

**UNIT- IV**

**CLOUD PROGRAMMING MODEL**

**Topics Covered:**

- Ⓢ **Features of Cloud and Grid Platforms**
- Ⓢ **Parallel and Distributed Programming Paradigms**
- Ⓢ **Mapping Applications to Parallel and Distributed Systems**
- Ⓢ **Programming Support of Google App Engine**
- Ⓢ **Programming on Amazon AWS**
- Ⓢ **Emerging Cloud Software Environments**

▶ Open Source Eucalyptus	OpenNebula	Sector/Sphere
▶ OpenStack	Manjrasoft Aneka Cloud	CloudSim

⇒ **Features of Cloud and Grid Platforms**

We summarize important features in real cloud and grid platforms. In four tables, we cover the capabilities, traditional features, data features, and features for programmers and runtime systems to use.

<b>Table 6.1</b> Important Cloud Platform Capabilities	
<b>Capability</b>	<b>Description</b>
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.

**Table 6.3** Traditional Features in Cluster, Grid, and Parallel Computing Environments

**Cluster management:** ROCKS and packages offering a range of tools to make it easy to bring up clusters

**Data management:** Included metadata support such as RDF triple stores (Semantic web success and can be built on MapReduce as in SHARD); SQL and NOSQL included in

**Grid programming environment:** Varies from link-together services as in Open Grid Services Architecture (OGSA) to GridRPC (Ninf, GridSolve) and SAGA

**OpenMP/threading:** Can include parallel compilers such as Cilk; roughly shared memory technologies. Even transactional memory and fine-grained data flow come here

**Portals:** Can be called (science) gateways and see an interesting change in technology from portlets to HUBzero and now in the cloud: Azure Web Roles and GAE

**Scalable parallel computing environments:** MPI and associated higher level concepts including ill-fated HP FORTRAN, PGAS (not successful but not disgraced), HPCS languages (X-10, Fortress, Chapel), patterns (including Berkeley dwarves), and functional languages such as F# for distributed memory

**Virtual organizations:** From specialized grid solutions to popular Web 2.0 capabilities such as Facebook

**Workflow:** Supports workflows that link job components either within or between grids and clouds; relate to LIMS Laboratory Information Management Systems.

### ➤ Data Features and Databases

- **Program Library:** Many efforts have been made to design a VM image library to manage images used in academic and commercial clouds.
- **Blobs and Drives:** This concept is similar to shared file systems such as Lustre used in TeraGrid. The cloud storage is intrinsically fault-tolerant while that on TeraGrid needs backup storage.
- **DPFS:** In general, data transport will be needed to link these two data views. It seems important to consider this carefully, as DPFS file systems are precisely designed for efficient execution of data-intensive applications.
- **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 MapReduce 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 emphasizing distribution and scalability. These are present in the three major clouds: BigTable in Google, SimpleDB in Amazon, and Azure Table for Azure.
- **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.

### ➤ 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.
- Here, **MapReduce** is used to group the processes in clouds to reduce individual process mapping. Example: Google MapReduce, Apache Hadoop uses MapReduce

- The major open source/commercial MapReduce implementations are Hadoop and Dryad with execution possible with or without VMs.
- We desire a 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, we need protection features that may help us to achieve scalability, security, privacy, and availability.

### ⇒ Parallel and Distributed 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. 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.
  - **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, parallel and distributed programming paradigms or models are offered to abstract many parts of the data flow from



users to provide users with an abstraction layer to hide implementation details of the data flow which users formerly ought to write codes for.

Other motivations behind parallel and distributed programming models are

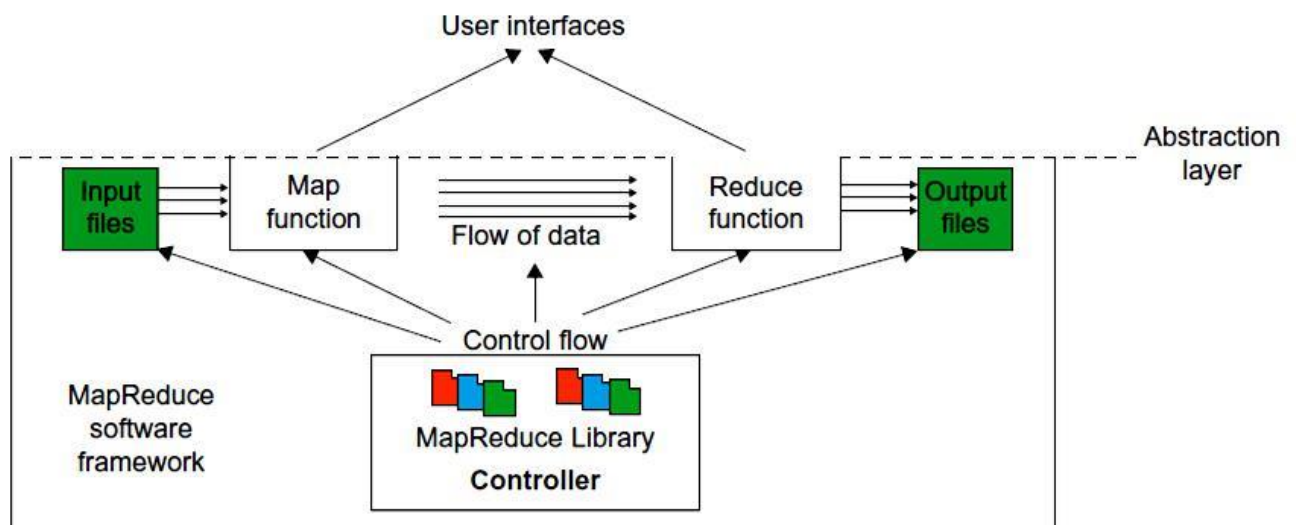
- (1) to improve productivity of programmers,
- (2) to decrease programs' time to market,
- (3) to leverage underlying resources more efficiently,
- (4) to increase system throughput, and
- (5) to support higher levels of abstraction.

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

Given below FIGURE illustrates the logical data flow from the Map to the Reduce function in MapReduce frameworks. 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.



**Figure:** 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.

### 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. The abstraction layer provides two well-defined interfaces in the form of two functions: Map and Reduce. 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 userdefined functions to the MapReduce library.

The overall structure of a user's program containing the Map, Reduce, and the Main functions is given below. The Map and Reduce are two major subroutines. They will be called to implement the desired function performed in the main program.

```

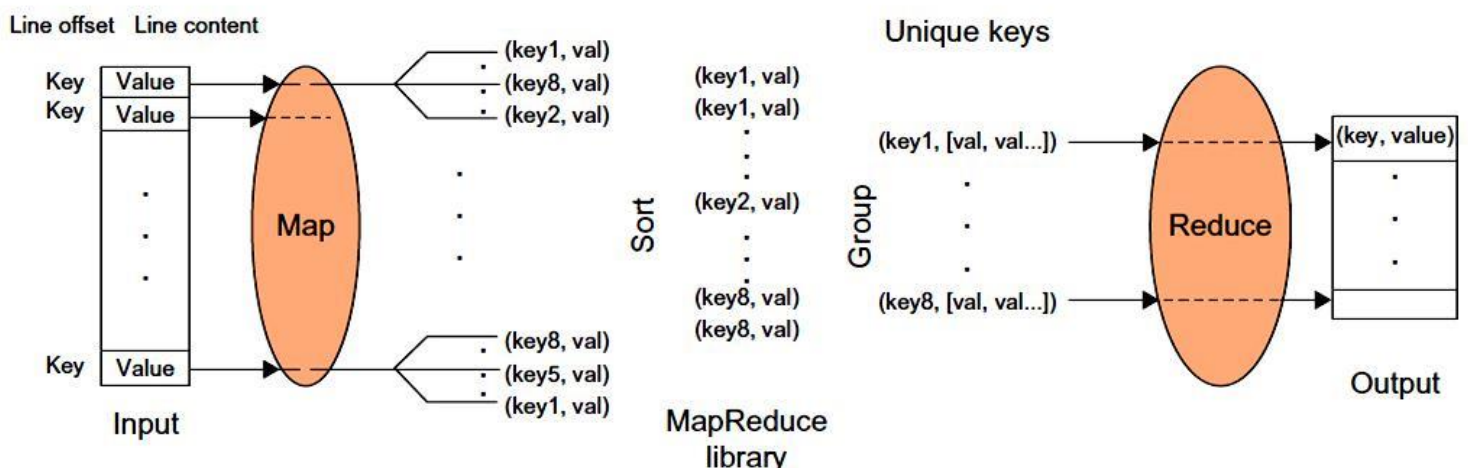
Map Function (... )
{
... ..
}
Reduce Function (... )
{
... ..
}
Main Function (... )
{
Initialize Spec object
... ..
MapReduce (Spec, & Results)
}

