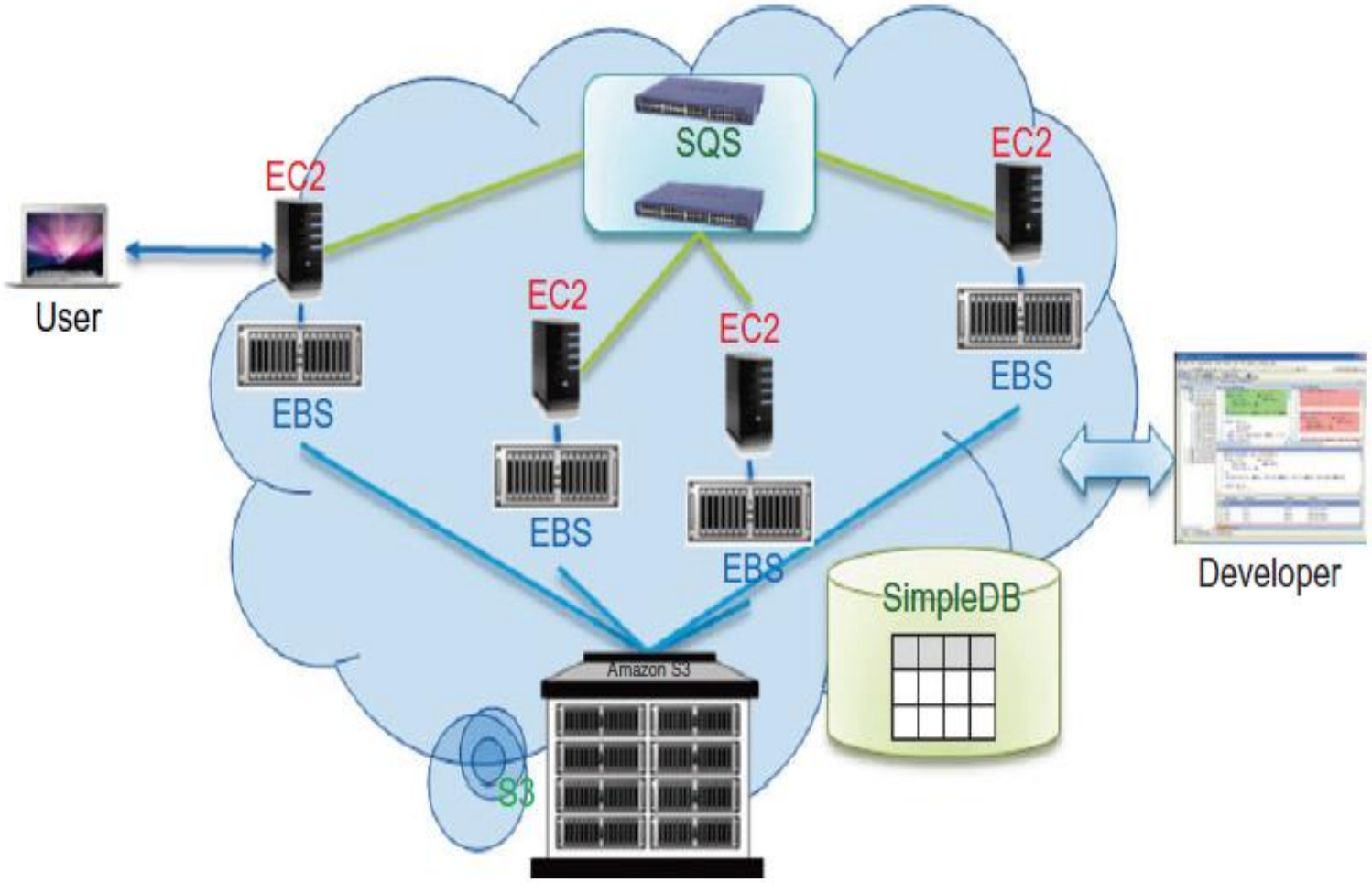
Amazon has been a leader in providing public cloud services. Amazon applies the IaaS model in providing its services

EC2 provides the virtualized platforms to the host VMs where the cloud application can run.

S3 (Simple Storage Service) provides the object-oriented storage service for users. Amazon S3 provides a simple web services interface that can be used to store and retrieve any amount of data, at any time, from anywhere on the web. S3 provides the object-oriented storage service for users

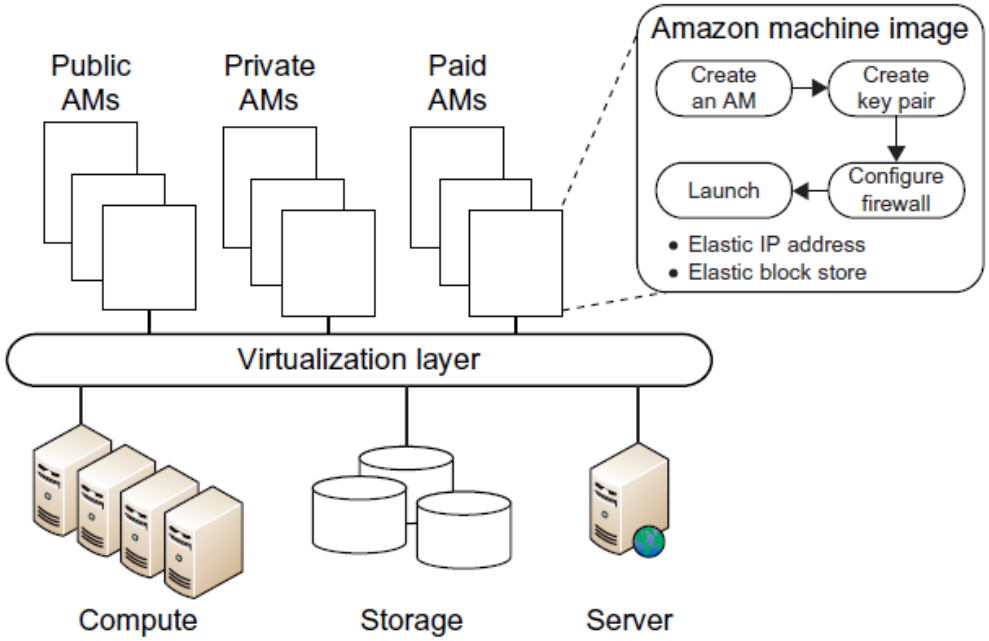
EBS (Elastic Block Service) provides the block storage interface which can be used to support traditional applications.

