

UNIT IV

CLOUD PROGRAMMING AND SOFTWARE ENVIRONMENTS

Features of Cloud and Grid Platforms, Parallel & Distributed Programming Paradigms, Programming Support of Google App Engine, Programming on Amazon AWS and Microsoft Azure, Emerging Cloud Software Environments.

FEATURES OF CLOUD AND GRID PLATFORMS

Cloud Capabilities and Platform Features

Commercial clouds need broad capabilities; these capabilities offer cost-effective utility computing with the elasticity to scale up and down in power. However, as well as this key distinguishing feature, commercial clouds offer a growing number of additional capabilities commonly termed “Platform as a Service” (PaaS). For Azure, current platform features include Azure Table, queues, blobs, Database SQL, and web and Worker roles. Amazon is often viewed as offering “just” Infrastructure as a Service (IaaS), but it continues to add platform features including SimpleDB (similar to Azure Table), queues, notification, monitoring, content delivery network, relational database, and MapReduce (Hadoop). Google does not currently offer a broad-based cloud.

Traditional Features Common to Grids and Clouds

Here we discuss about features related to workflow, data transport, security, and availability concerns that are common to today’s computing grids and clouds.

- **Workflow :** workflow has spawned many projects in the United States and Europe. Pegasus, Taverna, and Kepler are popular, but no choice has gained wide acceptance. A recent entry is Trident from Microsoft Research which is built on top of Windows Workflow Foundation. If Trident runs on Azure or just any old Windows machine, it will run workflow proxy services on external (Linux) environments. Workflow links multiple cloud and non cloud services in real applications on demand.
- **Data Transport :** The cost (in time and money) of data transport in (and to a lesser extent, out of) commercial clouds is often discussed as a difficulty in using clouds. If commercial clouds become an important component of

- **Security, Privacy, and Availability:** The following techniques are related to security, privacy, and availability requirements for developing a healthy and dependable cloud programming environment.
- Use virtual clustering to achieve dynamic resource provisioning with minimum overhead cost.
- Use stable and persistent data storage with fast queries for information retrieval.
- Use special APIs for authenticating users and sending e-mail using commercial accounts.
- Cloud resources are accessed with security protocols such as HTTPS and SSL.
- Fine-grained access control is desired to protect data integrity and deter intruders or hackers.
- Shared data sets are protected from malicious alteration, deletion, or copyright violations.
- Features are included for availability enhancement and disaster recovery with life migration of VMs.
- Use a reputation system to protect data centers. This system only authorizes trusted clients and stops pirates.

Important cloud platform capabilities	
Capability	description
Massive database storage service	Large disk capacity of cloud data storage services are provided
Massive data processing method and programming method	Programmers need to harness the machine power without considering the massive infrastructure management issues
Programming interface and service deployment	Cloud applications can use Ajax technologies in order to improve experience while web browsing
Physical or virtual computing platform	Virtual platform has unique capabilities to provide isolated environments

Infrastructure cloud features	
Operating systems	Windows, Apple, Linux, Android
Program library	Material and other images are being stored
Accounting	Economics activities are included
Registry	System resource information
Scheduling	Platform, basic staple of condor
Authentication and authorization	Need single signs for multiple systems
Virtualization	Cloud supports the basic feature of elasticity
Gang scheduling	Assigns multiple tasks in a scalable fashion

Traditional Features in Cluster, Grid and Parallel Computing Enviornments	
Portals	Called as gateways which observes changes in technologies
OpenMP/threading	It includes parallel compilers
Data management	Includes metadata supports
Cluster management	Includes packages offering different tools
Scalable parallel computing environments	High level MPI are associated with it
workflow	Job components supporting workflows are linked
Virtual organization	Grid solutions are included

Programs features supported by grids and clouds	
DPFS	Supports files systems in Google
Monitoring	Includes many grid solutions
Programming models	Models are built on other platforms
MapReduce	Supports map reducing programming functions
Notifications	Includes basic functions of subscription of systems
Blob	Includes basic storage concepts
Fault tolerance	Includes grids which are largely ignored
Queues	Includes queuing systems
Worker role	Uses implicit both in grids and Amazon
Table	Supports structures modeled of table data in Amazon
SQL	Includes relational databases
Web role	Describe Azure related important link

Data Features and Databases : they describe programming features related to the program library, blobs, drives, DPFS, tables, and various types of databases including SQL, NOSQL, and non relational databases and special queuing services.

- **Program Library:** A VM image library is designed to manage images used in academic and commercial clouds. The basic cloud environments description includes many management features allowing convenient deployment and configuring of images.
- **Blobs and Drives :** The basic storage concept in clouds is blobs for Azure and S3 for Amazon. These can be organized (approximately, as in directories) by containers in Azure. In addition to a service interface for blobs and S3, one can attach “directly” to compute instances as Azure drives and the Elastic Block Store for Amazon. This concept is similar to shared file systems such as Lustre used in TeraGrid.
- **DPFS :** DPFS file systems are precisely designed for efficient execution of data-intensive applications. However, the importance of DPFS for linkage with Amazon and Azure is not clear, as these clouds do not currently offer fine-grained support for compute-data affinity.

- **SQL and Relational Databases** : Both Amazon and Azure clouds offer relational databases and it is straightforward for academic systems to offer a similar capability unless there are issues of huge scale where, in fact, approaches based on tables and/or Map Reduce might be more appropriate.
- **Table and NOSQL Non-relational Databases**: A substantial number of important developments have occurred regarding simplified database structures—termed “NOSQL” typically emphasizes distribution and scalability. These are present in the three major clouds: BigTable ,in Google, SimpleDB , in Amazon, and Azure Table for Azure. Tables are clearly important in science as illustrated by the VOTable standard in astronomy and the popularity of Excel.
- Non relational databases, especially in terms of triple stores for metadata storage and access. The current cloud tables fall into two groups: Azure Table and Amazon SimpleDB are quite similar and support lightweight storage for “document stores,” while BigTable aims to manage large distributed data sets without size limitations.
- **Queuing Services** : Both Amazon and Azure offer similar scalable, robust queuing services that are used to communicate between the components of an application. The messages are short (less than 8 KB) and have a Representational State Transfer (REST) service interface with “deliver at least once” semantics. They are controlled by timeouts for posting the length of time allowed for a client to process.

Programming and Runtime Support: Programming and runtime support are desired to facilitate parallel programming and provide runtime support of important functions in today’s grids and clouds.

- **Worker and Web Roles** : The roles introduced by Azure provide nontrivial functionality, while preserving the better affinity support that is possible in a non virtualized environment. **Worker roles** are basic schedulable processes and are automatically launched. explicit scheduling is unnecessary in clouds for individual worker roles and for the “gang scheduling” supported transparently in Map Reduce. Web roles provide an interesting approach to portals. GAE is largely aimed at web applications, whereas science gateways are successful in Tera Grid.

- **Map Reduce** : Map Reduce, summarized in below table, has several advantages over traditional implementations for many task problems, as it supports dynamic execution, strong fault tolerance, and an easy-to-use high-level interface. The major open source/commercial Map Reduce implementations are Hadoop and Dryad with execution possible with or without VMs.

Comparison of MapReduce type systems					
Content	Google MapReduce	Apache hadoop	Microsoft Dryad	Twister	Azure Twister
Scheduling	Data locality	Data locality	Data locality	Data locality	Dynamic task
Programming model	MapReduce	MapReduce	DAG execution	Iterative MapReduce	MapReduce
Data Handling	GFS	HDFS	Shared directories	Local disks	Azure blob storage
Failure handling	Re-execution of failure tasks	Re-execution of failure tasks	Re-execution of failure tasks	Re-execution of failure tasks	Re-execution of failure tasks
Environment	Linux cluster	Linux cluster	Windows	Linux cluster	Windows azure
HLL Support	Sawzall	Pig latin	DryadLINQ	Pregel	N/A
Intermediate data transfer	File	File HTTP	File TCP	Publish subscriber	Files , TCP

Cloud programming model: Both the GAE and Manjrasoft Aneka environments represent programming models; both are applied to clouds, but are really not specific to this architecture. Iterative MapReduce is an interesting programming model that offers portability between cloud, HPC and cluster environments.

SaaS : SaaS environment that provides many useful tools to develop cloud applications over large data sets. In addition to the technical features, such as MapReduce, BigTable, EC2, S3, Hadoop, AWS, GAE, and WebSphere2.

PARALLEL & DISTRIBUTED PROGRAMMING PARADIGMS

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. A computer cluster or network of workstations is an example of a distributed computing system..

Running a parallel program on a distributed computing system (parallel and distributed programming) has several advantages for both users and distributed computing systems. From the **users' perspective**,

- it decreases application response time;

From the **distributed computing systems**

- It increases throughput and resource utilization.

Parallel Computing and Programming Paradigms

Consider a 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.
- **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.
- **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..
- **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.

MapReduce, Twister, and Iterative 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.