```

### MapReduce Logical Data Flow

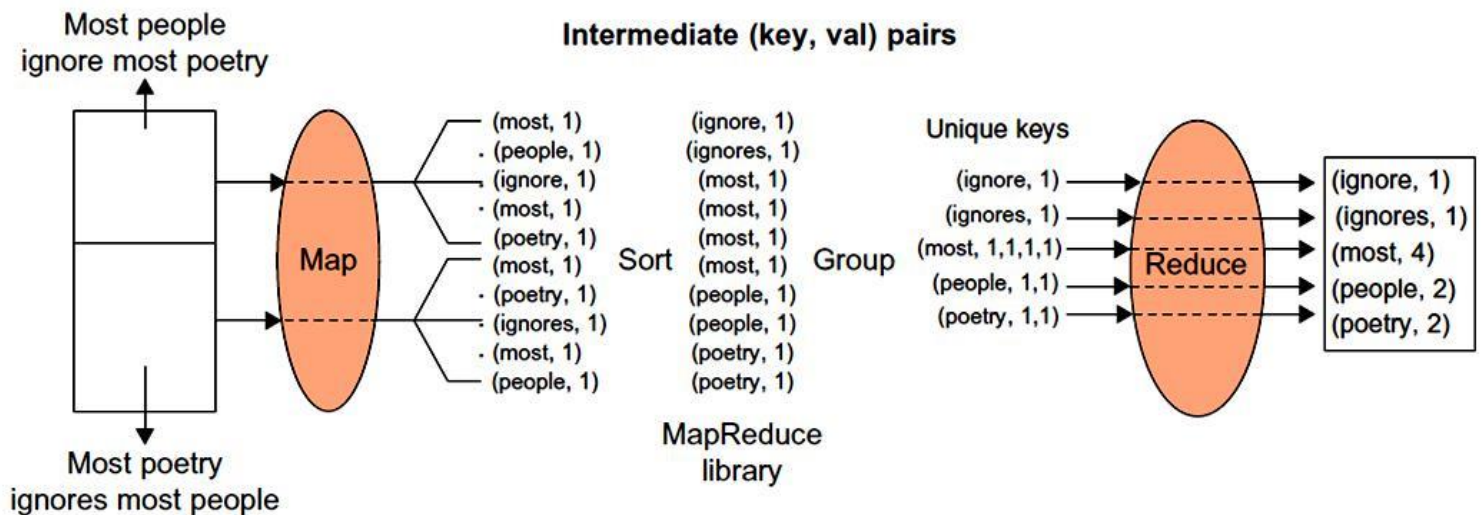
The input data to the Map function is in the form of a (key, value) pair. For example, the key is the line offset within the input file and the value is the content of the line. The output data from the Map function is structured as (key, value) pairs called intermediate (key, value) pairs. In other 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. 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]). The 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.

#### Intermediate (key, val) pairs



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

**Example:** sample MapReduce application word count with two lines (1) “most people ignore most poetry” and (2) “most poetry ignores most people.” The Map function simultaneously produces a number of intermediate (key, value) pairs for each line of content so that each word is the intermediate key with 1 as its intermediate value; for example, (ignore, 1). Then the MapReduce library collects all the generated intermediate (key, value) pairs and sorts them to group the 1’s for identical words; for example, (people, [1,1]). Groups are then sent to the Reduce function in parallel so that it can sum up the 1 values for each word and generate the actual number of occurrence for each word in the file; for example, (people, 2).



**Figure:** The data flow of a word-count problem using the MapReduce functions (Map, Sort, Group and Reduce) in a cascade operations.

### 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

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

Then the MapReduce library collects all the produced intermediate (key, value) pairs from all input (key, value) pairs, and sorts them based on the “key” part. Finally, the Reduce function is applied in parallel to each group producing the collection of values as output as illustrated here:

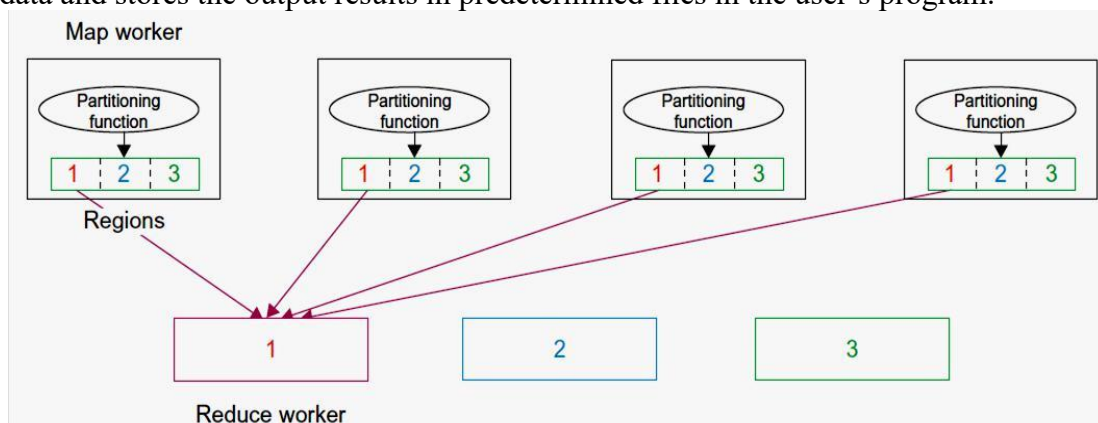
$$(key_2, \text{List}(val_2)) \xrightarrow{\text{Reduce Function}} \text{List}(val_2)$$

**MapReduce Actual Data and Control Flow:** The main responsibility of the MapReduce framework is to efficiently run a user’s program on a distributed computing system, which handles all partitioning, mapping, synchronization, communication, and scheduling details of such data flows

- 1) Data partitioning:** The MapReduce library splits the input data (files), already stored in GFS, into M pieces that also correspond to the number of map tasks.
- 2) Computation partitioning** This is implicitly handled (in the MapReduce framework) by obliging users to write their programs in the form of the Map and Reduce functions. The MapReduce library only

generates copies of a user program (e.g., by a fork system call) containing the Map and the Reduce functions, distributes them, and starts them up on a number of available computation engines.

- 3) **Determining the master and workers:** The MapReduce architecture is based on a master worker model and one of the copies of the user program becomes the master and the rest become workers. The master picks idle workers, and assigns the map and reduce tasks to them. A map/reduce worker is typically a computation engine such as a cluster node to run map/ reduce tasks by executing Map/Reduce functions.
- 4) **Reading the input data (data distribution):** Each map worker reads its corresponding portion of the input data, namely the input data split, and sends it to its Map function.
- 5) **Map function:** Each Map function receives the input data split as a set of (key, value) pairs to process and produce the intermediated (key, value) pairs.
- 6) **Combiner function:** This is an optional local function within the map worker which applies to intermediate (key, value) pairs. The user can invoke the Combiner function inside the user program. The Combiner function runs the same code written by users for the Reduce function as its functionality is identical to it. The Combiner function merges the local data of each map worker before sending it over the network to effectively reduce its communication costs.
- 7) **Partitioning function:** The intermediate (key, value) pairs produced by each map worker are partitioned into R regions, equal to the number of reduce tasks, by the Partitioning function to guarantee that all (key, value) pairs with identical keys are stored in the same region. As a result, since reduce worker i reads the data of region i of all map workers, all (key, value) pairs with the same key will be gathered by reduce worker i accordingly. a Partitioning function could simply be a hash function (e.g.,  $\text{Hash}(\text{key}) \bmod R$ ) that forwards the data into particular regions.
- 8) **Synchronization:** MapReduce applies a simple synchronization policy to coordinate map workers with reduce workers, in which the communication between them starts when all map tasks finish.
- 9) **Communication:** Reduce worker i, already notified of the location of region i of all map workers, uses a remote procedure call to read the data from the respective region of all map workers. Since all reduce workers read the data from all map workers, all-to-all communication among all map and reduce workers, which incurs network congestion, occurs in the network.
- 10) **Sorting and Grouping:** When the process of reading the input data is finalized by a reduce worker, the data is initially buffered in the local disk of the reduce worker. Then the reduce worker groups intermediate (key, value) pairs by sorting the data based on their keys, followed by grouping all occurrences of identical keys.
- 11) **Reduce function:** The reduce worker iterates over the grouped (key, value) pairs, and for each unique key, it sends the key and corresponding values to the Reduce function. Then this function processes its input data and stores the output results in predetermined files in the user's program.