SQS stands for Simple Queue Service, and its job is to ensure a reliable message service between two processes. The message can be kept reliably even when the receiver processes are not running. Users can access their objects through SOAP with either browsers or other client programs which support the SOAP standard.


SQS
EC2
EC2
EC2
EC2
User
EBS
EBS
EBS
EBS
SimpleDB
Amazon S3
S3
Developer

Amazon was the first company to introduce VMs in application hosting. Customers can rent VMs instead of physical machines to run their own applications. By using VMs, customers can load any software of their choice.

The elastic feature of such a service is that a customer can create, launch, and terminate server instances as needed, paying by the hour for active servers. Amazon provides several types of preinstalled VMs. Instances are often called Amazon Machine Images (AMIs) which are preconfigured with operating systems based on Linux or Windows, and additional software.



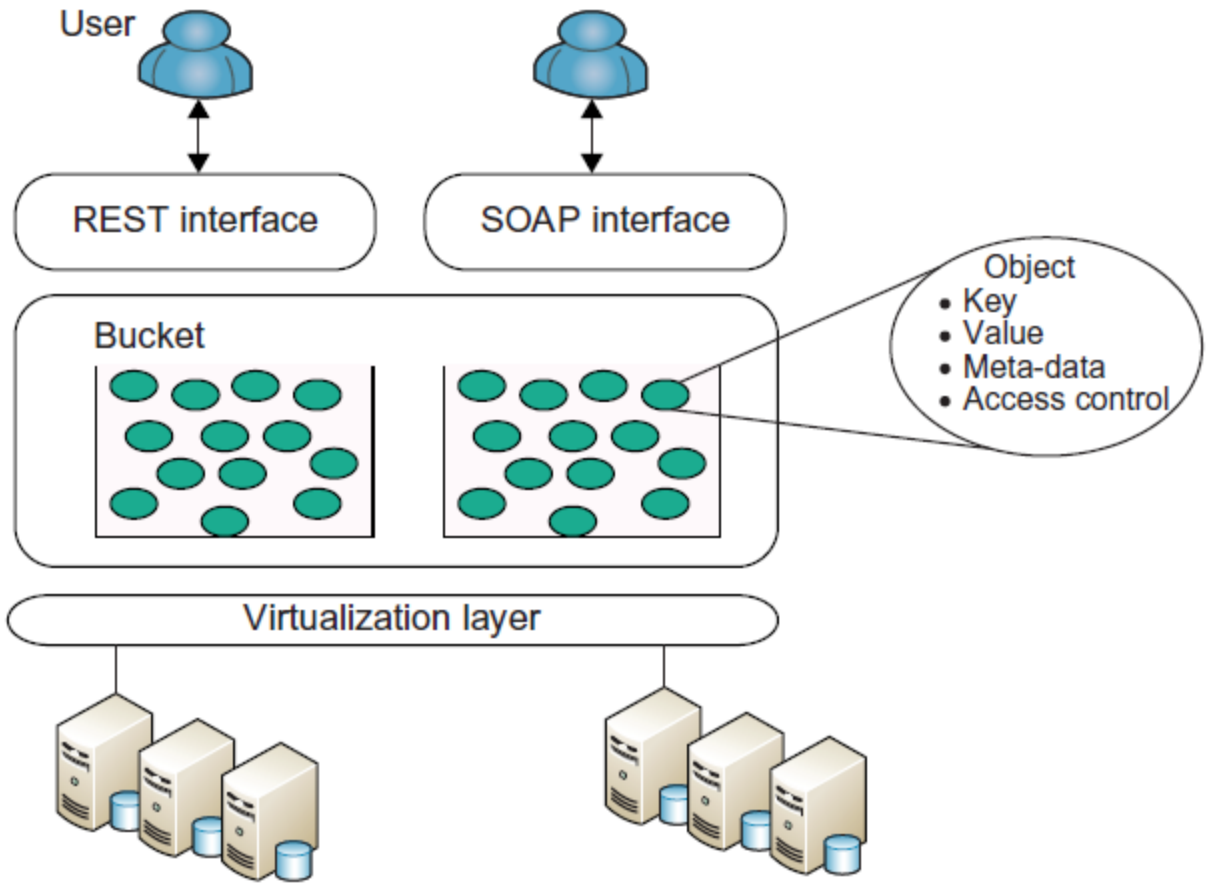
The fundamental operation unit of S3 is called an object. Each object is stored in a bucket and retrieved via a unique, developer-assigned key. In other words, the bucket is the container of the object.

Besides unique key attributes, the object has other attributes such as values, metadata, and access control information. From the programmer's perspective, the storage provided by S3 can be viewed as a very coarse-grained key-value pair.
Through the key-value programming interface, users can write, read, and delete objects containing from 1 byte to 5 gigabytes of data each.

There are two types of web service interface for the user to access the data stored in Amazon clouds. One is a REST (web 2.0) interface, and the other is a SOAP interface.

Authentication mechanisms to ensure that data is kept secure from unauthorized access. Objects can be made private or public, and rights can be granted to specific users.

• Per-object URLs and ACLs (access control lists).


User
REST interface
SOAP interface
Bucket
Object
• Key
• Value
• Meta-data
• Access control
Virtualization layer

The Elastic Block Store (EBS) provides the volume block interface for saving and restoring the virtual images of EC2 instances.

Traditional EC2 instances will be destroyed after use. The status of EC2 can now be saved in the EBS system after the machine is shut down. Users can use EBS to save persistent data and mount to the running instances of EC2.

Note that S3 is "Storage as a Service" with a messaging interface. EBS is analogous to a distributed file system accessed by traditional OS disk access mechanisms. EBS allows you to create storage volumes from 1 GB to 1 TB that can be mounted as
EC2 instances.

SimpleDB provides a simplified data model based on the relational database data model. Structured data from users must be organized into domains.