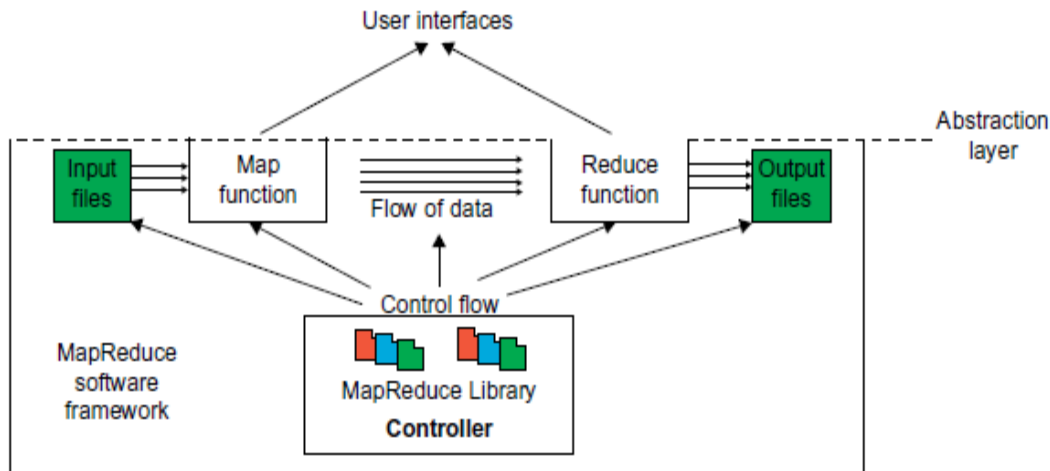


Fig: MapReduce framework

Formal Definition of MapReduce

The MapReduce software framework provides an abstraction layer with the data flow and flow of control to users, and hides the implementation of all data flow steps such as data partitioning, mapping, synchronization, communication, and scheduling. Here, although the data flow in such frameworks is predefined, the abstraction layer provides two well-defined interfaces in the form of two functions: Map and Reduce. The MapReduce function, `MapReduce (Spec, & Results)`, takes an important parameter which is a specification object, the Spec. 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)
}

```

MapReduce Logical Data Flow : The input data to both the Map and the Reduce functions has a particular structure. This also pertains for the output data. The input data to the Map function is in the form of a (key, value) pair. 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. 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]).

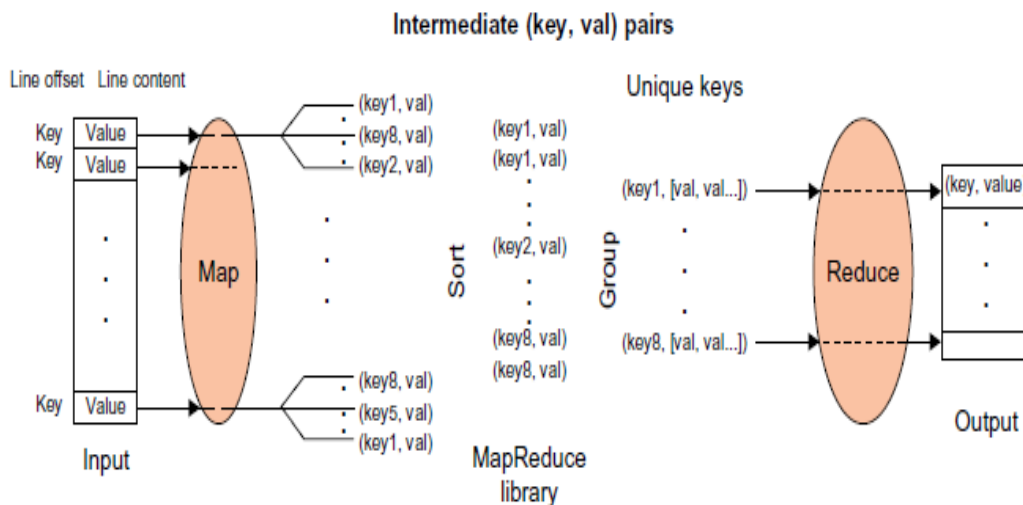
Formal Notation of MapReduce Data Flow: The Map function is applied in parallel to every input (key, value) pair, and produces new set of intermediate (key, value) pairs as follows:

$$(key_1, val_1) \xrightarrow{\text{Map Function}} \text{List}(key_2, val_2)$$

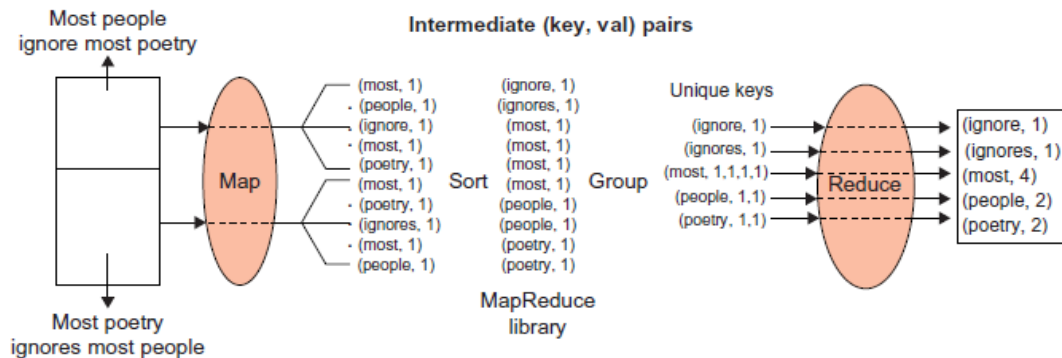
The Reduce function is applied in parallel to each group producing the collection of values as output as

$$(key_2, \text{List}(val_2)) \xrightarrow{\text{Reduce Function}} \text{List}(val_2)$$

Strategy to Solve MapReduce Problems : after grouping all the intermediate data, the values of all occurrences of the same key are sorted and grouped together. As a result, after grouping, each key becomes unique in all intermediate data. Therefore, finding unique keys is the starting point to solving a typical MapReduce problem.



MapReduce logical data flow in 5 processing stages over successive (key, value) pairs.



Map Reduce Actual Data and Control Flow: the following distinct steps:

1. **Data partitioning**
2. **computation portioning**
3. **determining the master and worker**
4. **reading the input data**
5. **Map function**
6. **combiner function**
7. **partitioning function**
8. **synchronization**
9. **communication**
10. **sorting and grouping**
11. **reduce function**

Twister and Iterative MapReduce

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

- Map Reduce 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 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.

These changes will lead to major performance increases at the cost of poorer fault tolerance and ease to support dynamic changes such as the number of available nodes.

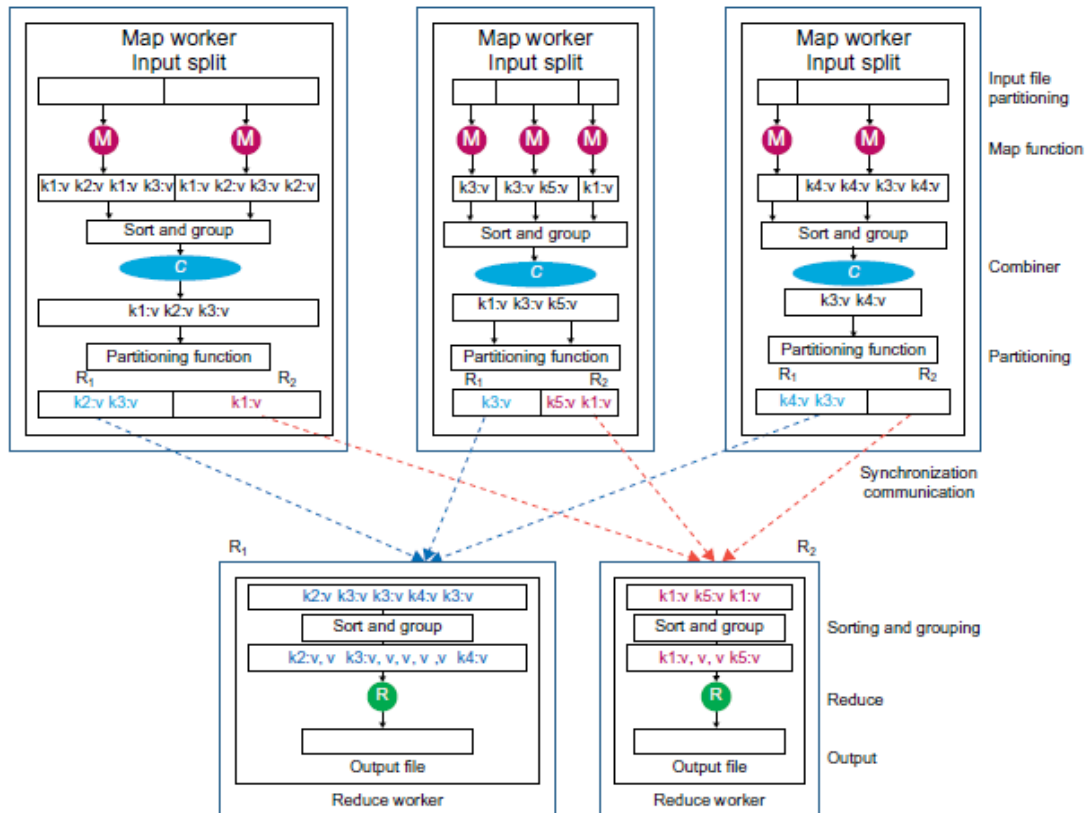


Fig: Data flow implementation of many functions in the Map workers and in the Reduce workers through multiple sequences of partitioning.

The Map-Reduce pair is iteratively executed in long-running threads

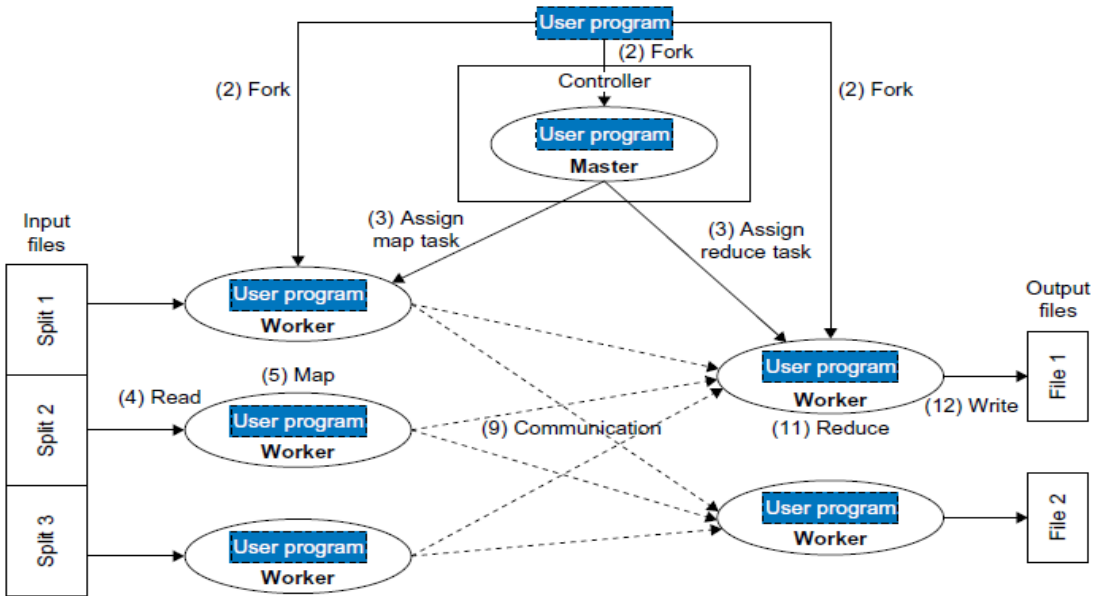
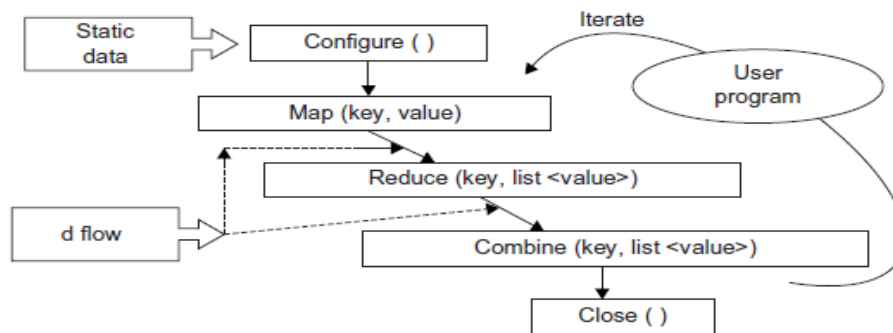
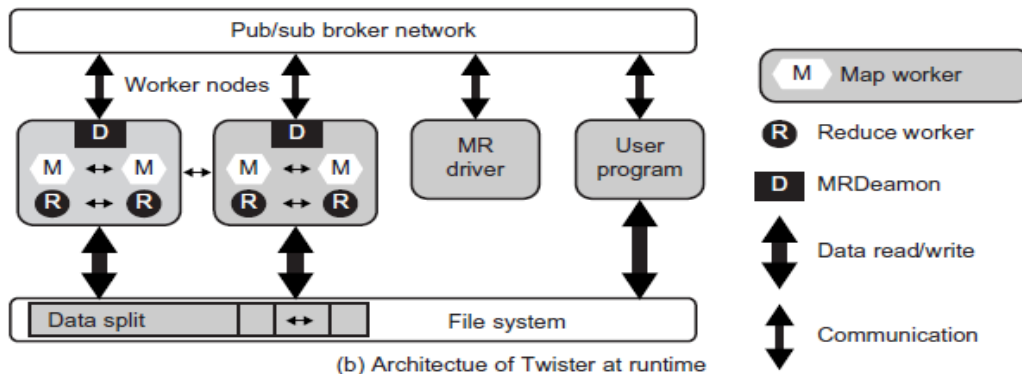


Fig: Control flow implementation of the MapReduce functionalities in Map workers and Reduce workers

- Twister is much faster than traditional MapReduce. Twister distinguishes the static data which is never reloaded from the dynamic δ flow that is communicated.



(a) Twister for iterative MapReduce programming



(b) Architecture of Twister at runtime

Fig: Twister: An iterative MapReduce programming paradigm for repeated MapReduce executions

The Map-Reduce pair is iteratively executed in long-running threads, the different thread and process structures of 4 parallel programming paradigms: namely Hadoop, Dryad, Twister (also called MapReduce++), and MPI.

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 layers.

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

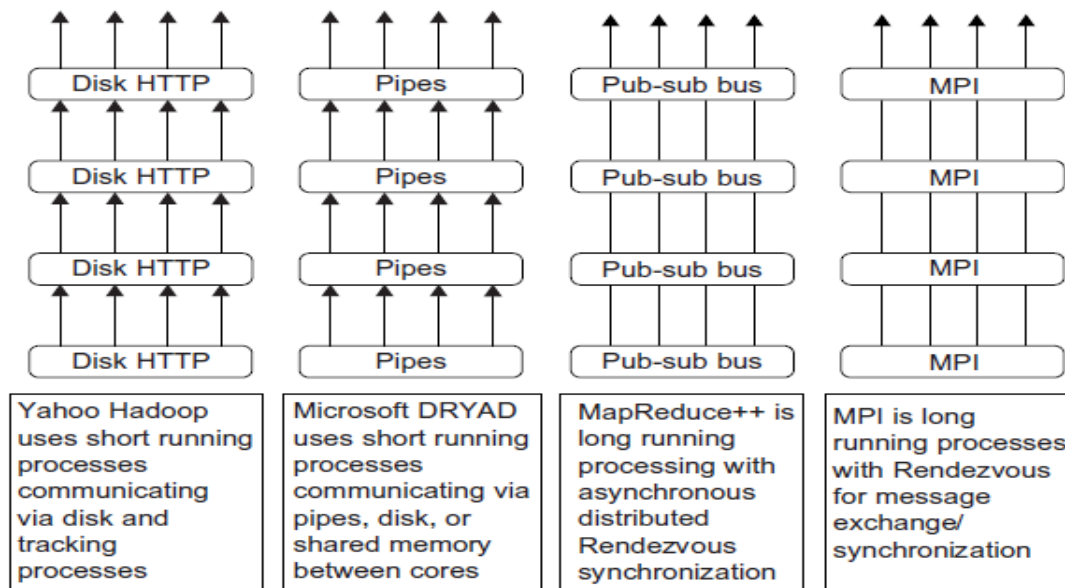


Fig: Thread and process structure of four parallel programming paradigms at runtimeS.

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.

- The mapping of blocks to DataNodes is determined by the NameNode. The NameNode (master) also manages the file system's metadata and namespace.

HDFS Features: Distributed file systems have special requirements, such as performance, scalability, concurrency control, fault tolerance, and security requirements to operate efficiently.

- 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.

HDFS Fault Tolerance: One of the main aspects of HDFS is its fault tolerance characteristic.

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.
- **Replica placement** The placement of replicas is another factor to fulfill the desired fault tolerance in HDFS.