**Figure: Use of MapReduce partitioning function to link the Map and Reduce workers**



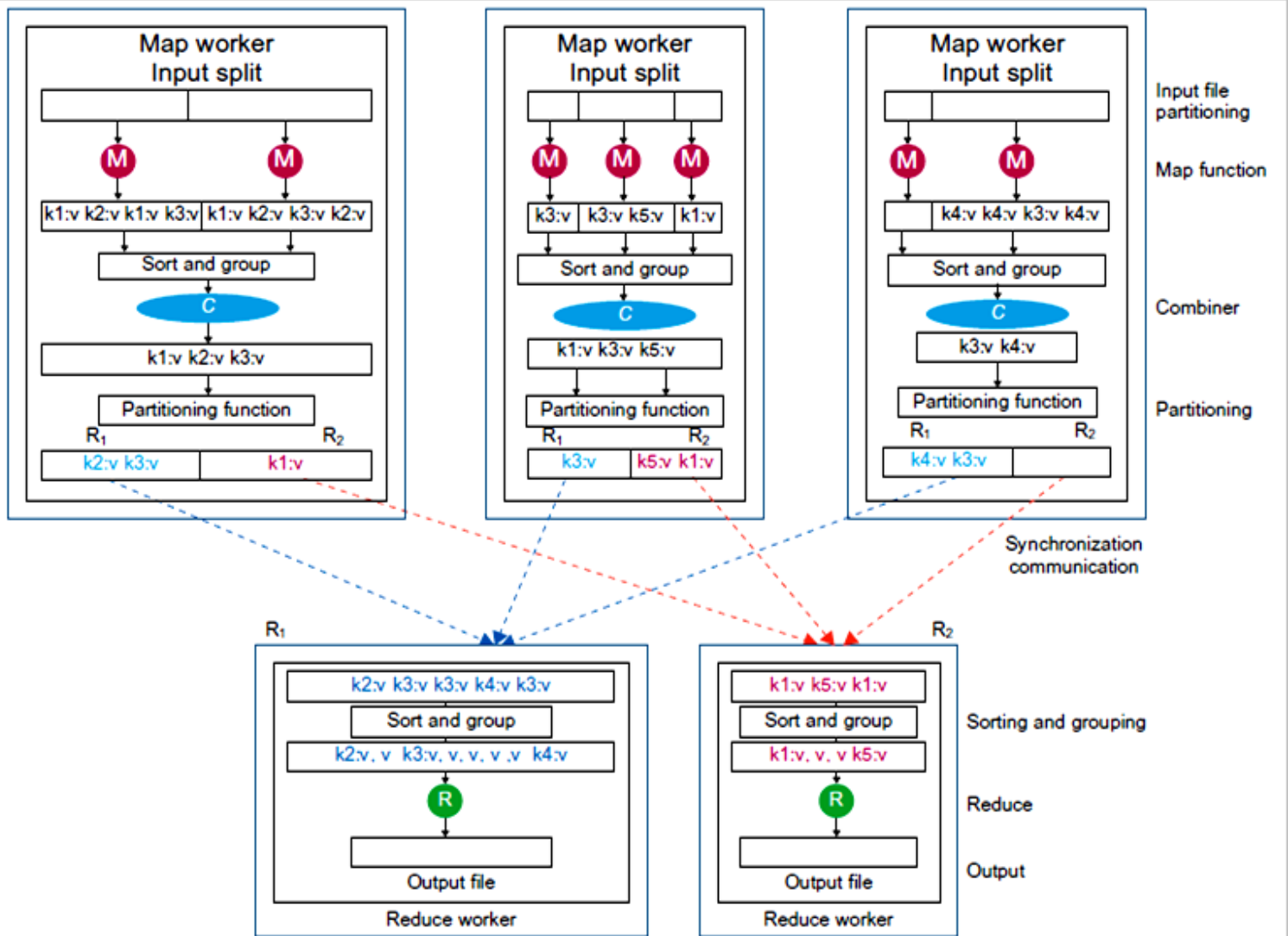
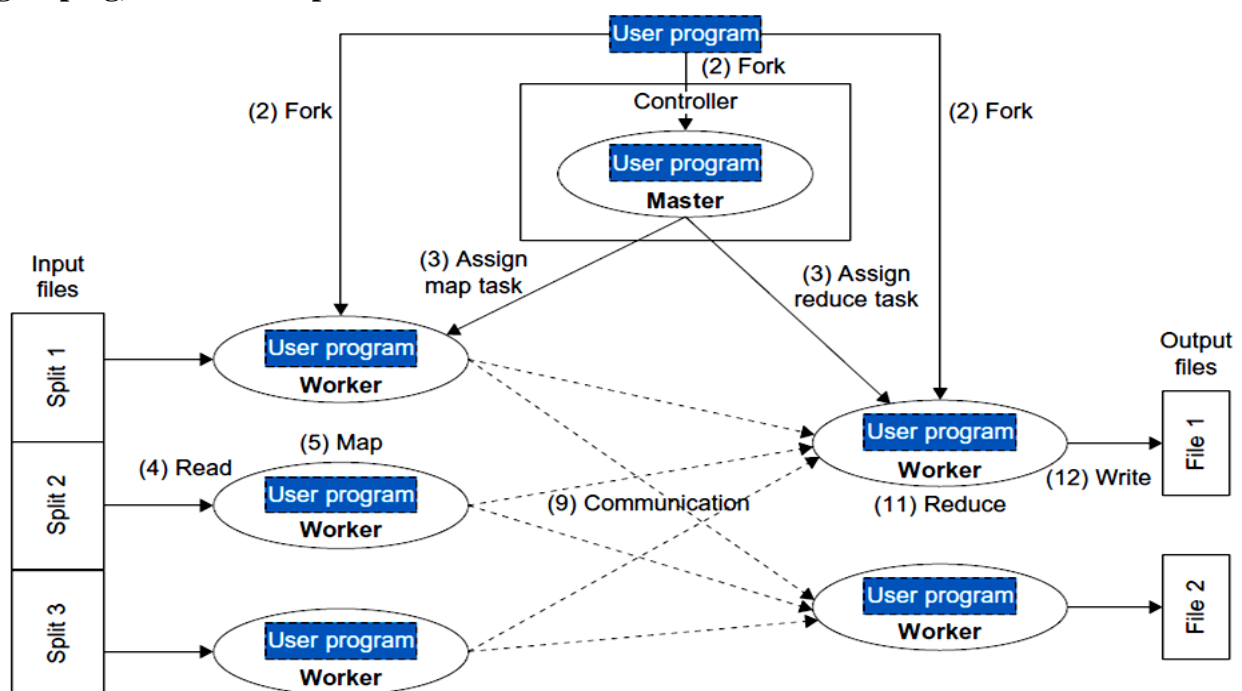


Figure: Data flow implementation of many functions in the Map workers and in the Reduce workers through multiple sequences of partitioning, combining, synchronization and communication, sorting and grouping, and reduce operations.





## Twister and Iterative MapReduce

It is important to understand the performance of different runtimes and, in particular, to compare Message Passing Interface (MPI) and MapReduce. The two major sources of parallel overhead are load imbalance and communication. The communication overhead in MapReduce can be quite high, for two reasons:

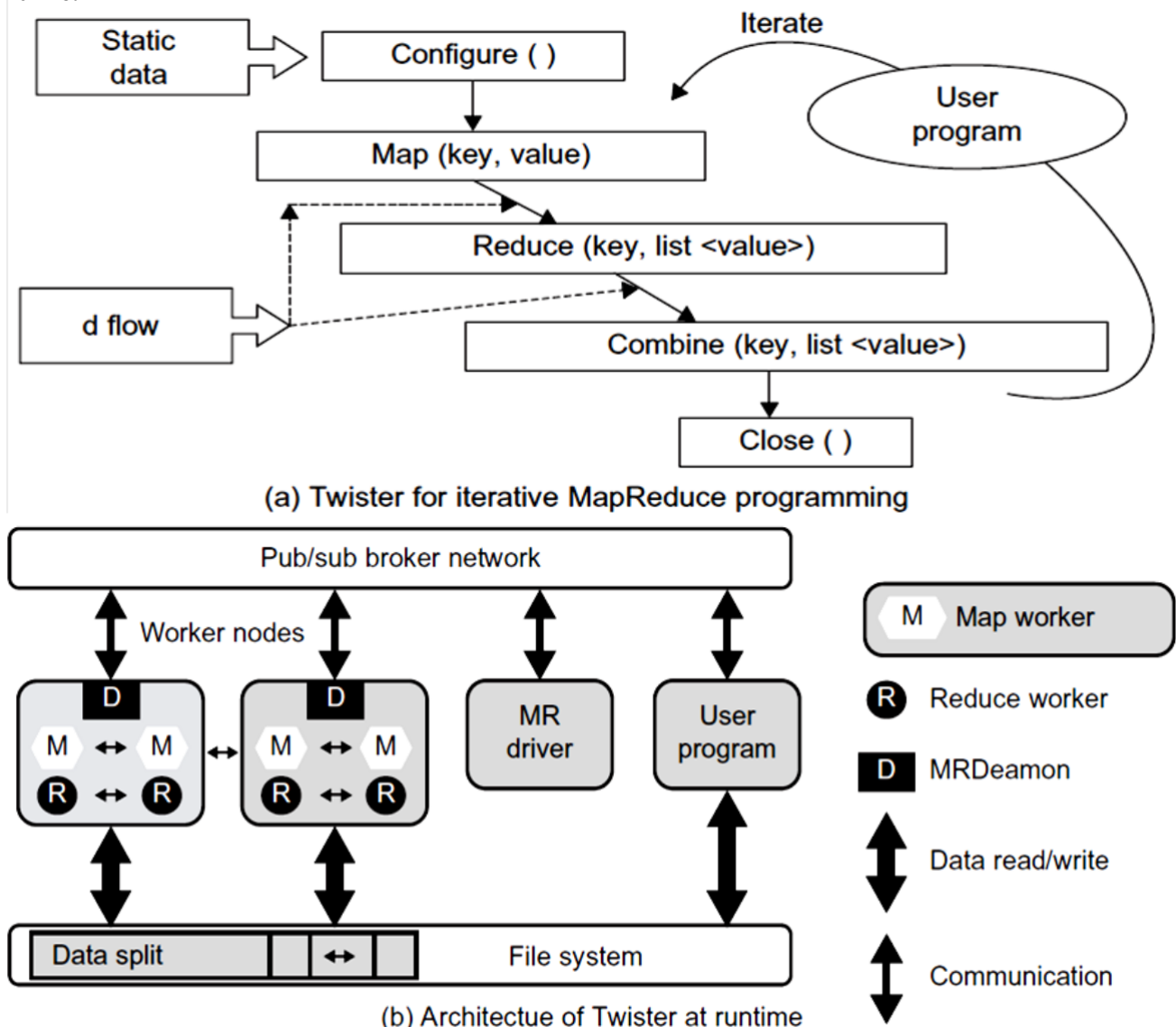
- MapReduce reads and writes via files, whereas MPI transfers information directly between nodes over the network.
- MPI does not transfer all data from node to node, but just the amount needed to update information. We can call the MPI flow  $\delta$  flow and the MapReduce flow full data flow.