Each domain can be considered a table. The items are the rows in the table. A cell in the table is recognized as the value for a specific attribute (column name) of the corresponding row. This is similar to a table in a relational database.

However, it is possible to assign multiple values to a single cell in the table. This is not permitted in a traditional relational database which wants to maintain data consistency.

Windows Azure

Azure is a cloud computing platform which was launched by Microsoft in February 2010. It is an open and flexible cloud platform which helps in development, data storage, service hosting, and service management. The Azure tool hosts web applications over the internet with the help of Microsoft data centers.

Azure as PaaS (Platform as a Service)

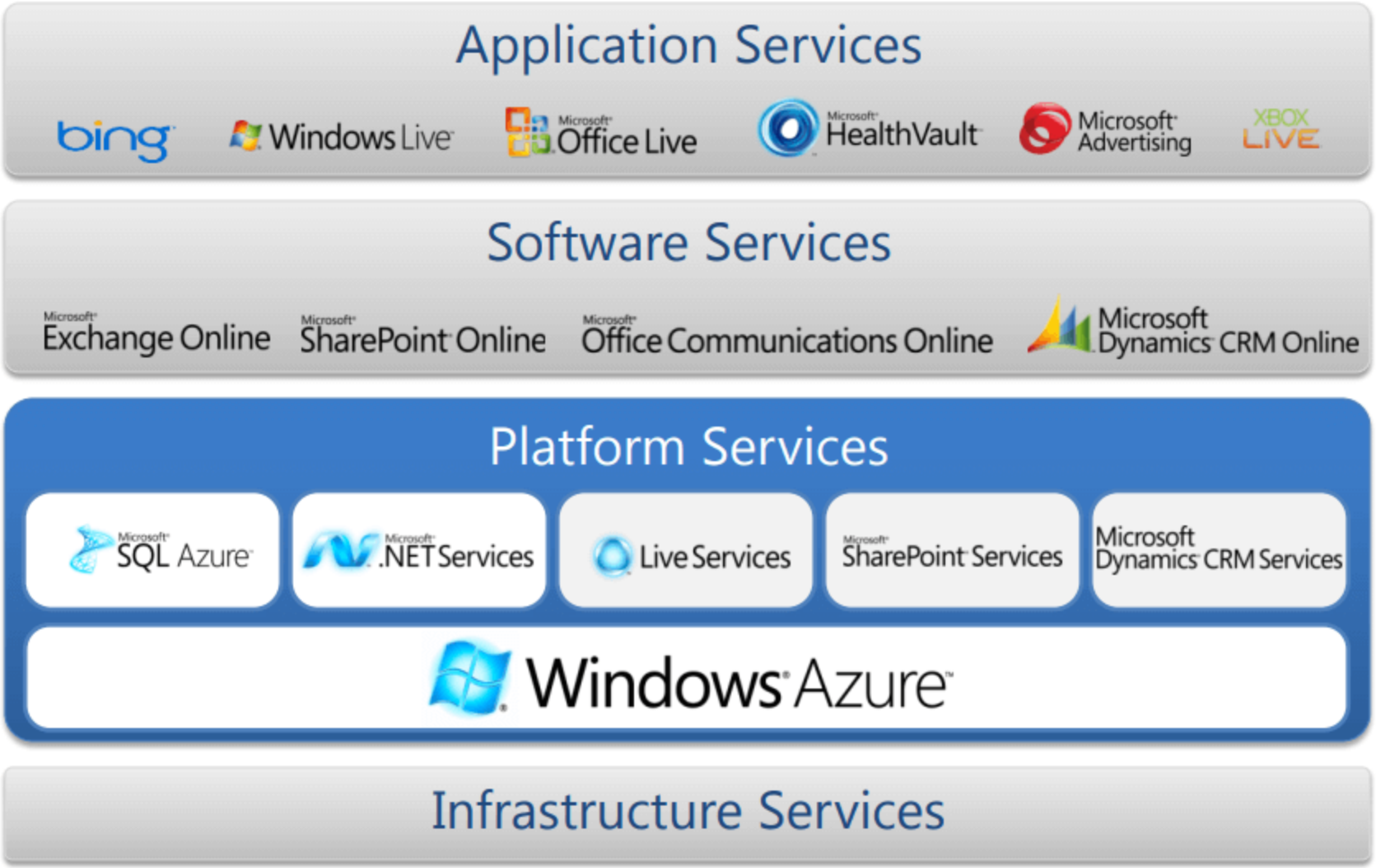
A platform is provided to clients to develop and deploy software. The clients can focus on the application development rather than having to worry about hardware and infrastructure. It also takes care of most of the operating systems, servers and networking issues.

There are mainly three types of clouds in Microsoft Azure are:

PAAS

SAAS

IAAS


Application Services
bing
Windows Live
Office Live
HealthVault
Microsoft Advertising
XBOX LIVE
Software Services
Exchange Online
SharePoint Online
Office Communications Online
Microsoft Dynamics CRM Online
Platform Services
SQL Azure
.NET Services
Live Services
SharePoint Services
Microsoft Dynamics CRM Services
Windows Azure
Infrastructure Services

Azure as IaaS

IaaS(Infrastructure as a Service) is the foundational cloud platform layer. This Azure service is used by IT administrators for processing, storage, networks or any other fundamental computer operations. It allows users to run arbitrary software.