- **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.

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.

- **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.
- **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.

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 map

reduce engine also has a master/slave architecture consisting of a single JobTracker as the master and a number of TaskTrackers as the slaves (workers).

- 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.

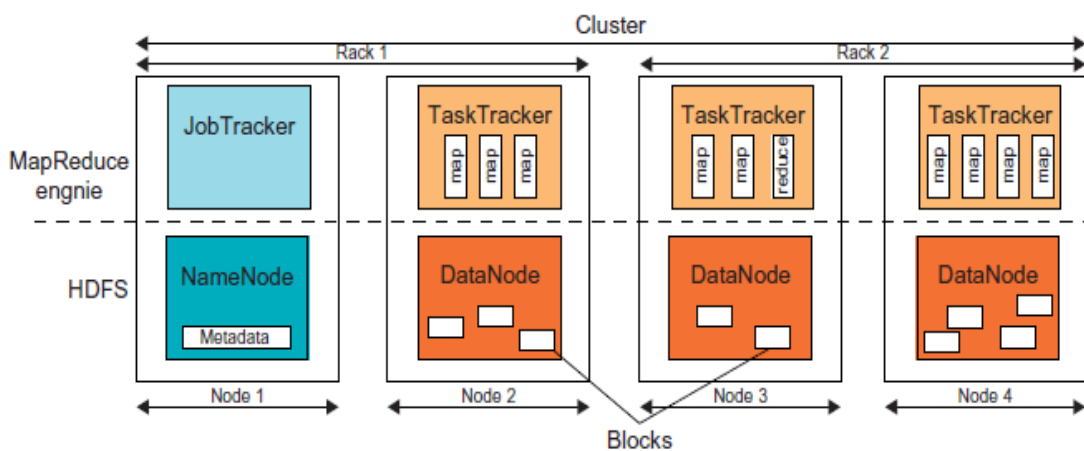


Fig :HDFS and MapReduce architecture in Hadoop

Running a Job in Hadoop

Three components contribute in running a job in this system: a user node, a JobTracker, and several TaskTrackers.

- **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.
- **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.
- **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.

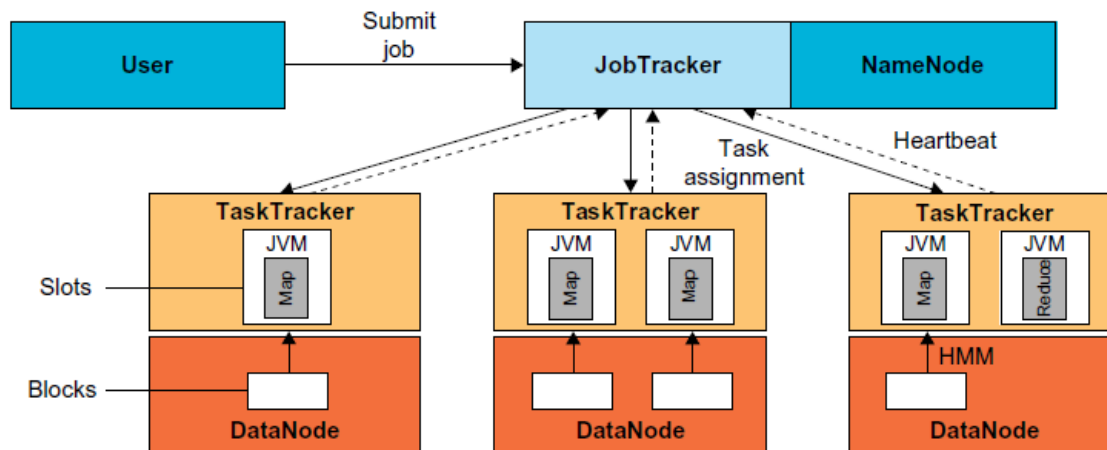


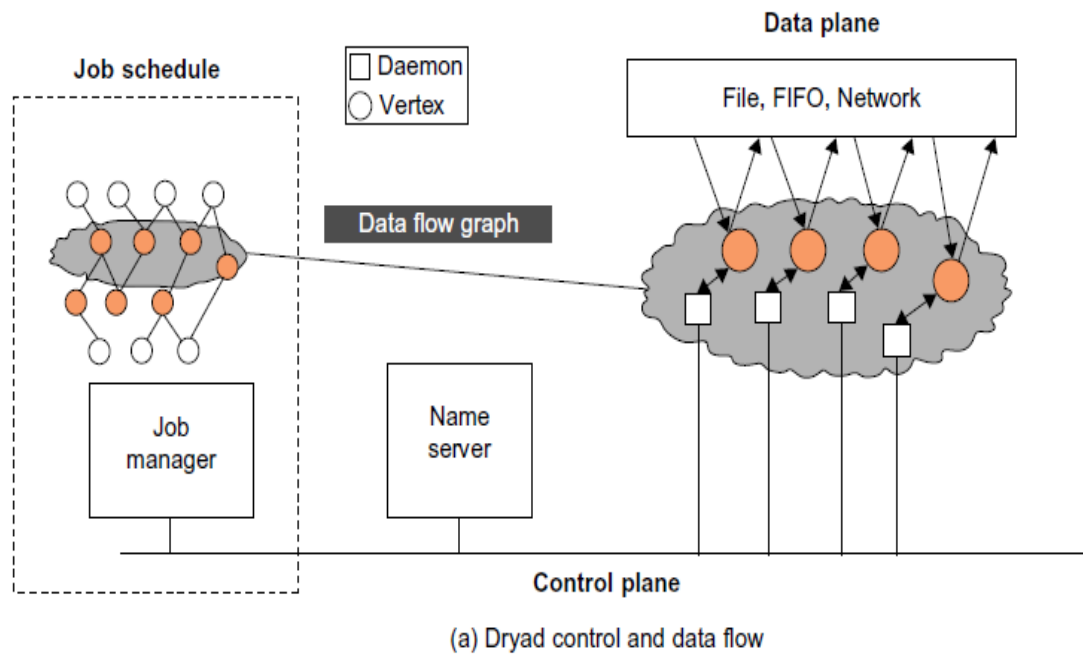
fig: Data flow in running a MapReduce job at various task trackers using the Hadoop library
Task running check A task running check is performed by receiving periodic heartbeat messages to the JobTracker from the TaskTrackers.

Comparison of MapReduce type systems					
Content	Google MapReduce	Apache hadoop	Microsoft Dryad	Twister	Azure Twister
Scheduling	Data locality	Data locality	Data locality	Data locality	Dynamic task
Programming model	MapReduce	MapReduce	DAG execution	Iterative MapReduce	MapReduce
Data Handling	GFS	HDFS	Shared directories	Local disks	Azure blob storage
Failure handling	Re-execution of failure tasks	Re-execution of failure tasks	Re-execution of failure tasks	Re-execution of failure tasks	Re-execution of failure tasks
Environment	Linux cluster	Linux cluster	Windows	Linux cluster	Windows azure
HLL Support	Sawzall	Pig latin	DryadLINQ	Pregel	N/A
Intermediate data transfer	File	File HTTP	File TCP	Publish subscriber	Files , TCP

Dryad and DryadLINQ from Microsoft : Dryad and DryadLINQ, both developed by Microsoft.

- Dryad is more flexible than MapReduce as the data flow of its applications is not dictated/pre determined and can be easily defined by users. To achieve such flexibility, a Dryad program or job is defined by a directed acyclic graph (DAG) where vertices are computation engines and edges are communication channels between vertices.

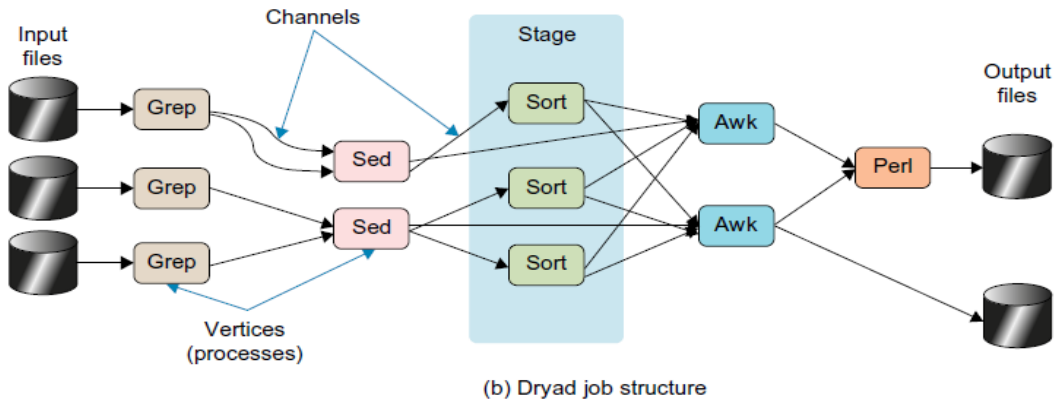
- The two main components handling the control flow of Dryad are the job manager and the name server.



- A Dryad job is controlled by the job manager, which is responsible for deploying the program to the multiple nodes in the cluster. It runs either within the computing cluster or as a process in the user's workstation which can access the cluster.

The job manager has the code to construct the DAG as well as the library to schedule the work running on top of the available resources. The job manager is able to:

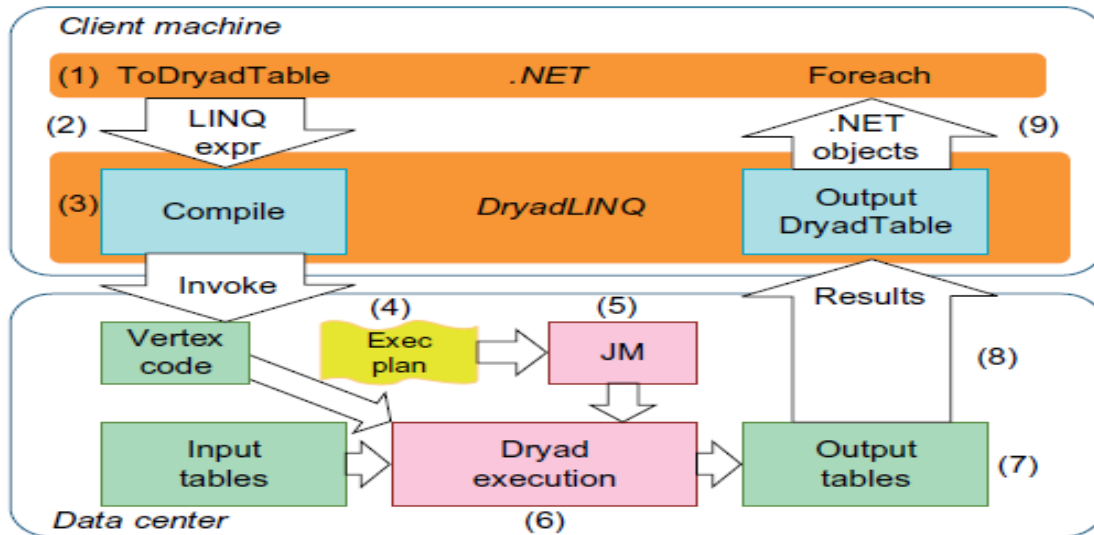
1. Map the data flow graph to the underlying resources.
2. Schedule all necessary communications and synchronization across the respective resources



- The Dryad 2D pipe job structure. During 2D pipe execution, Dryad defines many operations to construct and change the DAG dynamically.
- The operations include creating new vertices, adding graph edges, merging two graphs, as well as handling job input and output.
- Dryad also has a fault-tolerant mechanism built in. As it is built on a DAG, there are typically two types of failures: vertex failures and channel failures, which are handled differently.

DryadLINQ from Microsoft

- DryadLINQ is built on top of Microsoft's Dryad execution framework.
- Dryad can perform acyclic task scheduling and run on large-scale servers. The goal of DryadLINQ is to make large-scale, distributed cluster computing available to ordinary programmers.
- The Dryad distributed execution engine and .NET Language Integrated Query (LINQ). LINQ is particularly for users familiar with a database programming model.



The execution is divided into nine steps as follows:

1. A .NET user application runs, and creates a DryadLINQ expression object.
2. The application calls `ToDryadTable` triggering a data-parallel execution.
3. DryadLINQ compiles the LINQ expression into a distributed Dryad execution plan.
4. DryadLINQ invokes a custom Dryad job manager which is used to manage and monitor the execution flow of the corresponding task.
5. The job manager creates the job graph using the plan created in step 3.
6. Each Dryad vertex executes a vertex-specific program
7. When the Dryad job completes successfully it writes the data to the out table
8. The job manager process terminates, and it returns control back to DryadLINQ.