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:

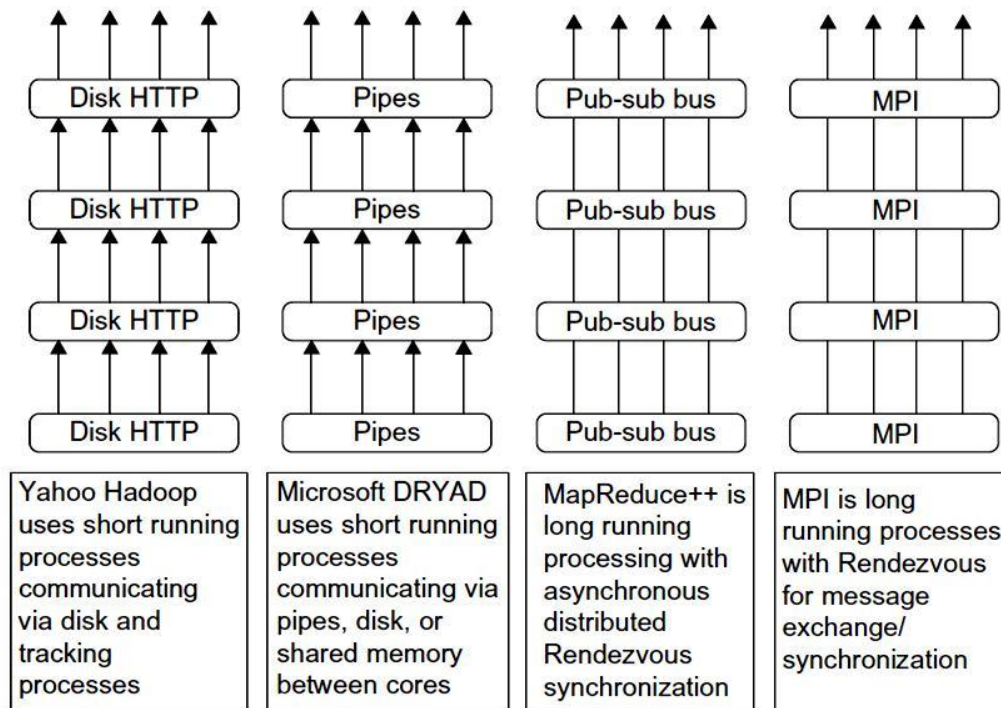
- Stream information between steps without writing intermediate steps to disk.
- Use long-running threads or processors to communicate the  $\delta$  (between iterations) flow.

The Twister programming paradigm and its implementation architecture at run time are illustrated in below Figure. Twister distinguishes the static data which is never reloaded from the dynamic  $\delta$  flow that is communicated.

**Figure: Twister - An iterative MapReduce programming paradigm for repeated MapReduce executions. (a) Twister for iterative MapReduce programming and (b) Architectue of Twister at runtime.**



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



**Figure: compare 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.
  - The Hadoop implementation of MapReduce uses the Hadoop Distributed File System (HDFS) as its underlying layer.
  - 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.
- **HDFS Architecture:** HDFS has a master/slave architecture containing a single NameNode as the master and a number of DataNodes as workers (slaves). To store a file in this architecture, HDFS splits the file into fixed-size blocks (e.g., 64 MB) and stores them on workers (DataNodes). The mapping of blocks to DataNodes is determined by the NameNode. The NameNode (master) also manages the file system's metadata and namespace. In such systems, the namespace is the area maintaining the metadata, and metadata refers to all the information stored by a file system that is needed for overall management of all files.

Each DataNode, usually one per node in a cluster, manages the storage attached to the node. Each Data Node is responsible for storing and retrieving its file blocks.

### ► HDFS Features:

Distributed file systems have special requirements, such as performance, scalability, concurrency control, fault tolerance, and security requirements, to operate efficiently. Two important characteristics of HDFS are,



#### HDFS Fault Tolerance

- **Block Replication:** HDFS stores a file as a set of blocks and each block is replicated and distributed across the whole cluster. The replication factor is set by the user and is three by default.
- **Replica placement:** HDFS stores one replica in the same node the original data is stored, one replica on a different node but in the same rack, and one replica on a different node in a different rack to provide three copies of the data.
- **Heartbeat and Blockreport messages:** Heartbeats and Blockreports are periodic messages sent to the NameNode by each DataNode in a cluster. Receipt of a Heartbeat implies that the DataNode is functioning properly, while each Blockreport contains a list of all blocks on a DataNode.



**HDFS High-Throughput Access to Large Data Sets (Files),** this is for providing batch processing rather than interactive processing, data access throughput in HDFS is more important than latency. Applications run on HDFS typically have large data sets, individual files are broken into large blocks (e.g., 64MB) to allow HDFS to decrease the amount of metadata storage required per file.



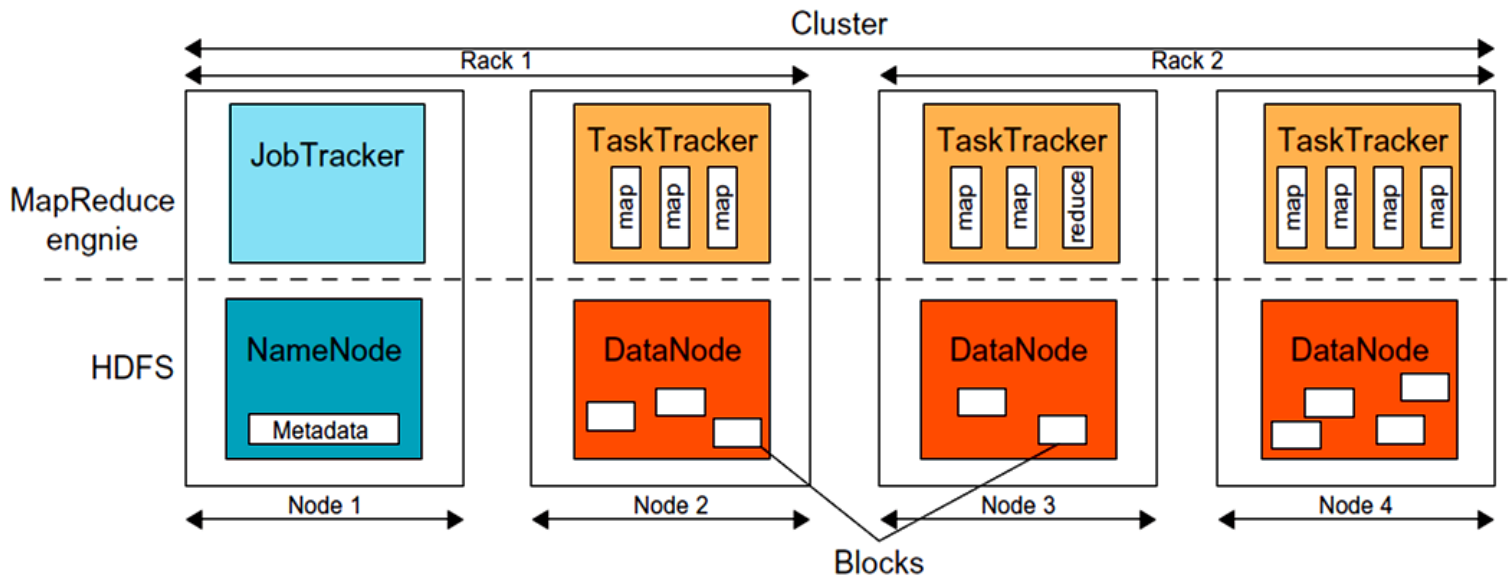
**HDFS Operation:** The control flow of HDFS operations such as write and read can properly highlight roles of the NameNode and DataNodes in the managing operations. The control flow of the main operations of HDFS on files is further described to manifest the interaction between the user, the NameNode, and the DataNodes in such systems.

- 1) **Reading a file:** To read a file in HDFS, a user sends an “open” request to the NameNode to get the location of file blocks. For each file block, the NameNode returns the address of a set of DataNodes containing replica information for the requested file. The user calls the read function to connect to the closest DataNode containing the first block of the file. After the first block is streamed from the respective DataNode to the user, the established connection is terminated and the same process is repeated for all blocks of the requested file until the whole file is streamed to the user.
- 2) **Writing to a file:** To write a file in HDFS, a user sends a “create” request to the NameNode to create a new file in the file system namespace. If the file does not exist, the NameNode notifies the user and allows him to start writing data to the file by calling the write function. The first block of the file is written to an internal queue termed the data queue while a data streamer monitors its writing into a DataNode.