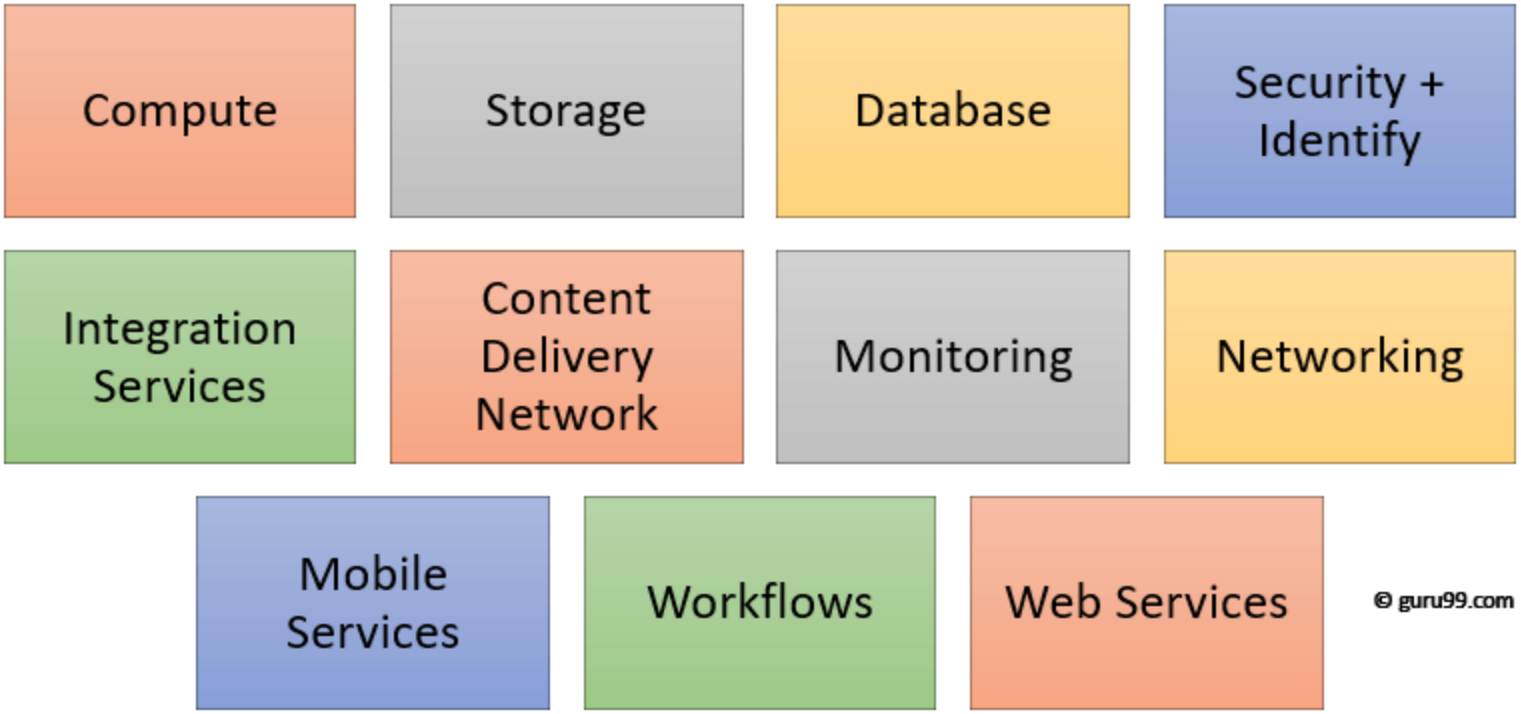
Azure as PaaS

PaaS is a computing platform which includes an operating system, programming language execution environment, database or web services. This Azure service is used by developers and application providers.

Azure As SaaS

SaaS (Software as a Service) is software which is centrally hosted and managed. It is a single version of the application is used for all customers. You can scale out to multiple instances. This helps you to ensure the best performance in all locations. The software is licensed through a monthly or annual subscription. MS Exchange, Office, Dynamics are offered as a SaaS

Azure Domains (Components)



Compute

It offers computing operations like app hosting, development, and deployment in Azure Platform. It has the following components:

Virtual Machine: Allows you to deploy any language, workload in any operating system

Virtual Machine Scale Sets: Allows you to create thousands of similar virtual machines in minutes

Azure Container Service: Create a container hosting solution which is optimized for Azure. You scale and arrange applications using Kube, DC/OS, Swarm or Docker

Storage

Azure store is a cloud storage solution for modern applications. It is designed to meet the needs of their customer's demand for scalability. It allows you to store and process hundreds of terabytes of data. It has the following components:

Blob Storage: Azure Blob storage is a service which stores unstructured data in the cloud as objects/blobs. You can store any type of text or binary data, such as a document, media file, or application installer.

Queue Storage: It provides cloud messaging between application components. It delivers asynchronous messaging to establish communication between application components.

Database

This category includes Database as a Service (DBaaS) which offers SQL and NoSQL tools. It also includes databases like Azure Cosmos DB and Azure Database for PostgreSQL. It has the following components:

SQL Database: It is a relational database service in the Microsoft cloud based on the market-leading Microsoft SQL Server engine.

DocumentDB: It is a fully managed NoSQL database service which is It built for fast and predictable performance and ease of development.

Content Delivery Network

Content Delivery Network (CDN) caches static web content at strategically placed locations. This helps you to offer speed for delivering content to users. It has the following components:

VPN Gateway: VPN Gateway sends encrypted traffic across a public connection.

Traffic Manager: It helps you to control and allows you to do the distribution of user traffic for services like WebApps, VM, Azure, and cloud services in different Datacenters

Enterprise Integration Services:

Service Bus: Service Bus is an information delivery service which works on the third-party communication system.

SQL Server Stretch Database: This service helps you migrates any cold data securely and transparently to the Microsoft Azure cloud

Monitoring + Management Services

These services allow easy management of Azure deployment.

Azure Resource Manager: It makes it easy for you to manage and visualize resource in your app. You can even control who is your organization can act on the resources.

Azure Networking

Virtual Network: Perform Network isolation and segmentation. It offers filter and Route network traffic.

Load Balancer: Offers high availability and network performance of any application. Load balance information Internet traffic to Virtual machines.

Application Gateway: It is a dedicated virtual appliance that offers an Application Delivery Controller (ADC) as a service.

Web and Mobile Services

Web Apps: Web Apps allows you to build and host websites in the programming language of your choice without the need to manage its infrastructure.
Mobile Apps: Mobile Apps Service offers a highly scalable, globally available mobile app development platform for users.
API Apps: API apps make it easier to develop, host and consume APIs in the cloud and on-premises.

Workflows in the cloud

It provides a visual designer to create and automate your process as a series of steps known as a workflow.

Notification Hubs: Azure Notification Hubs offers an easy-to-use, multi-platform, scaled-out push engine.

Event Hubs: Azure Event Hubs is data streaming platform which can manage millions of events per second. Data sent to an event hub can be transformed and stored using any real-time analytics offers batching/storage adapters.