9. Control returns to the user application

Sawzall and Pig Latin High-Level Languages

- Sawzall was developed by Rob Pike with an initial goal of processing Google's log files.
- Sawzall is a high-level language built on top of Google's MapReduce framework. Sawzall is a scripting language that can do parallel data processing.
- Sawzall can do distributed, fault-tolerant processing of very large-scale data sets, even at the scale of the data collected from the entire Internet.

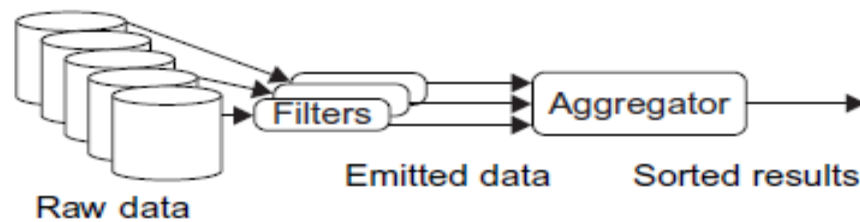


Fig: the overall flow of filtering, aggregating, and collating in Sawzall.

- The Sawzall runtime engine translates the corresponding scripts to MapReduce programs running on many nodes. The Sawzall program can harness the power of cluster computing automatically as well as attain reliability from redundant servers.

Pig Latin : Pig Latin is a high-level data flow language developed by Yahoo! it has been implemented on top of Hadoop in the Apache Pig project. Pig Latin, Sawzall and DryadLINQ are different approaches to building languages on top of MapReduce and its extensions.

Comparison of high level data analysis languages			
Content	Sawzall	DryadLINQ	Pig Latin
Target Runtime	Google MapReduce	Dryad	Hadoop
Typing	Static	Static	dynamic
Programming style	Imperative	Imperative and declarative	procedural
Category	Interpreted	Compiled	Compiled
Data model	Google protocol buffer	Partition file	yahoo

Pig Latin Data Types		
Data type	Description	example
Map	Collection of data items related to set of keys	['Microsoft' -> { ('windows') ('Azure') } 'Redhat' -> 'Linux']
Tuple	Includes sequence of fields of any Pig Latin Type	('clouds', 'Grods')
Atom	Includes a simple atomic values	'Clouds'
Bag	Includes collection of tuples which each member has a different schema	{ ('Clouds', 'Grids') ('Clouds', 'Iaas', 'Paas') }

Pig Latin Operators	
Command	Description
STORE	Data is written in the file systems
JOIN	Based on keys two or more inputs are being joined
UNION	Merges two or more data sets
SPLIT	Data splitted into two or more sets on filter conditions
FILTER	To predict and remove records which are not returning true
CROSS	Two or more products are crossed

FOREACH GENERATE	For each output and record an expression is applied
-------------------------	---

Mapping Applications to Parallel and Distributed Systems: In the past, Fox has discussed mapping applications to different hardware and software in terms of five application architectures. These initial five categories are listed in [Table](#) , followed by a sixth emerging category to describe data-intensive computing. The original classifications largely described simulations and were not aimed directly at data analysis.

Category 1 corresponds to regular problems, whereas *category 2* includes dynamic irregular cases with complex geometries for solving partial differential equations or particle dynamics, *Category 3* consists of asynchronously interacting objects and is often considered the people’s view of a typical parallel problem. It probably does describe the concurrent threads in a modern operating system, as well as some important applications, such as event-driven simulations and areas such as search in computer games and graph algorithms. *Category 4* is the simplest algorithmically, with disconnected parallel components. *Category 5* refers to the coarse-grained linkage of different “atomic” problems

Category	Class	Description	Machine architecture
1	synchronous	Problem is implemented with instruction level	SIMD
2	Asynchronous	Described with computation and inter programming	Shared memory
3	Loosely synchronous	They exhibit iterative compute communication stages	MIMD on MPP
4	Meta problems	They are finite combination of different categories	Grids of clusters
5	MapReduce++	They describes file to file operation	MapReduce

PROGRAMMING SUPPORT OF GOOGLE APP ENGINE

The programming support model for GAE are discussed here are

Programming the Google App Engine

Google File System (GFS)

Big Table, Google’s NOSQL System

Chubby, Google’s Distributed Lock Service

PROGRAMMING THE GOOGLE APP ENGINE

The below fig. summarizes some key features of GAE programming model for two supported languages: **Java and Python**, client environment that includes an Eclipse plug-in for Java allows us to debug your **GAE** on once local machine.

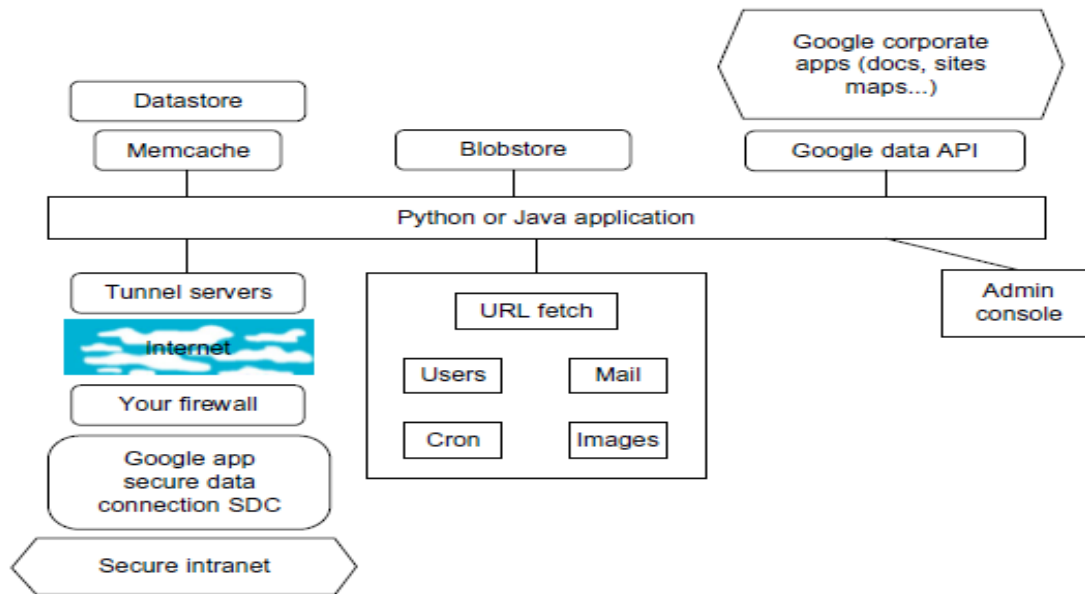


Fig: Programming environment for Google AppEngine.

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, in the above fig. the components are described as

- The **blobstore** which is suitable for large files as its size limit is 2 GB.
- The Google **SDC Secure Data Connection** can tunnel through the Internet and link your intranet to an external GAE application.