### Architecture of MapReduce in Hadoop

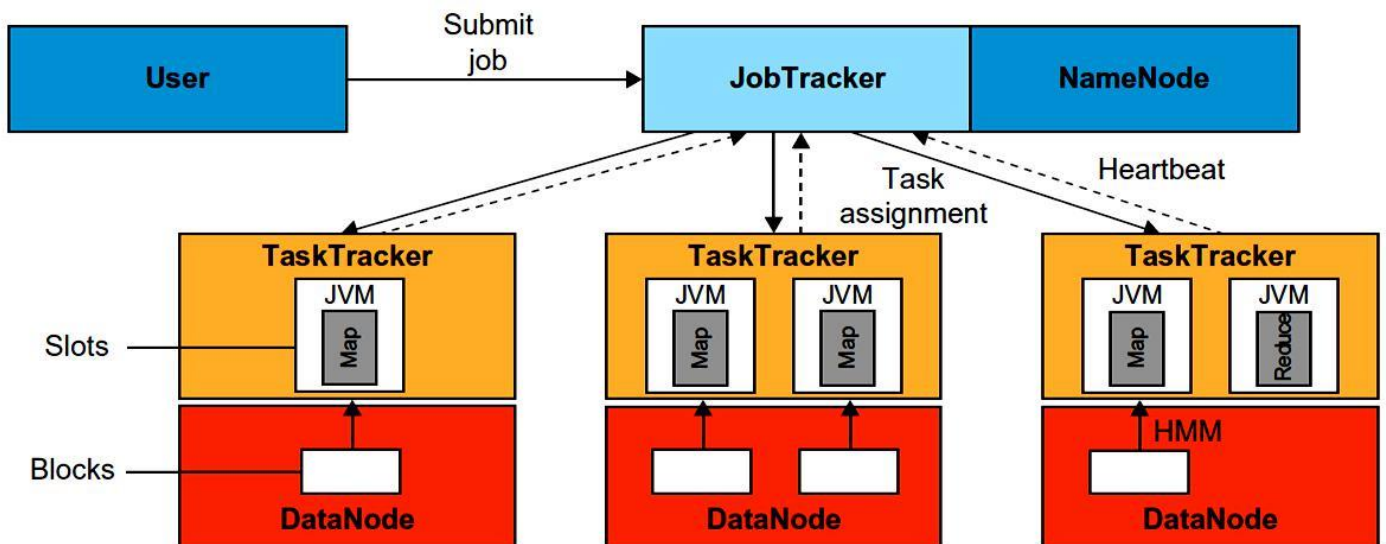
The topmost layer of Hadoop is the MapReduce engine that manages the data flow and control flow of MapReduce jobs over distributed computing systems. The MapReduce engine also has a master/slave architecture consisting of a single JobTracker as the master and a number of TaskTrackers as the slaves (workers). The JobTracker manages the MapReduce job over a cluster and is responsible for monitoring jobs and assigning tasks to TaskTrackers.



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

#### ▶ Running a Job in Hadoop

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



**Figure: Data flow in running a MapReduce job at various task trackers using the Hadoop library.**

**Job Submission** Each job is submitted from a user node to the JobTracker node that might be situated in a different node within the cluster through the following procedure:

- A user node asks for a new job ID from the JobTracker and computes input file splits.
- The user node copies some resources, such as the job's JAR file, configuration file, and computed input splits, to the JobTracker's file system.
- The user node submits the job to the JobTracker by calling the submitJob() function.



**Task Assignment** The JobTracker creates one map task for each computed input split by the user node and assigns the map tasks to the execution slots of the TaskTrackers. The JobTracker considers the localization of the data when assigning the map tasks to the TaskTrackers. The JobTracker also creates reduce tasks and assigns them to the TaskTrackers. 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.

### ⇒ Mapping Applications to Parallel and Distributed Systems:

Mapping applications to different hardware and software in terms of five application architectures, these initial five categories are listed in Table.

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 <a href="#">Section 3.5</a> .	Grids of clusters
6	MapReduce++ (Twister)	This describes file (database) to file (database) operations which have three subcategories (see also <a href="#">Table 6.11</a> ): 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

**Category 1** was popular 20 years ago, but is no longer significant. It describes applications that can be parallelized with lock-step operations controlled by hardware, run on SIMD (single-instruction and multiple-data) machines.

**Category 2** is now much more important and corresponds to a SPMD (single-program and multiple data) model running on MIMD (multiple instruction multiple data) machines. Here, each decomposed and includes dynamic irregular cases with complex geometries for solving partial differential equations or particle dynamics. Also note that category 2 consists of compute–communicate phases and the computations are synchronized by communication. No additional synchronization is needed.

**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. Shared memory is natural due to the low latency often needed to perform dynamic synchronization. It wasn’t clear in the past, but now it appears that this category is not very common in large-scale parallel problems of importance.

**Category 4** is the simplest algorithmically, with disconnected parallel components. Both grids and clouds are very natural for this class, which does not need high-performance communication between different nodes.

**Category 5** refers to the coarse-grained linkage of different “atomic” problems, Grids or clouds are suitable for metaproblems as coarse grained decomposition does not usually require stringent performance.

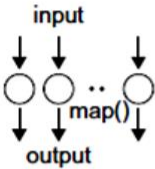
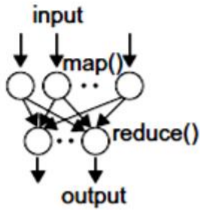
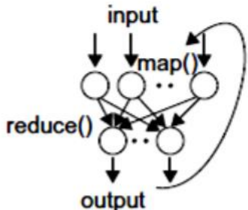
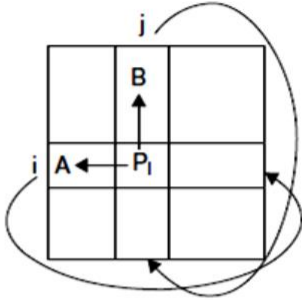
we added a sixth category to cover data-intensive applications motivated by the clear importance of MapReduce as a new programming model. We call this category MapReduce++, and it has three subcategories: “map only” applications similar to pleasingly parallel category 4.

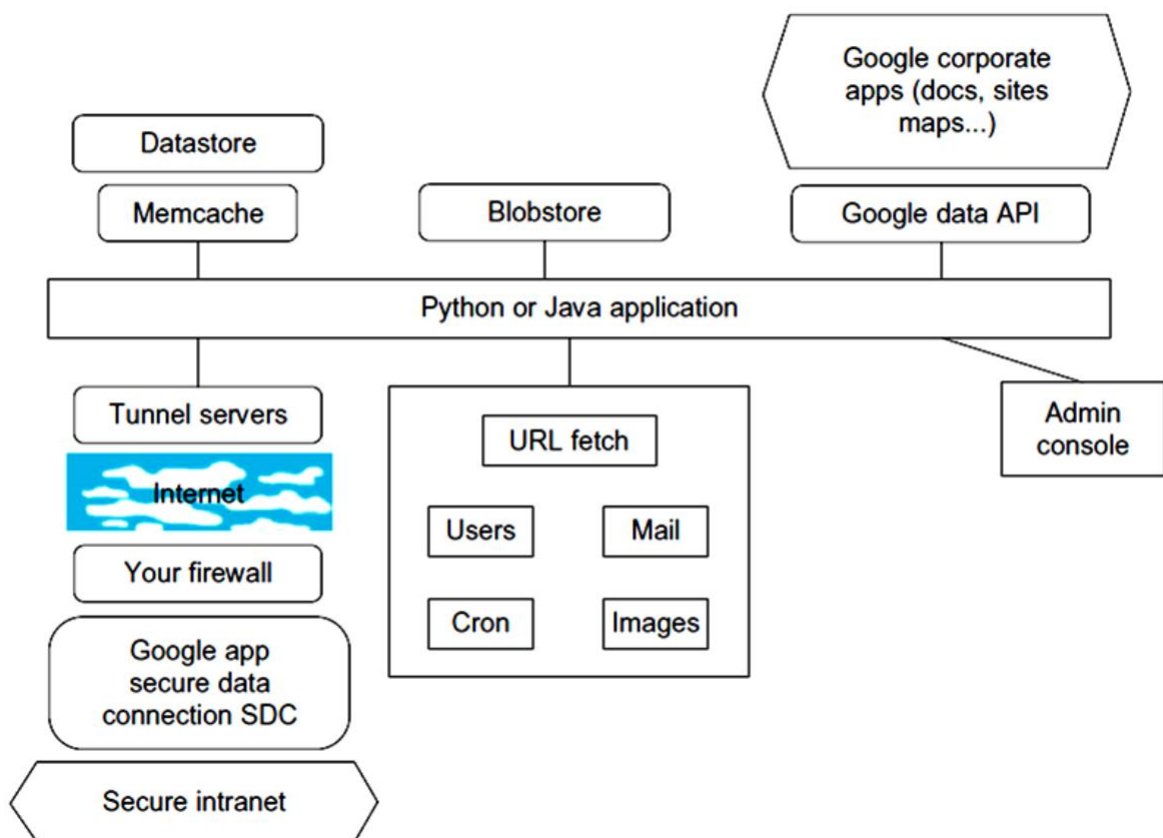
## ⇒ **Programming Support of Google App Engine**

### ➤ **Programming the Google App Engine**

- Below Figure summarizes some key features of GAE programming model for two supported languages: Java and Python.
- There are several powerful constructs for storing and accessing data. The data store is a NOSQL data management system for entities that can be, at most, 1 MB in size and are labeled by a set of schema-less properties. Queries can retrieve entities of a given kind filtered and sorted by the values of the properties. Java offers Java Data Object (JDO) and Java Persistence API (JPA) interfaces implemented by the open source Data Nucleus Access platform, while Python has a SQL-like query language called GQL. The data store is strongly consistent and uses optimistic concurrency control.
- The data store implements transactions across its distributed network using “entity groups.” A transaction manipulates entities within a single group. Entities of the same group are stored together for efficient execution of transactions. Your GAE application can assign entities to groups when the entities are created. The performance of the data store can be enhanced by in-memory caching using the memcache, which can also be used independently of the data store.