Eucalyptus

Eucalyptus is a product from Eucalyptus Systems (www.eucalyptus.com) that was developed out of a research project at the University of California, Santa Barbara. Eucalyptus was initially aimed at bringing the cloud computing paradigm to academic supercomputers and clusters.

Eucalyptus provides an AWS-compliant EC2-based web service interface for interacting with the cloud service.

Additionally, Eucalyptus provides services, such as the AWS-compliant Walrus, and a user interface for managing users and images.

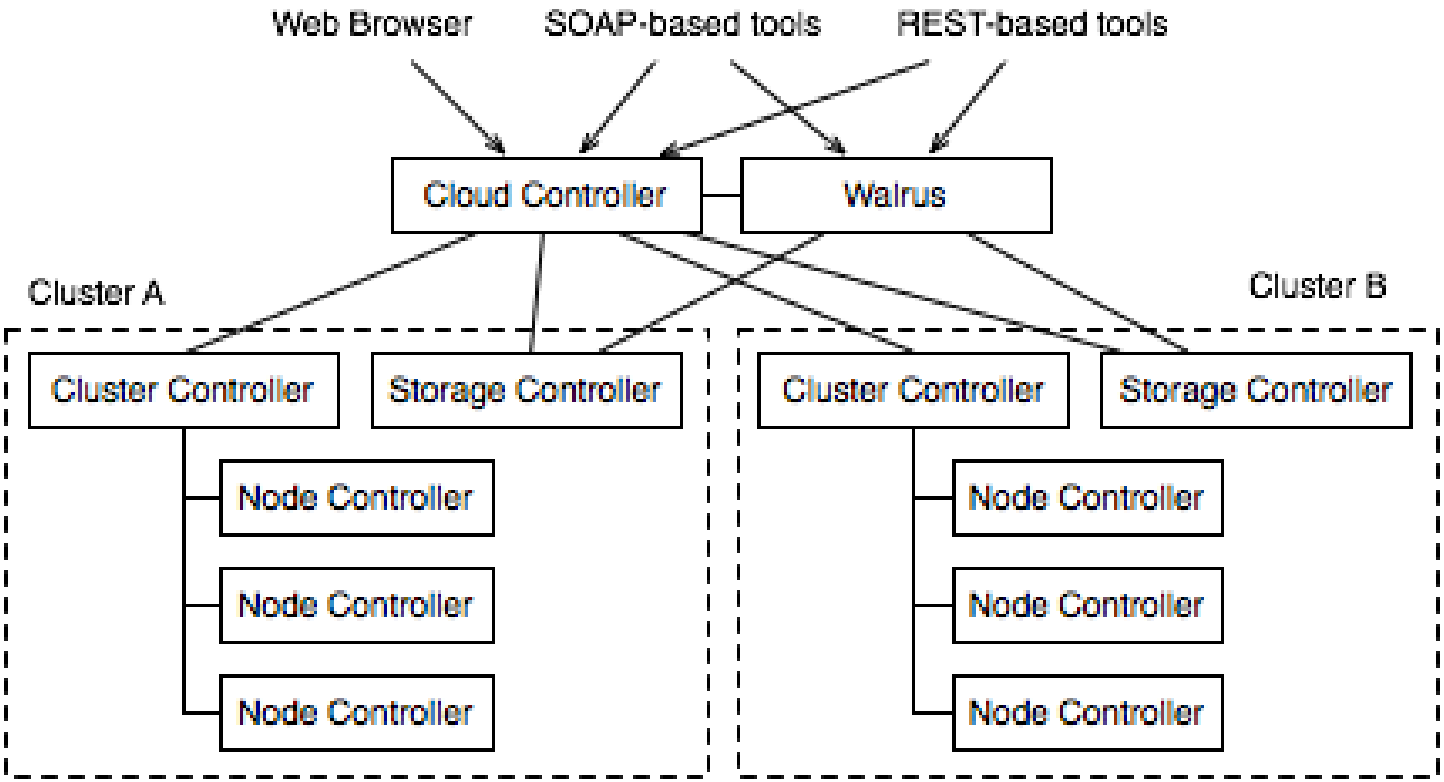
Eucalyptus is an open source Linux based software architecture which provides an EC2-compatible cloud computing platform and S3-compatible cloud storage platform. It implements scalable, efficient-enhancing and private and hybrid clouds within and organization's IT infrastructure. It gives an Infrastructure as a Service (IaaS) solution. Users can use commodity hardware.

Eucalyptus was developed to support the high performance computing (HPC). Eucalyptus can be deployed without modification on all major Linux OS distributions, including Ubuntu, RHEL/CentOS, openSUSE, and Debian.

Eucalyptus Features

For implementing, managing and maintaining the virtual machines, network and storage Eucalyptus has variety of features.

- SSH Key Management
- Image Management
- Linux-based VM Management
- IP Address Management
- Security Group Management
- Volume and Snapshot Management


Web Browser
SOAP-based tools
REST-based tools
Cloud Controller
Walrus
Cluster A
Cluster B
Cluster Controller
Storage Controller
Node Controller
Node Controller
Node Controller
Cluster Controller
Storage Controller
Node Controller
Node Controller
Node Controller

Components of Eucalyptus:

1. **Cluster Controller (CC)** Cluster Controller manages the one or more Node controller and responsible for deploying and managing instances on them. It communicates with Node Controller and Cloud Controller simultaneously. CC also manages the networking for the running instances under certain types of networking modes available in Eucalyptus.

2. Cloud Controller (CLC) Cloud Controller is front end for the entire ecosystem. CLC provides an Amazon EC2/S3 compliant web services interface to the client tools on one side and interacts with the rest of the components of the Eucalyptus infrastructure on the other side.

3. Node Controller (NC) It is the basic component for Nodes. Node controller maintains the life cycle of the instances running on each nodes. Node Controller interacts with the OS, hypervisor and the Cluster Controller simultaneously.