- 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,
- 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.

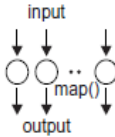
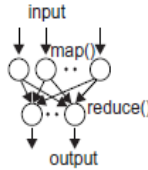
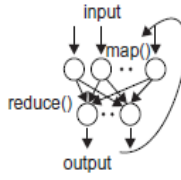
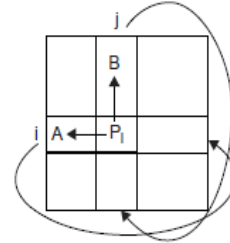
Table 6.11 Comparison of MapReduce++ Subcategories along with the Loosely Synchronous Category Used in MPI			
Map-Only	Classic MapReduce	Iterative MapReduce	Loosely Synchronous
 <ul style="list-style-type: none"> Document conversion (e.g., PDF->HTML) Brute force searches in cryptography Parametric sweeps Gene assembly PolarGrid Matlab data analysis (www.polargrid.org) 	 <ul style="list-style-type: none"> High-energy physics (HEP) histograms Distributed search Distributed sort Information retrieval Calculation of pairwise distances for sequences (BLAST) 	 <ul style="list-style-type: none"> Expectation maximization algorithms Linear algebra Data mining including <ul style="list-style-type: none"> Clustering K-means Deterministic annealing clustering Multidimensional scaling (MDS) 	 <ul style="list-style-type: none"> Many MPI scientific applications utilizing a wide variety of communication constructs including local interactions Solving differential equations and particle dynamics with short-range forces
← Domain of MapReduce and Iterative Extensions →			MPI

Fig: comparison of different map reducing methods.

GOOGLE FILE SYSTEM (GFS)

- GFS was built primarily as the fundamental storage service for Google's search engine, In addition, GFS was designed for Google applications, and Google applications were built for GFS.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.

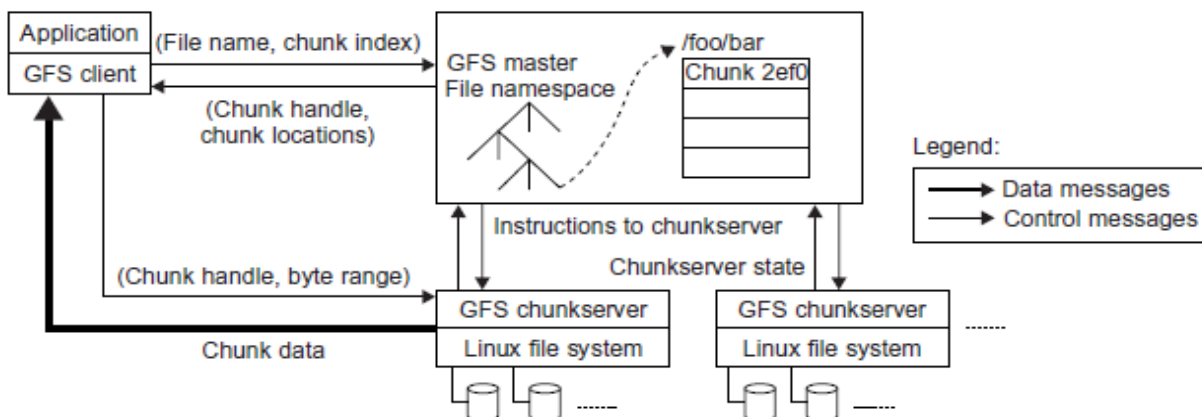
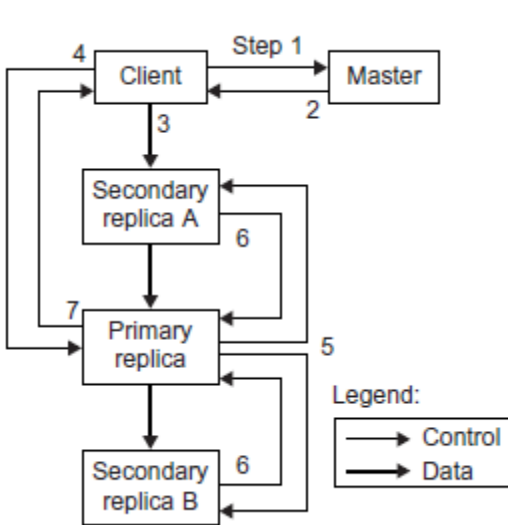


Fig: Architecture of Google File System (GFS)



balancing or fail recovery.

GFS architecture is as shown above,

- 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

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. However, this design does have a potential weakness, as the single GFS master could be the performance bottleneck and the single point of failure. To mitigate this,

Google uses a shadow master to replicate all the data on the master, and the design guarantees that all the data operations are performed directly between the client and the chunk server.

BIG TABLE, GOOGLE'S NOSQL SYSTEM

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. 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. It is not possible to solve such a large scale of structured or semi structured 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 millions of operations per second. 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).

Big Tables 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.

Tablet Location Hierarchy

The below fig. shows the **tablet location hierarchy**,

- The first level is a file stored in Chubby that contains the location of the root tablet. The root tablet contains the location of all tablets in a special METADATA table. Each METADATA tablet contains the location of a set of user tablets.
- The root tablet is just the first tablet in the METADATA table, but is treated specially; it is never split to ensure that the tablet location hierarchy has no more than three levels.
- The METADATA table stores the location of a tablet under a row key that is an encoding of the tablet's table identifier and its end row. BigTable includes many optimizations and fault-tolerant features. Chubby can guarantee the availability of the file for finding the root tablet.
- The BigTable master can quickly scan the tablet servers to determine the status of all nodes

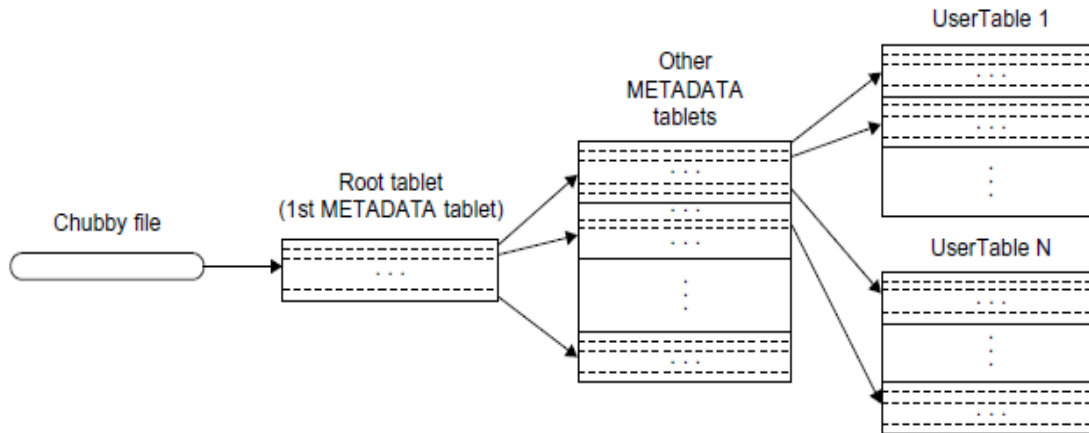


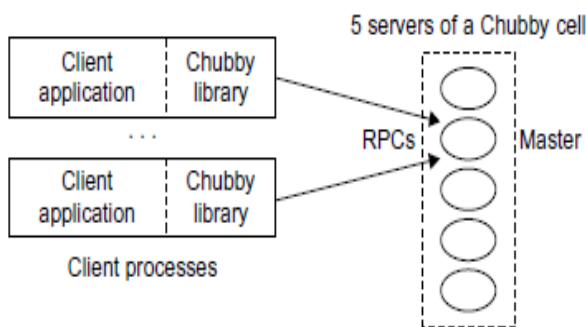
Fig: Tablet location hierarchy in using the BigTable.

- The tablet server uses compaction to store data efficiently. Shared logs are used for logging the operations of multiple tablets so as to reduce the log space as well as keep the system consistent.

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 below fig. 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

Fig: Structure of Google Chubby for distributed lock service

whole file system reliable and consistent. Chubby has become Google's primary internal name service. GFS and BigTable use Chubby to elect a primary from redundant replicas.

PROGRAMMING ON AMAZON AWS AND MICROSOFT AZURE

- programming on Amazon EC2
- Amazon Simple Storage Service (S3)
- Amazon Elastic Block Store (EBS) and SimpleDB
- Microsoft Azure Programming Support.