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



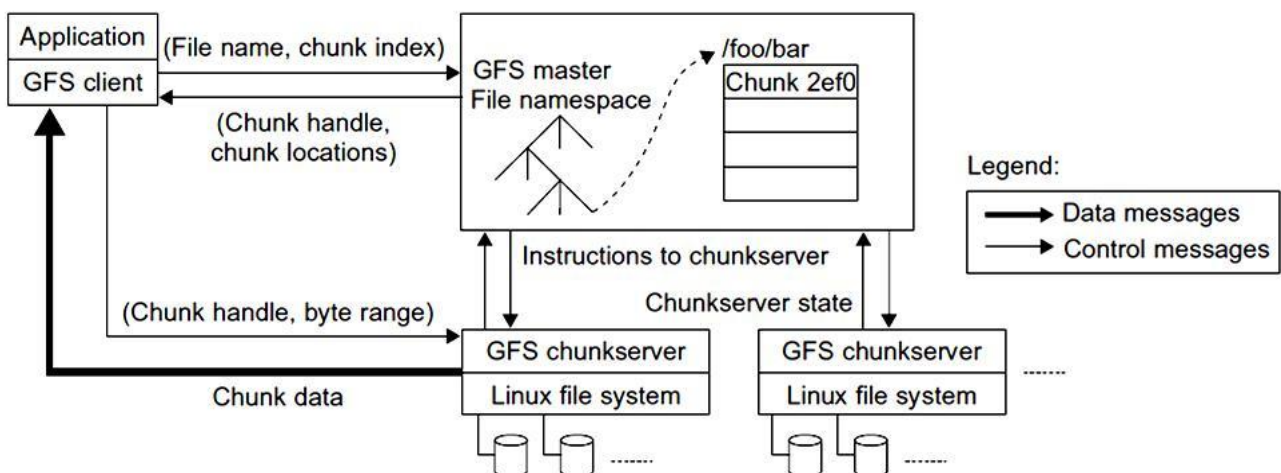
**Figure: Programming environment for Google App Engine.**



- Google added the blobstore which is suitable for large files as its size limit is 2 GB. There are several mechanisms for incorporating external resources. The Google SDC Secure Data Connection can tunnel through the Internet and 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. There is a specialized mail mechanism to send e-mail from your GAE application.
- Applications can access resources on the Internet, such as web services or other data, using GAE's URL fetch service. The URL fetch service retrieves web resources using the same highspeed Google infrastructure that retrieves web pages for many other Google products. There are dozens of Google "corporate" facilities including maps, sites, groups, calendar, docs, and YouTube, among others. These support the Google Data API which can be used inside GAE.
- Applications can access resources on the Internet, such as web services or other data, using GAE's URL fetch service. The URL fetch service retrieves web resources using the same highspeed Google infrastructure that retrieves web pages for many other Google products. There are dozens of Google "corporate" facilities including maps, sites, groups, calendar, docs, and YouTube, among others. These support the Google Data API which can be used inside GAE.

### ➤ Google File System (GFS)

- GFS typically will hold a large number of huge files, each 100MB or larger, with files that are multiple GB in size quite common. Thus, Google has chosen its file data block size to be 64MB instead of the 4 KB in typical traditional file systems.
- Other nodes act as the chunk servers for storing data, while the single master stores the metadata. The file system namespace and locking facilities are managed by the master. The master periodically communicates with the chunk servers to collect management information as well as give instructions to the chunk servers to do work such as load balancing or fail recovery.



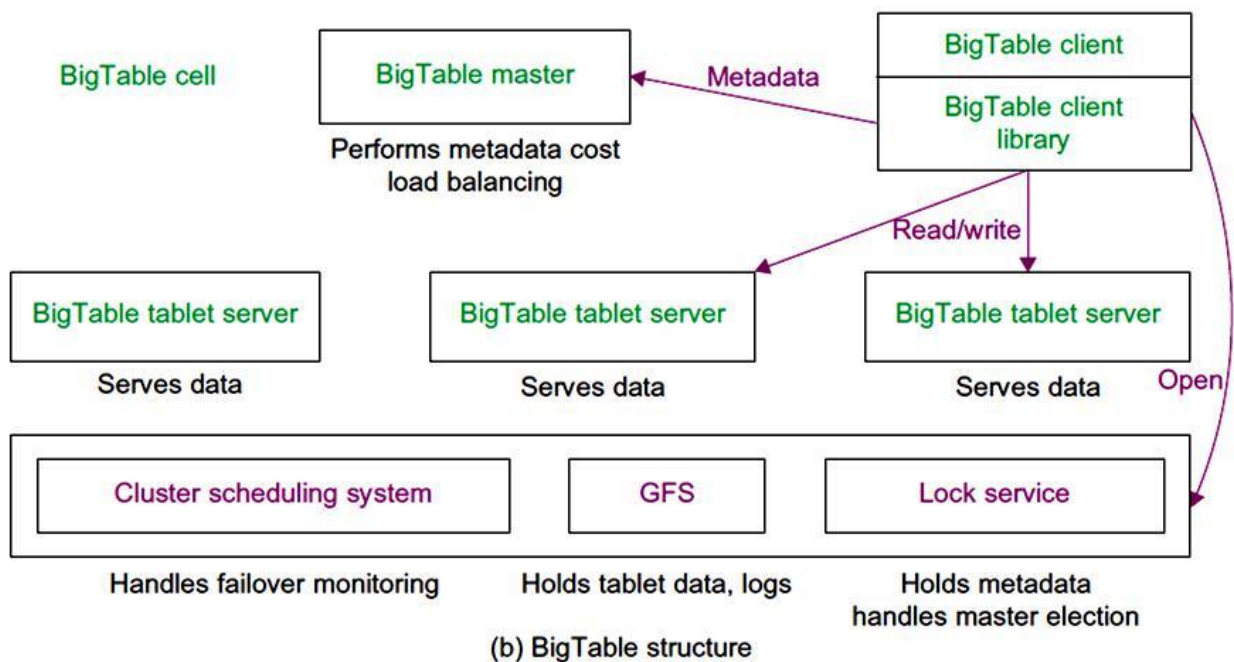
**Figure: Architecture of Google File System (GFS).**

### ➤ BigTable, Google's NOSQL System

- Big Table was designed to provide a service for storing and retrieving structured and semi structured data.



- Big Table applications include storage of web pages, per-user data, and geographic locations. Here we use web pages to represent URLs and their associated data, such as contents, crawled metadata, links, anchors, and page rank values.
- BigTable can be viewed as a distributed multilevel map. It provides a fault-tolerant and persistent database as in a storage service. The BigTable system is scalable, which means the system has thousands of servers, terabytes of in-memory data, petabytes of disk-based data, millions of reads/writes per second, and efficient scans.
- BigTable is used in many projects, including Google Search, Orkut, and Google Maps/Google Earth, among others. One of the largest BigTable cell manages ~200 TB of data spread over several thousand machines.



Large tables are broken into tablets at row boundaries. A tablet holds a contiguous range of rows. Clients can often choose row keys to achieve locality. The system aims for about 100MB to 200MB of data per tablet.

### ➤ 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.

### ⇒ Programming on Amazon AWS :

we will consider the programming support in the AWS platform. The Elastic MapReduce capability is equivalent to Hadoop running on the basic EC2 offering. Amazon has NOSQL support. Amazon offers the Simple Queue Service (SQS) and Simple Notification Service (SNS).