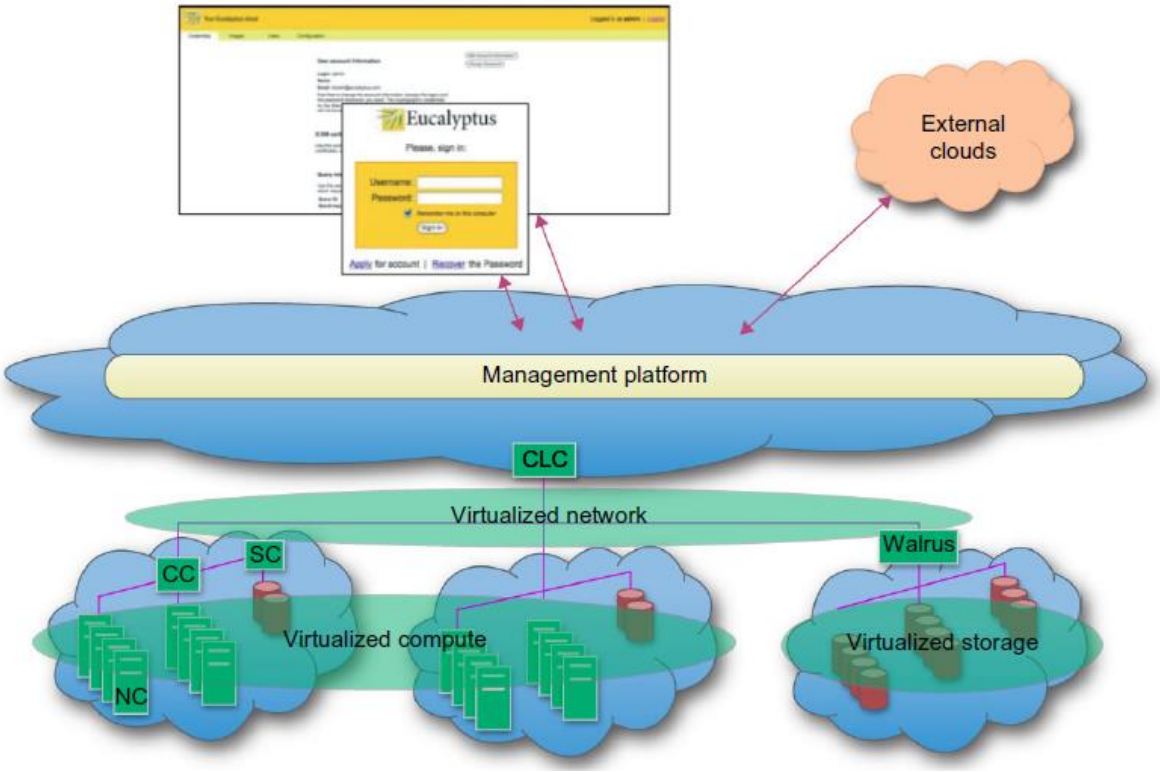
4. Walrus Storage Controller (WS3) Walrus Storage Controller is a simple file storage system. WS3 stores the the machine images and snapshots. It also stores and serves files using S3 APIs.

5. Storage Controller (SC) Allows the creation of snapshots of volumes. It provides persistent block storage over AoE or iSCSI to the instances.

VM Image Management

Eucalyptus takes many design queues from Amazon's EC2, and its image management system is no different. Eucalyptus stores images in Walrus, the block storage system that is analogous to the Amazon S3 service. As

As such, any user can bundle her own root file system, and upload and then register this image and link it with a particular kernel and ramdisk image. This image is uploaded into a user-defined bucket within Walrus, and can be retrieved anytime from any availability zone



Nimbus is a powerful toolkit focused on converting a computer cluster into an Infrastructure-as-a-Service (IaaS) cloud for scientific communities. Essentially, it allows a deployment and configuration of virtual machines (VMs) on remote resources to create an environment suitable for the users' requirements. Being written in Python and Java, it is totally free and open-source software, released under the Apache License.

Nimbus consists of two basic products:

- Nimbus Infrastructure is an open source EC2/S3-compatible IaaS solution with features that benefit scientific community interests, like support for auto-configuring clusters, proxy credentials, batch schedulers, best-effort allocations, etc.

- Nimbus Platform is an integrated set of tools for a multi-cloud environment that automates and simplifies the work with infrastructure clouds (deployment, scaling, and management of cloud resources) for scientific users.

Nimbus Goals

Allow providers to build clouds

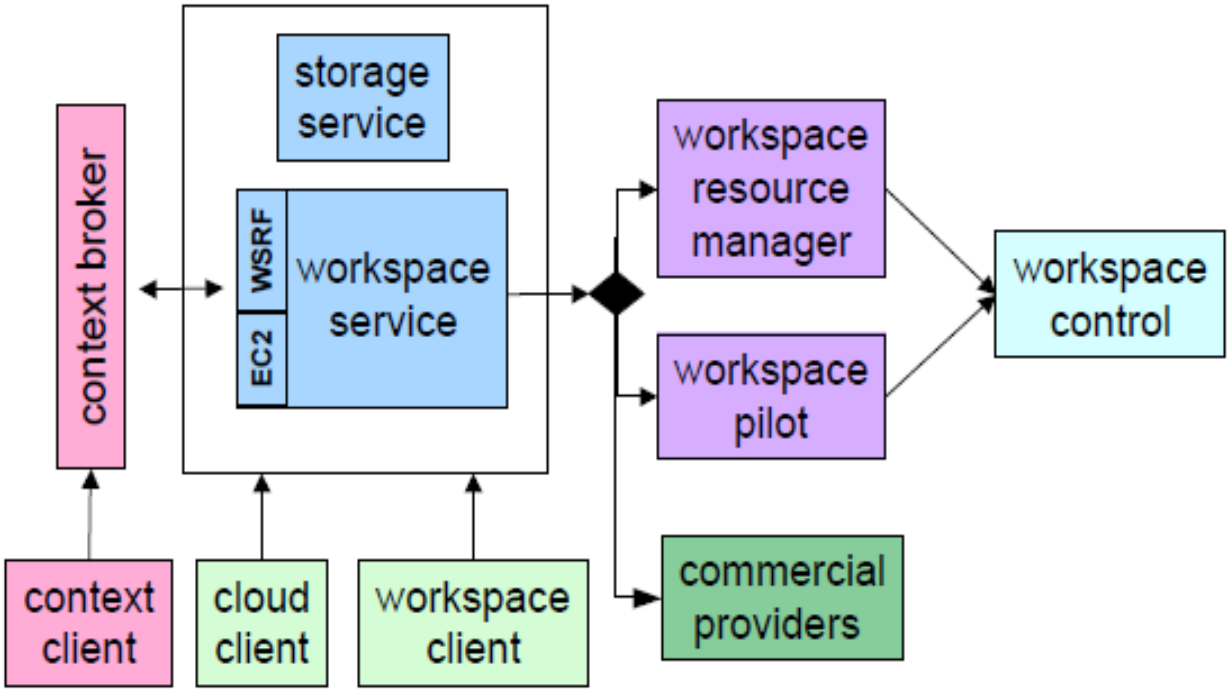
- Private clouds (privacy, expense considerations)
- Workspace Service: open source EC2 implementation