Amazon offers the Simple Queue Service (SQS) and Simple Notification Service (SNS), Elastic load balancing automatically distributes incoming application traffic across multiple Amazon EC2 instances and allows you to avoid non operating nodes and to equalize load on functioning images. Both auto-scaling and elastic load balancing are enabled by Cloud Watch which monitors running instances.

Programming on Amazon EC2

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 workflow to create an AMI is

Create an AMI→Create Key Pair→Configure Firewall→Launch

There exist three types of AMI

Private AMI : images are created by us and which are private by default.

Public AMI: images are created by the users and released to the AWS community.

Paid QAMI: one can create image providing specify functions.

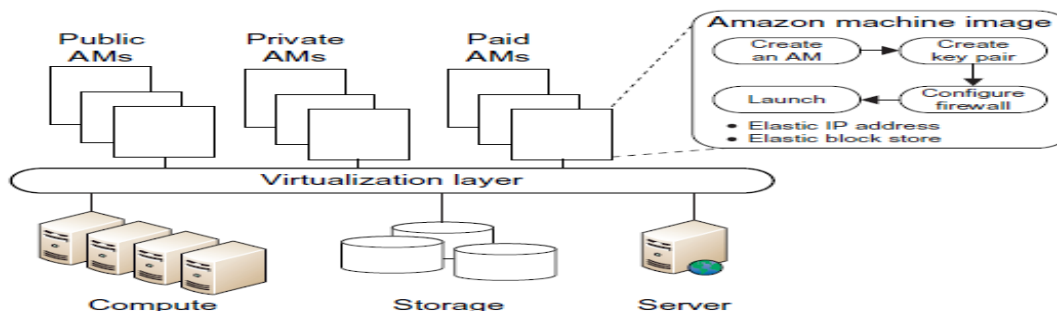


Fig: Amazon EC2 execution environment.

Amazon Simple Storage Service (S3) : 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. Users can access their objects through Simple Object Access Protocol (SOAP) with either browsers or other client programs which support SOAP.

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. 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.

Some key features of S3:

- Redundant through geographic dispersion.
- Designed to provide 99.999999999 percent durability and 99.99 percent availability of objects
- Authentication mechanisms to ensure that data is kept secure from unauthorized access. • Per-object URLs and ACLs (access control lists).
- Default downloads protocol of HTTP. A BitTorrent protocol interface is provided to lower costs for high-scale distribution.
- \$0.055 (more than 5,000 TB) to 0.15 per GB per month storage (depending on total amount).
- First 1 GB per month input or output free and then \$.08 to \$0.15 per GB for transfers outside an S3 region.
- There is no data transfer charge for data transferred between Amazon EC2 and Amazon S3.

Amazon Elastic Block Store (EBS) and SimpleDB

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.

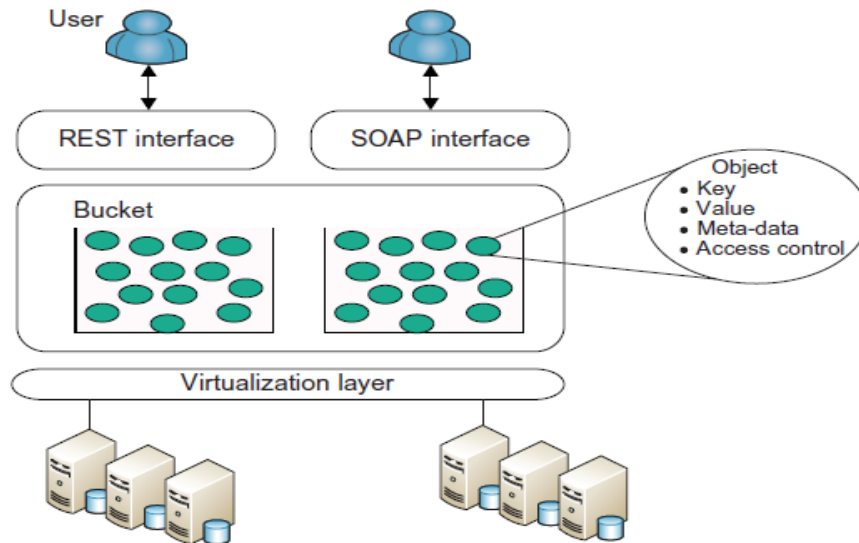


Fig: Amazon S3 execution environment

Multiple volumes can be mounted to the same instance. These storage volumes behave like raw, unformatted block devices, with user-supplied device names and a block device interface.

Amazon SimpleDB Service: 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 .

MICROSOFT AZURE PROGRAMMING SUPPORT

When the system is running, services are monitored and one can access event logs, trace/debug data, performance counters, IIS web server logs, crash dumps, and other log files. This information can be saved in Azure storage. Note that there is no debugging capability for running cloud applications, but debugging is done from a trace. One can divide the basic features into storage and compute capabilities. The Azure application is linked to the Internet through a customized compute VM called a web role supporting basic Microsoft web hosting. Such configured VMs are often called appliances. The other important compute class is the worker role reflecting the importance in cloud computing of a pool of compute resources that are scheduled as needed. The roles support HTTP(S) and TCP. Roles offer the following methods.

- **The OnStart()** method which is called by the Fabric on startup, and allows you to perform initialization tasks.
- **The OnStop()** method which is called when the role is to be shut down and gives a graceful exit.
- **The Run()** method which contains the main logic

SQLAzure : Azure offers a very rich set of storage capabilities, All the storage modalities are accessed with REST interfaces except for the recently introduced Drives that are analogous to Amazon EBS. The basic storage system is built from blobs which are analogous to S3 for Amazon. Blobs are arranged as a three-level hierarchy: Account → Containers → Page or Block Blobs. Containers are analogous to directories in traditional file systems with the account acting as the root. The block blob is used for streaming data and each such blob is made up as a sequence of blocks of upto 4MB each, while each block has a 64 byte ID. Block blobs can be up to 200GB in size. Page blobs are for random read/write access and consist of an array of pages with a maximum blob size of 1 TB. One can associate metadata with blobs as <name, value> pairs with up to 8 KB per blob.

Azure Tables : The Azure Table and Queue storage modes are aimed at much smaller data volumes. Queues provide reliable message delivery and are naturally used to support work spooling between web and worker roles.

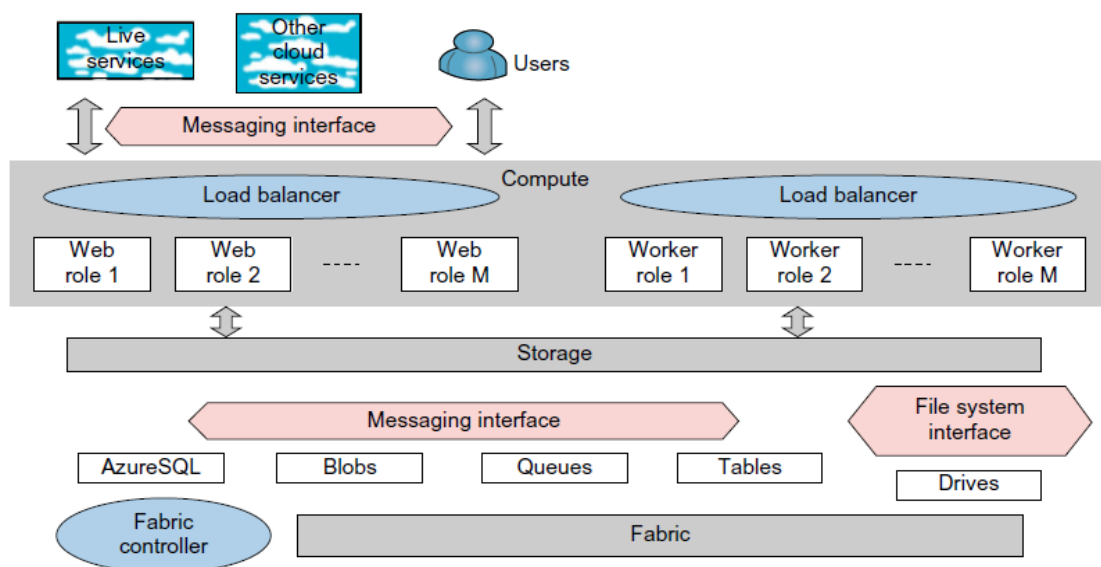


Fig: Features of the Azure cloud platform

Queues consist of an unlimited number of messages which can be retrieved and processed at least once with an 8 KB limit on message size. Azure supports PUT, GET, and DELETE message operations as well as CREATE and DELETE for queues. Each account can have any number of Azure tables which consist of rows called entities and columns called properties.