### ➤ 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.

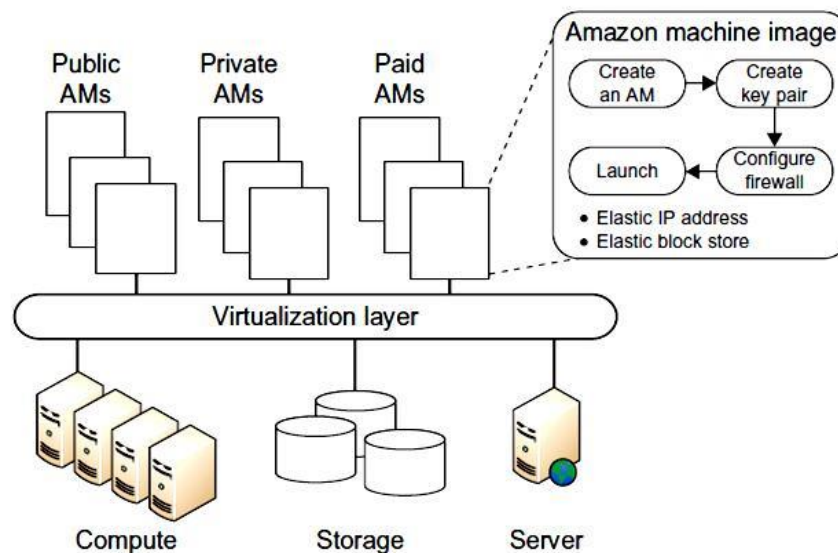
AMIs are the templates for instances, which are running VMs. The workflow to create a VM is Create an AMI→Create Key Pair→Configure Firewall→Launch.

### Three Types of AMI

**Private AMI Images** created by you, which are private by default. You can grant access to other users to launch your private images.

**Public AMI Images** created by users and released to the AWS community, so anyone can launch instances based on them and use them any way they like.

**Paid QAMI You can create images** providing specific functions that can be launched by anyone willing to pay you per each hour of usage on top of Amazon's charges.



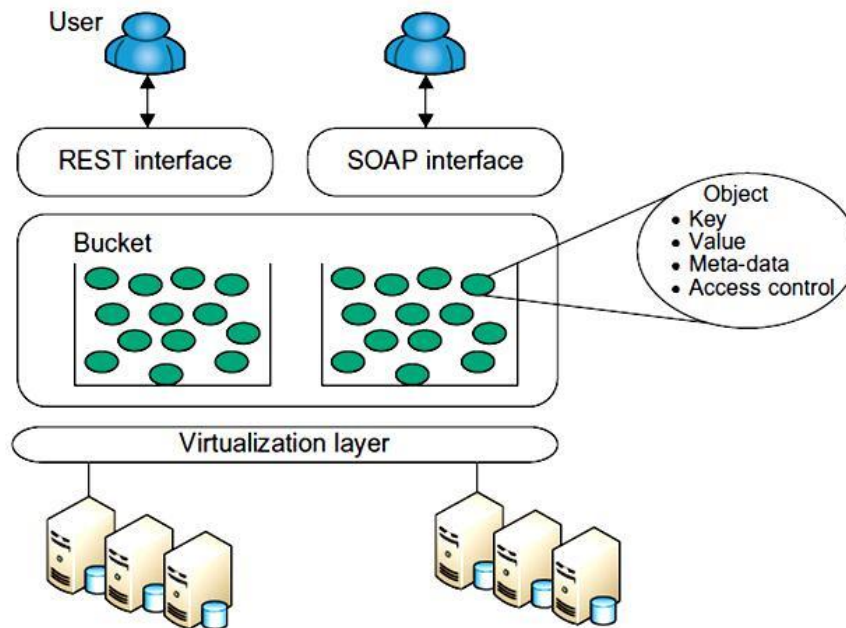
**Figure: Amazon EC2 execution environment.**

Both auto-scaling and elastic load balancing are enabled by CloudWatch which monitors running instances. CloudWatch is a web service that provides monitoring for AWS cloud resources, starting with Amazon EC2. It provides customers with visibility into resource utilization, operational performance, and overall demand patterns—including metrics such as CPU utilization, disk reads and writes, and network traffic.

### ➤ 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. SQS is responsible for ensuring a reliable message service between two processes, even if the receiver processes are not running.

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.



**Figure: Amazon S3 execution environment**

**Here are some key features of S3:**

- Redundant through geographic dispersion
- Designed to provide 99.999999999 percent durability and 99.99 percent availability of objects over a given year with cheaper reduced redundancy storage (RRS).
- 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).
- Default download protocol of HTTP. A BitTorrent protocol interface is provided to lower costs for high-scale distribution.
- 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 within the same region.

### ➤ **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. Note that S3 is “Storage as a Service” with a messaging interface. EBS allows you to create storage volumes from 1 GB to 1 TB that can be mounted as EC2 instances.

### ➤ **Amazon SimpleDB Service**

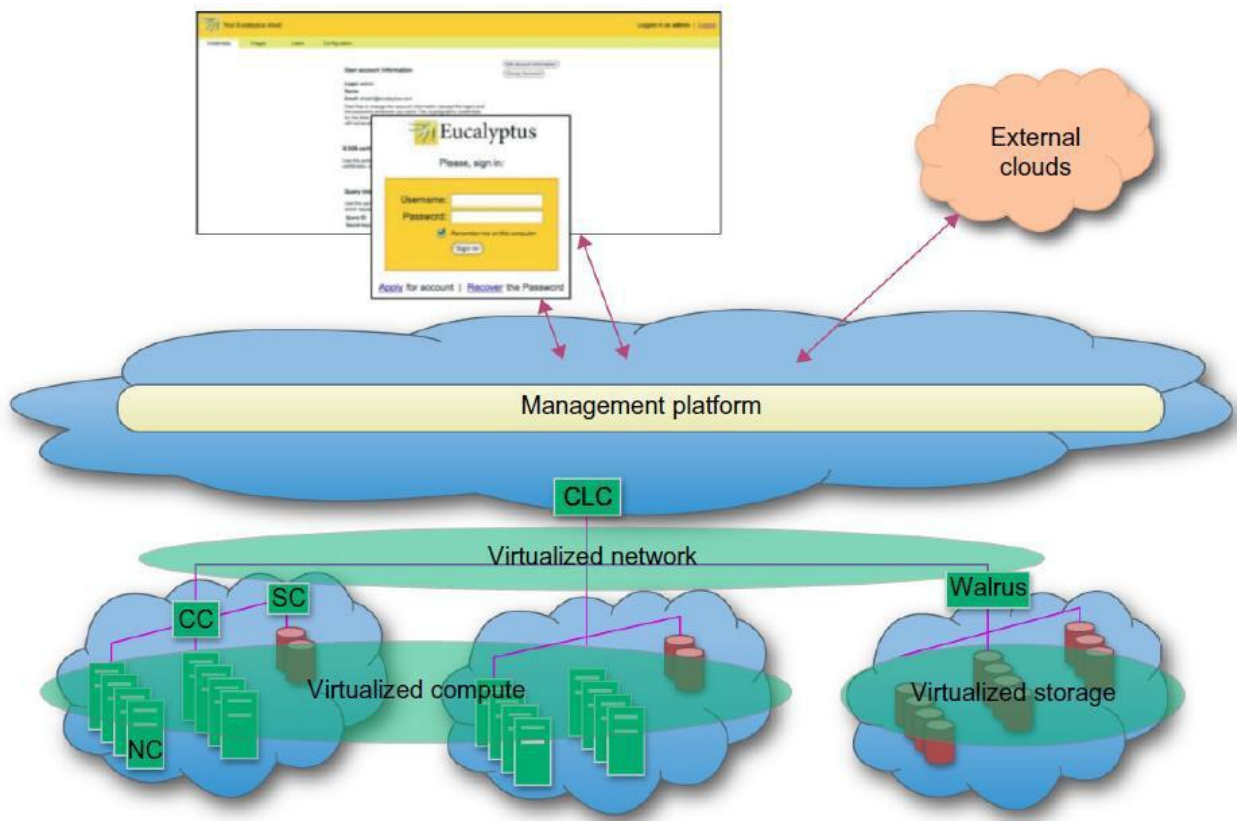
SimpleDB provides a simplified data model based on the relational database data model. Structured data from users must be organized into domains. Each domain can be considered a table. The items are the rows in the table.

## ⇒ Emerging Cloud Software Environments

We will assess popular cloud operating systems and emerging software environments. We cover the open source Eucalyptus, OpenNebula, Open Stack , and ClodSim.

### ➤ Open Source Eucalyptus

- 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 Architecture: The Eucalyptus system is an open software environment. Figure shows the architecture based on the need to manage VM images. The system supports cloud programmers in VM image management as follows. Essentially, the system has been extended to support the development of both the computer cloud and storage cloud.
- VM Image Management: Eucalyptus stores images in Walrus, the block storage system that is analogous to the Amazon S3 service. 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.

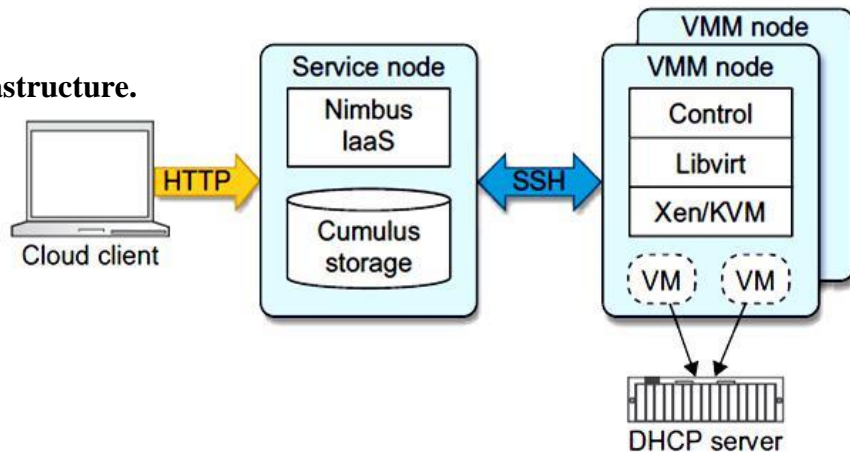


**Figure: The Eucalyptus architecture for VM image management.**

- **Nimbus:** Nimbus is a set of open source tools that together provide an IaaS cloud computing solution. Nimbus is a set of open source tools that together provide an IaaS cloud computing solution. A storage cloud implementation called Cumulus has been tightly integrated with the other central services, although it can also be used stand-alone.