Allow users to use cloud computing

- Do whatever it takes to enable scientists to use IaaS
- Context Broker: turnkey virtual clusters IaaS Gateway: interoperability

Allow developers to experiment with Nimbus

- For research or usability/performance improvements
- Community extensions and contributions


storage service
WSRF
EC2
workspace service
context broker
workspace resource manager
workspace pilot
workspace control
commercial providers
context client
cloud client
workspace client

1. Workspace service: Allows clients to manage and administer VMs by providing to two interfaces; One interface is based on the web service resource framework (WSRF) and the other is based on EC2 WSDL. This service communicates with a workspace resource manager or a workspace pilot to manage instances.

2.Workspace resource manager: Implements VM instance creation on a site and management.

3. Workspace pilot: provides virtualization with significant changes to the site configurations.

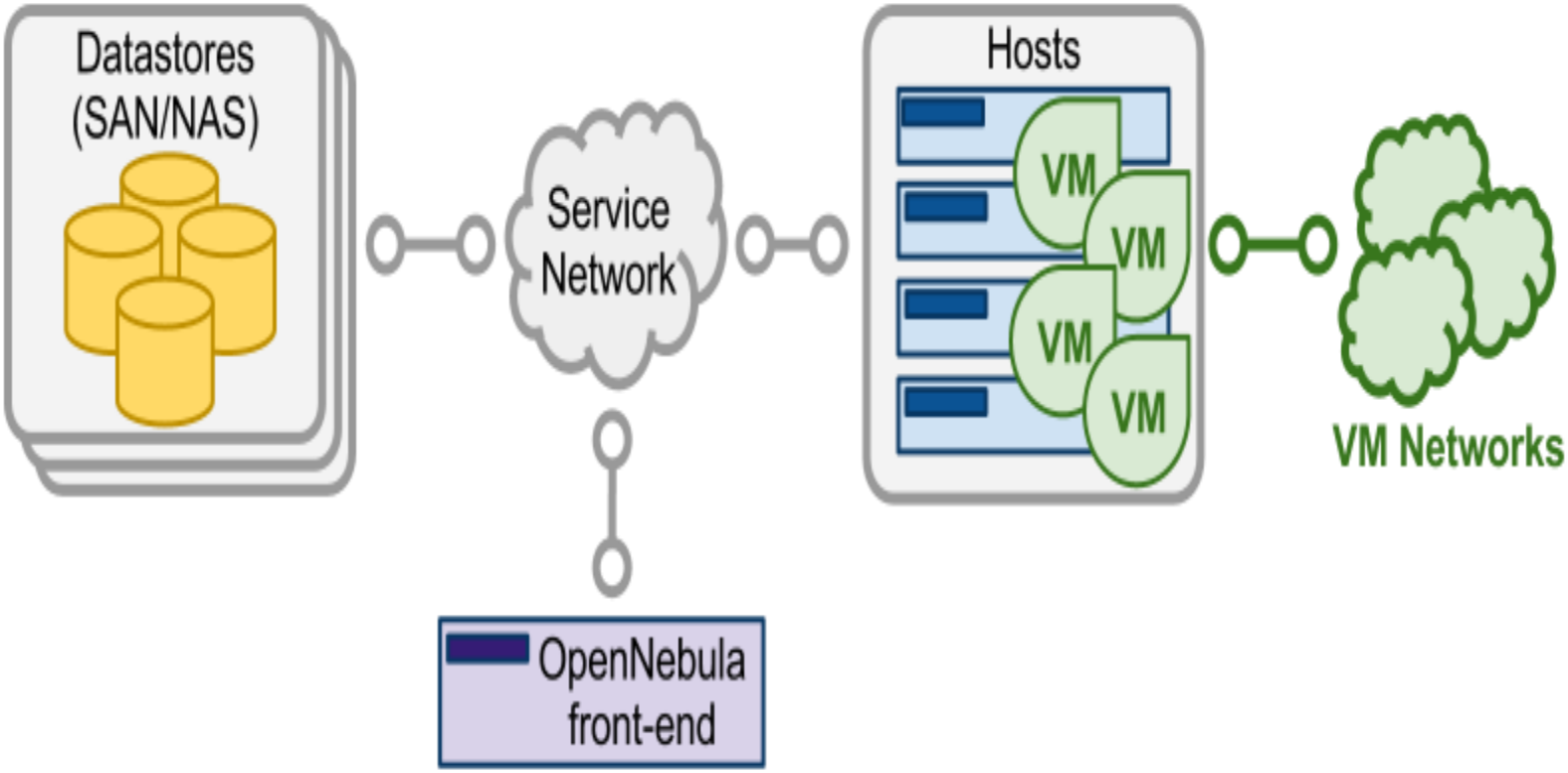
4. Workspace control: implements VM instance management such as start, stop and pause VM. It also provides image management and sets up networks and provides IP assignment.

5. Context broker: allows clients coordinate large virtual cluster launches automatically and repeatedly.

6. Workspace client: a complex client that provides full access to the workspace service functionality.

7. Cloud client: a simpler client providing access to selected functionalities in the workspace service.

OpenNebula is a simple but feature-rich and flexible solution to build and manage enterprise clouds and virtualized DCs, that combines existing virtualization technologies with advanced features for multi-tenancy, automatic provision and elasticity. OpenNebula follows a bottom-up approach driven by sysadmins, devops and users real needs.