There is no limit to the number of entities in a table and the technology is designed to scale well to a large number of entities stored on distributed computers. All entities can have up to 255 general properties which are <name, type, value> triples.

EMERGING CLOUD SOFTWARE ENVIRONMENTS

Here popular cloud operating systems and emerging software environments are discussed which are as follows:

Open Source Eucalytus and Nimbus

Open Nebula, Sector/Sphere and Open Stack

Manjara soft Aneka Cloud and Applications

OPEN SOURCE EUCALYPTUS AND NIMBUS

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.
- The below diagram Shows the architecture based on the need to manage VM images. The system supports cloud programmers in VM image management as follows

Eucalyptus Architecture

The Eucalyptus system is an open software environment.

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. where any user can bundle her own root

file system, and upload and then register this image and link it with a particular kernel and ram disk image.

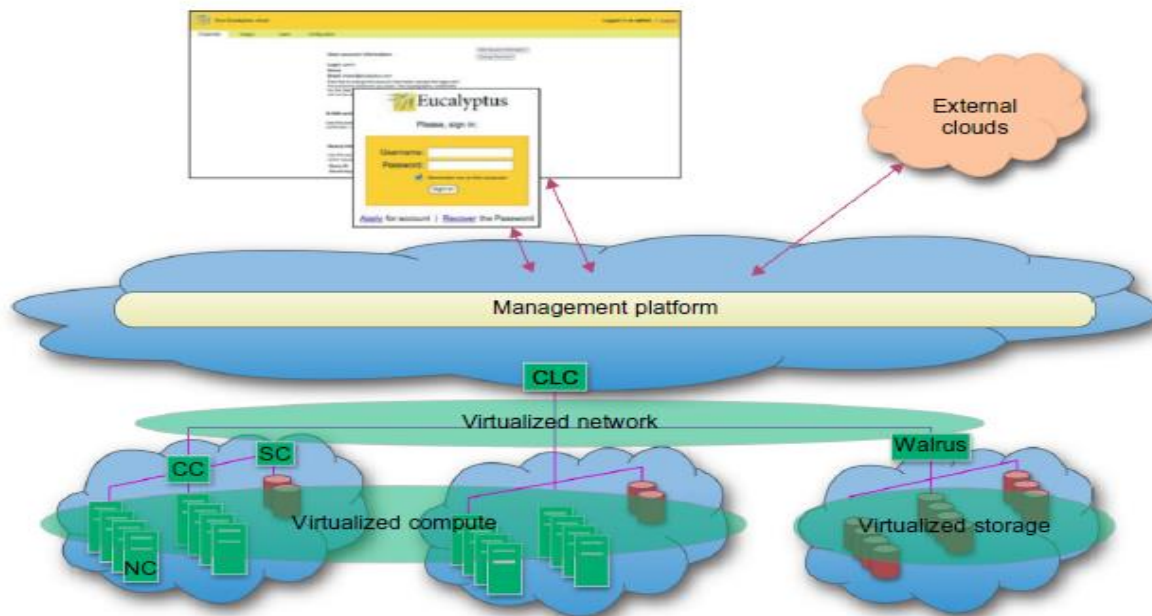


Fig: Eucalyptus architecture for VM image management

This image is uploaded into a user-defined bucket within Walrus, and can be retrieved anytime from any availability zone. This allows users to create specialty virtual appliances and deploy them within Eucalyptus with ease

- The Eucalyptus system is available in a commercial proprietary version, as well as the open source version we just described.

Nimbus : Nimbus is a set of open source tools which together provide an IaaS cloud computing solution. The below fig. shows the architecture of Nimbus, which allows a client to lease remote resources by deploying VMs on those resources and configuring them to represent the environment desired by the user. To this end, Nimbus provides a special web interface known as Nimbus Web. Its aim is to provide administrative and user functions in a friendly interface. Nimbus Web is centered on a Python Django

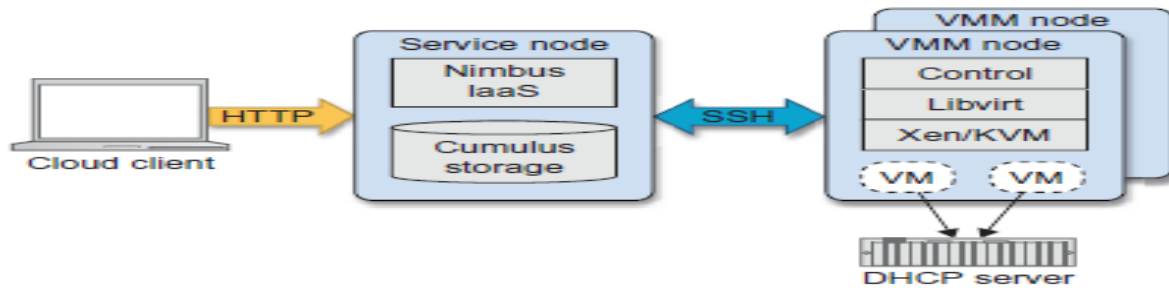


Fig: Nimbus cloud infrastructure.

As shown in the above architecture

- A storage cloud implementation called Cumulus has been tightly integrated with the other central services, although it can also be used stand-alone.
- Cumulus is compatible with the Amazon S3 REST API, but extends its capabilities by including features such as quota management.
- Therefore, clients such as boto and s2cmd, that work against the S3 REST API, work with Cumulus.
- Nimbus supports two resource management strategies.
 - The first is the default “resource pool” mode. In this mode, the service has direct control of a pool of VM manager nodes and it assumes it can start VMs.
 - The other supported mode is called “pilot.” Here, the service makes requests to a cluster’s Local Resource Management System (LRMS) to get a VM manager available to deploy VMs.

OPEN NEBULA, SECTOR/SPHERE AND OPEN STACK

OpenNebula is an open source toolkit which allows users to transform existing infrastructure into an IaaS cloud with cloud-like interfaces. Its architecture is as shown below, which shows the OpenNebula architecture and its main components.

- The architecture of OpenNebula is designed to be flexible and modular to allow integration with different storage and network infrastructure configurations, and hypervisor technologies.
- it has the following components
- it has the following components, **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.

- Another 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.
- 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,
 - OpenNebula offers management interfaces to integrate the core's functionality within other data-center management tools, such as accounting or monitoring frameworks. To this end, OpenNebula implements the **libvirt**.

Sector/Sphere : Sector/Sphere is a software platform that supports very large distributed data storage and simplified distributed data processing over large clusters of commodity computers, either within a data center or across multiple data centers.

- The system consists of the Sector distributed file system and the Sphere parallel data processing framework.

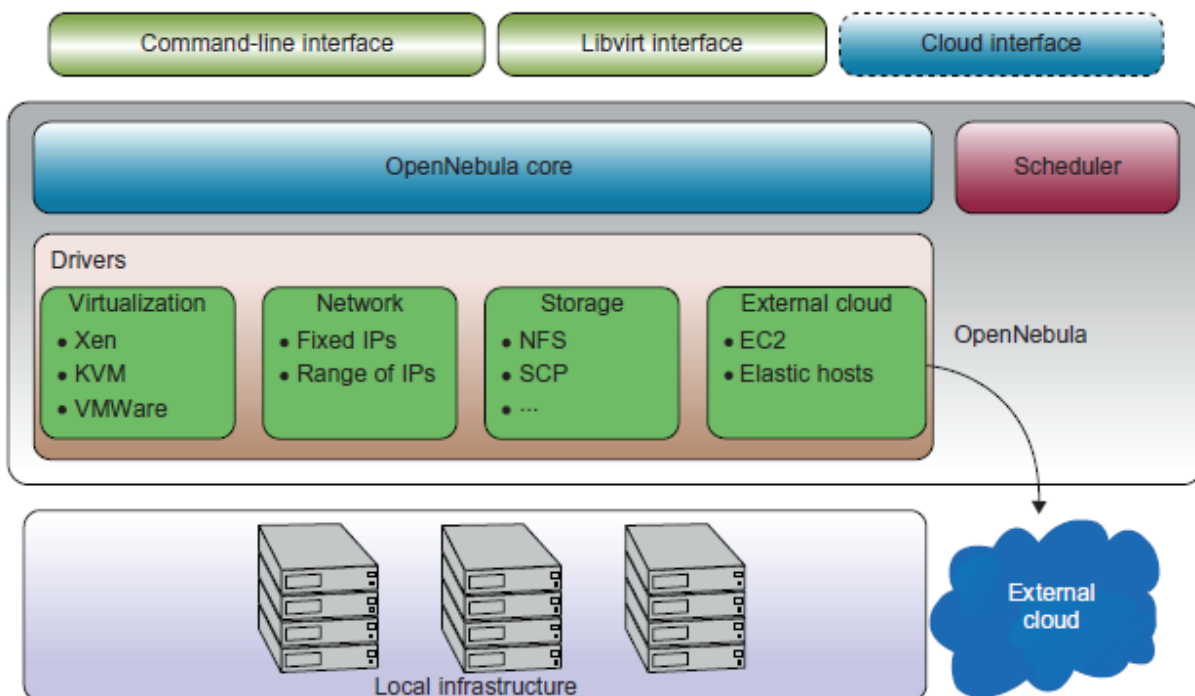


Fig: OpenNebula architecture and its main components.

- Sector is a distributed file system (DFS) that can be deployed over a wide area and allows users to manage large data sets from any location with a high speed network

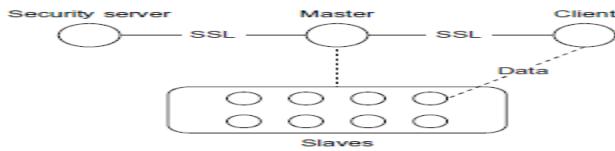


fig: the sector /sphere system architecture

connection. The fault tolerance is implemented by replicating data in the file system and managing the replicas.