A cloud architecture is defined by three components: storage, networking and virtualization. Therefore, the basic components of an OpenNebula system are:

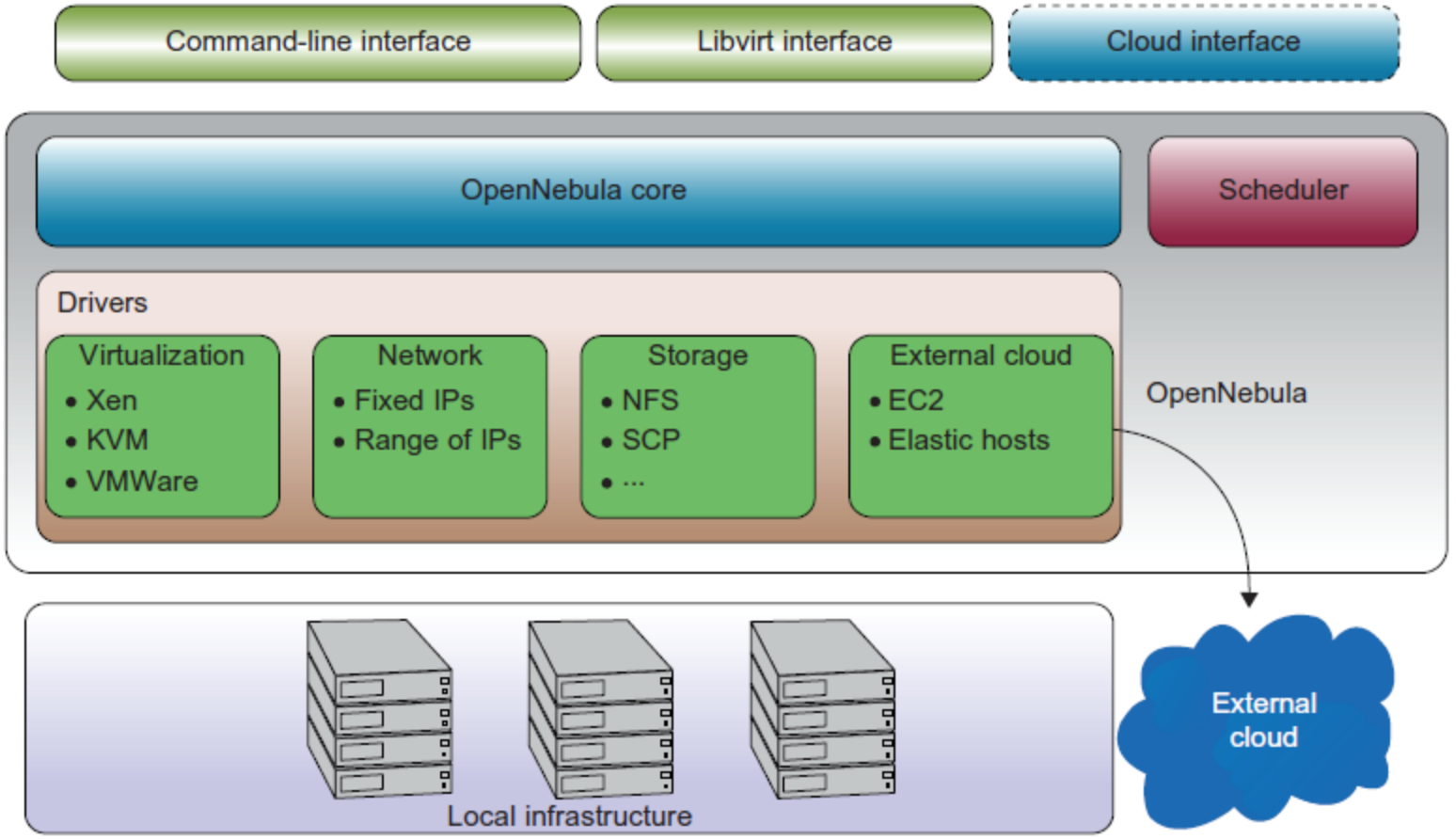
- **Front-end** that executes the OpenNebula services.
- Hypervisor-enabled **hosts** that provide the resources needed by the VMs.
- **Datastores** that hold the base images of the VMs.
- Physical **networks** used to support basic services such as interconnection of the storage servers and OpenNebula control operations, and VLANs for the VMs.

OpenNebula presents a highly modular architecture that offers broad support for commodity and enterprise-grade hypervisor, monitoring, storage, networking and user management services.

Front-End

The machine that holds the OpenNebula installation is called the front-end. This machine needs network connectivity to all the hosts, and possibly access to the storage Datastores (either by direct mount or network). The base installation of OpenNebula takes less than 150MB.

Figure shows the OpenNebula architecture and its main components.



- The architecture of OpenNebula has been designed to be flexible and modular to allow integration with different storage and network infrastructure configurations, and hypervisor technologies.

- Here, the core is a centralized component that manages the VM full life cycle, including setting up networks dynamically for groups of VMs and managing their storage requirements, such as VM disk image deployment or on-the-fly software environment creation.

- Important component is the capacity manager or scheduler. It governs the functionality provided by the core. The default capacity scheduler is a requirement/rank matchmaker. However, it is also possible to develop more complex scheduling policies, through a lease model and advance reservations

- The last main components are the access drivers. They provide an abstraction of the underlying infrastructure to expose the basic functionality of the monitoring, storage, and virtualization services available in the cluster.

- Therefore, OpenNebula is not tied to any specific environment and can provide a uniform management layer regardless of the virtualization platform.

- OpenNebula implements the libvirt API , an open interface for VM management, as well as a command-line interface (CLI).

- A subset of this functionality is exposed to external users through a cloud interface. OpenNebula is able to adapt to organizations with changing resource needs, including addition or failure of physical resources . Some essential features to support changing environments are live migration and VM snapshots

- when the local resources are insufficient, OpenNebula can support a hybrid cloud model by using cloud drivers to interface with external clouds.

- Their local infrastructure with computing capacity from a public cloud to meet peak demands, or implement HA strategies. OpenNebula currently includes an EC2 driver, which can submit requests to Amazon EC2 and Eucalyptus, as well as an Elastic Hosts driver .