The architecture consists of the following four components (**security servers, master, client, slaves**)

- The first component is the **security server**, which is responsible for authenticating master servers, slave nodes, and users.
- The **master servers** are considered as the infrastructure core. The master server maintains file system metadata, schedules jobs, and responds to users' requests.
- The **slave nodes**, is the place where data is stored and processed. The slave nodes can be located within a single data center or across multiple data centers with high-speed network connections.
- The **client component** it provides tools and programming APIs for accessing and processing Sector data.

OpenStack : OpenStack was 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.

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.
- The below fig. 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 workers through a queue.

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.

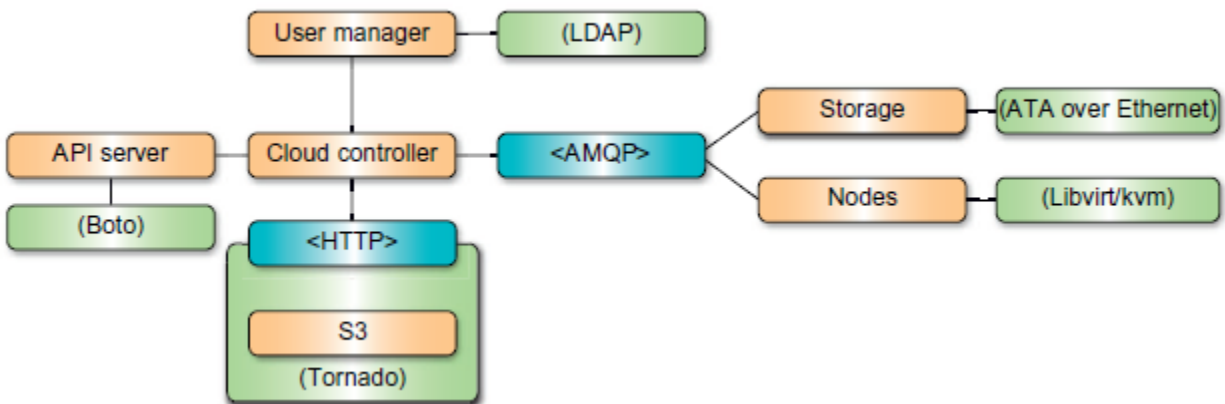


FIG: OpenStack Nova system architecture

- 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.

MANJARA SOFT ANEKA CLOUD AND APPLICATIONS

Aneka (www.manjrasoft.com/) is a cloud application platform developed by Manjrasoft, based in Melbourne, Australia.

- It is designed to support rapid development and deployment of parallel and distributed applications on private or public clouds.
- It provides a rich set of APIs for transparently exploiting distributed resources and expressing the business logic of applications by using preferred programming abstractions.
- System administrators can leverage a collection of tools to monitor and control the deployed infrastructure.

Some of the **key advantages of Aneka over other** workload distribution solutions include:

- Support of multiple programming and application environments
- Simultaneous support of multiple runtime environments
- Rapid deployment tools and framework
- Ability to harness multiple virtual and/or physical machines for accelerating application provisioning based on users' Quality of Service/service-level agreement (QoS/SLA) requirements.
- Built on top of the Microsoft .NET framework, with support for Linux environments through Mono

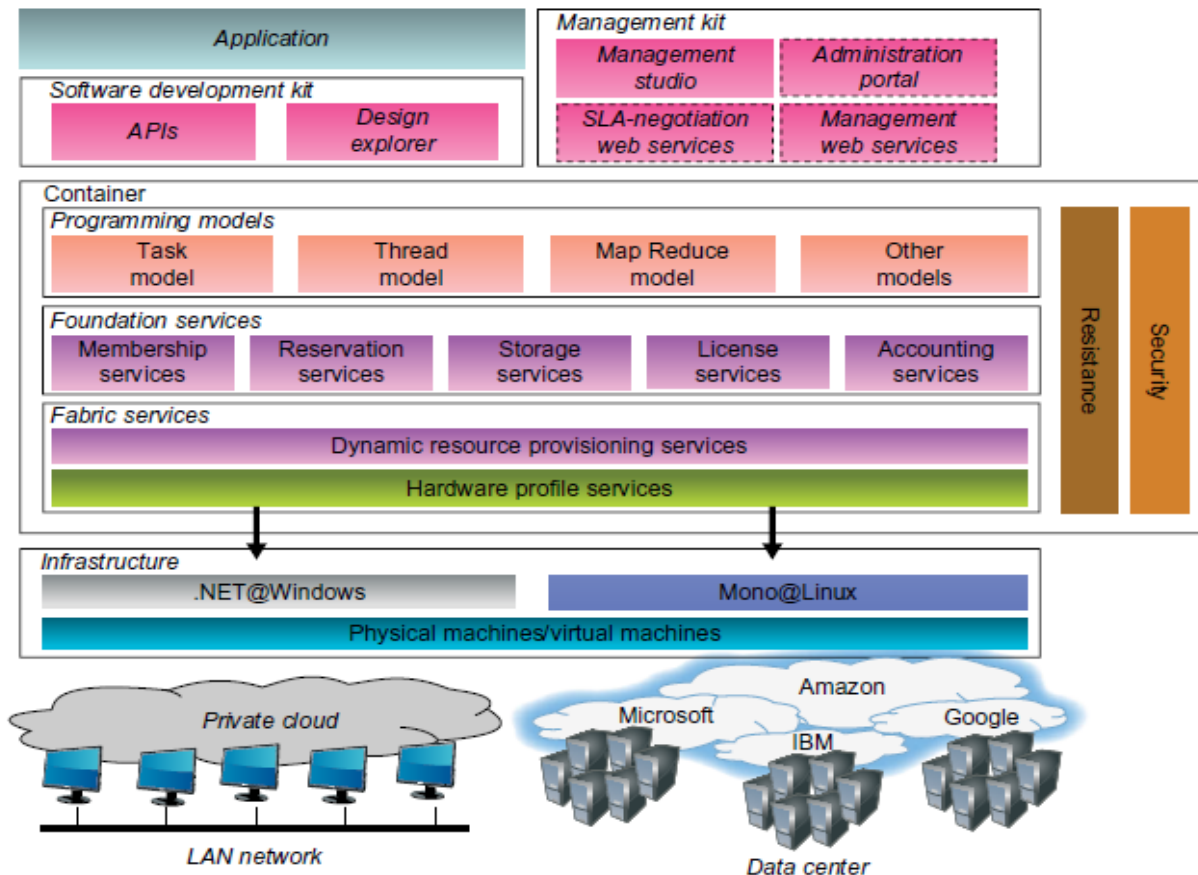


Fig: Architecture of Aneka Cloud

Aneka offers three types of capabilities which are essential for building, accelerating, and managing clouds and their applications:

1. **Build** Aneka includes a new SDK which combines APIs and tools to enable users to rapidly develop applications. Aneka also allows users to build different runtime environments such as enterprise/private cloud by harnessing compute resources in network or enterprise data centers, Amazon EC2, and hybrid clouds.
2. **Accelerate** Aneka supports rapid development and deployment of applications in multiple runtime environments running different operating systems such as Windows or Linux/UNIX..

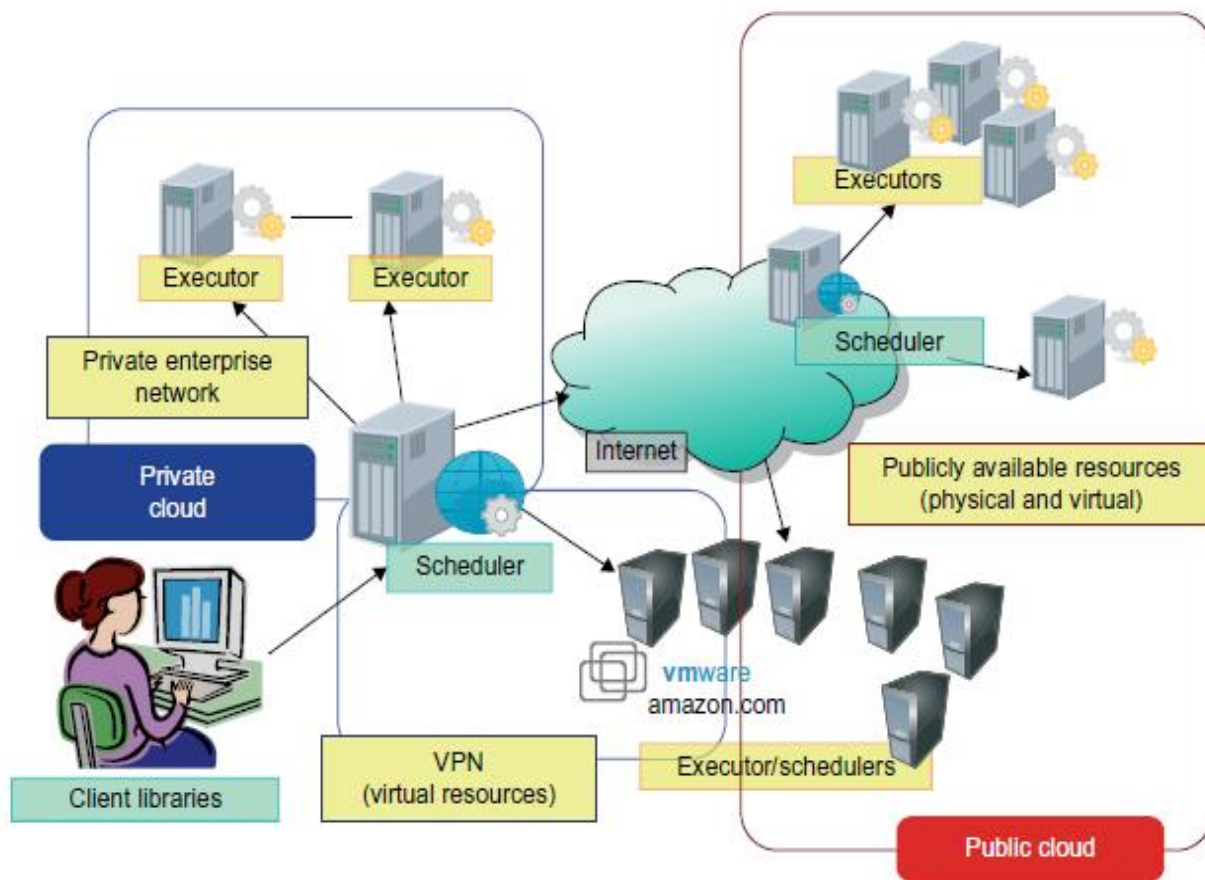


Fig: Aneka using private cloud resources along with dynamically leased public cloud resources

3. Manage Management tools and capabilities supported by Aneka include a GUI and APIs to set up, monitor, manage, and maintain remote and global Aneka compute clouds.

The three important programming models supported by Aneka for both cloud and traditional parallel applications:

1. **Thread programming model.**
2. **Task programming model.**
3. **MapReduce programming model.**

Aneka Architecture

Aneka as a cloud application platform features a homogeneous distributed runtime environment for applications. This environment is built by aggregating together physical and virtual nodes hosting the Aneka container. The container is a

lightweight layer that interfaces with the hosting environment and manages the services deployed on a node.

The available services can be aggregated into three major categories:

Fabric Services: Fabric services implement the fundamental operations of the infrastructure of the cloud. These services include HA and failover for improved reliability, node membership and directory, resource provisioning, performance monitoring, and hardware profiling.

Foundation Services: Foundation services constitute the core functionalities of the Aneka middleware. They provide a basic set of capabilities that enhance application execution in the cloud..

Application Services: Application services deal directly with the execution of applications and are in charge of providing the appropriate runtime environment for each application model.

Virtual Appliances: Machine virtualization offers a unique opportunity to break software dependencies between applications and their hosting environments.