Figure: Nimbus cloud infrastructure.



### ➤ OpenNebula

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

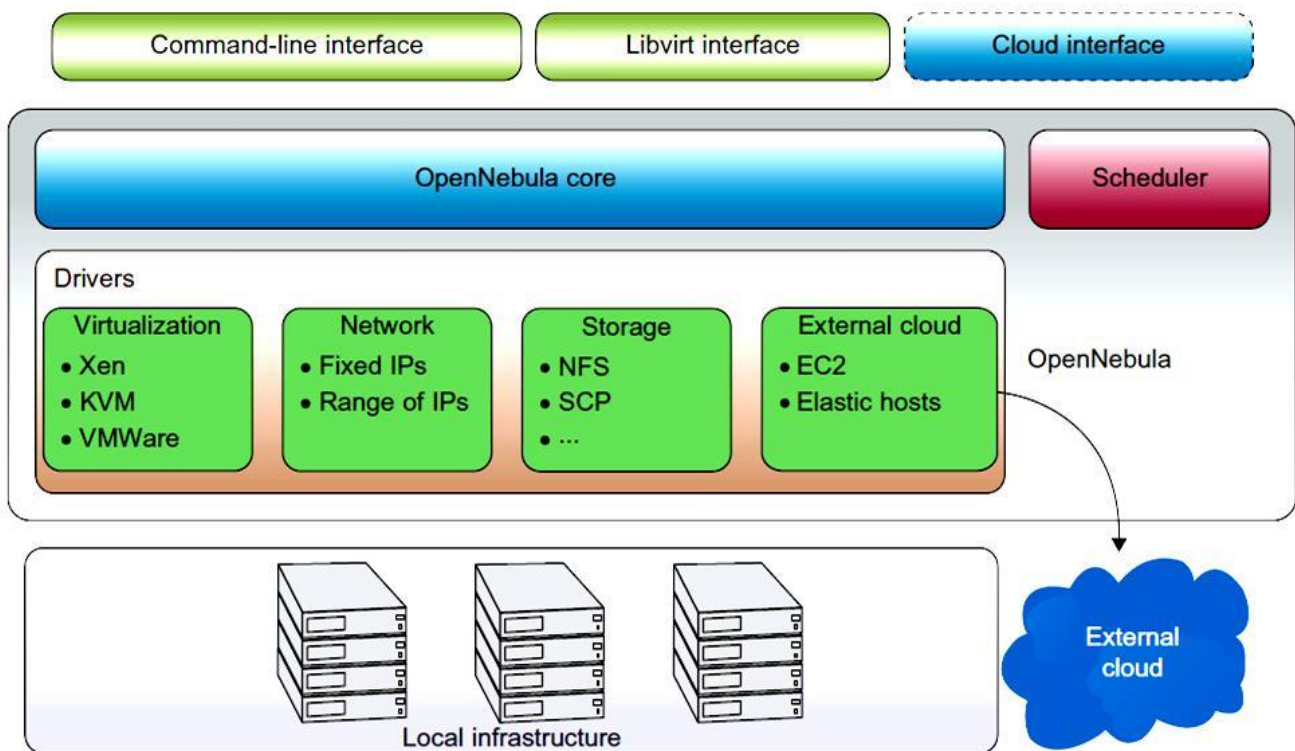


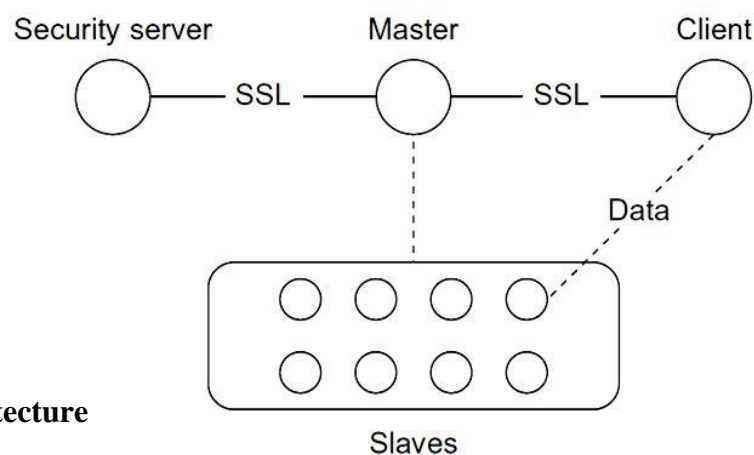
Figure: OpenNebula architecture and its main components

- 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.
- 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 access drivers , provide an abstraction of the underlying infrastructure to expose the basic functionality of the monitoring, storage, and virtualization services available in the cluster.
- OpenNebula can support a hybrid cloudmodel by using cloud drivers to interface with external clouds. Regarding storage, an Image Repository allows users to easily specify disk images from a catalog without worrying about low-level disk configuration attributes or block device mapping.

### ➤ **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. 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 connection
- Since Sector is aware of the network topology when it places replicas, it also provides better reliability, availability, and access throughout. The communication is performed using User Datagram Protocol (UDP) for message passing and user-defined type (UDT) for data transfer.



**The Sector/Sphere system architecture**

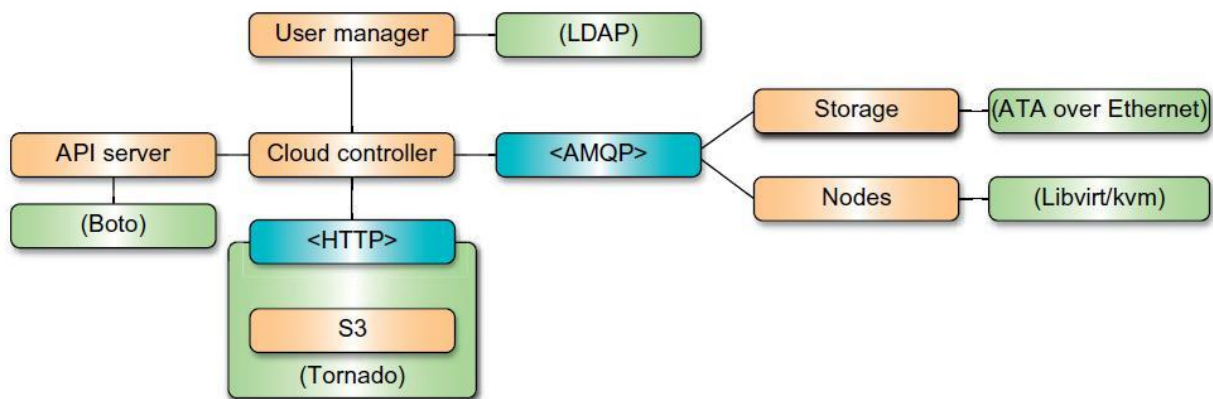
- Sphere is a parallel data processing engine designed to work with data managed by Sector. This coupling allows the system to make accurate decisions about job scheduling and data location. Sphere provides a programming framework that developers can use to process data stored in Sector.
- The first component is the security server, which is responsible for authenticating master servers, slave nodes, and users. The last component is the client component.

### ➤ **OpenStack**

- OpenStack focuses on the development of two aspects of cloud computing to address compute and storage aspects with the OpenStack Compute and OpenStack Storage solutions. “OpenStack Compute is the internal fabric of the cloud creating and managing large groups of virtual private servers” and “OpenStack Object Storage is software for creating redundant, scalable object storage using clusters of commodity servers to store terabytes or even petabytes of data.”

## 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.
  - 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. Nova integrates networking components to manage private networks, public IP addressing, virtual private network (VPN) connectivity, and firewall rules. It includes the following types:
    - NetworkController manages address and virtual LAN (VLAN) allocations
    - RoutingNode governs the NAT (network address translation) conversion of public IPs to private IPs, and enforces firewall rules
    - AddressingNode runs Dynamic Host Configuration Protocol (DHCP) services for private Networks
    - TunnelingNode provides VPN connectivity
- The network state (managed in the distributed object store) consists of the following:
- VLAN assignment to a project
  - Private subnet assignment to a security group in a VLAN
  - Private IP assignments to running instances
  - Public IP allocations to a project
  - Public IP associations to a private IP/running instance



**Figure: OpenStack Nova system architecture. The AMQP (Advanced Messaging Queuing Protocol)**

## Open Stack 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. A ring represents a mapping between the names of entities stored on disk and their physical locations. Objects are stored as binary files with metadata stored in the file's extended attributes.

---

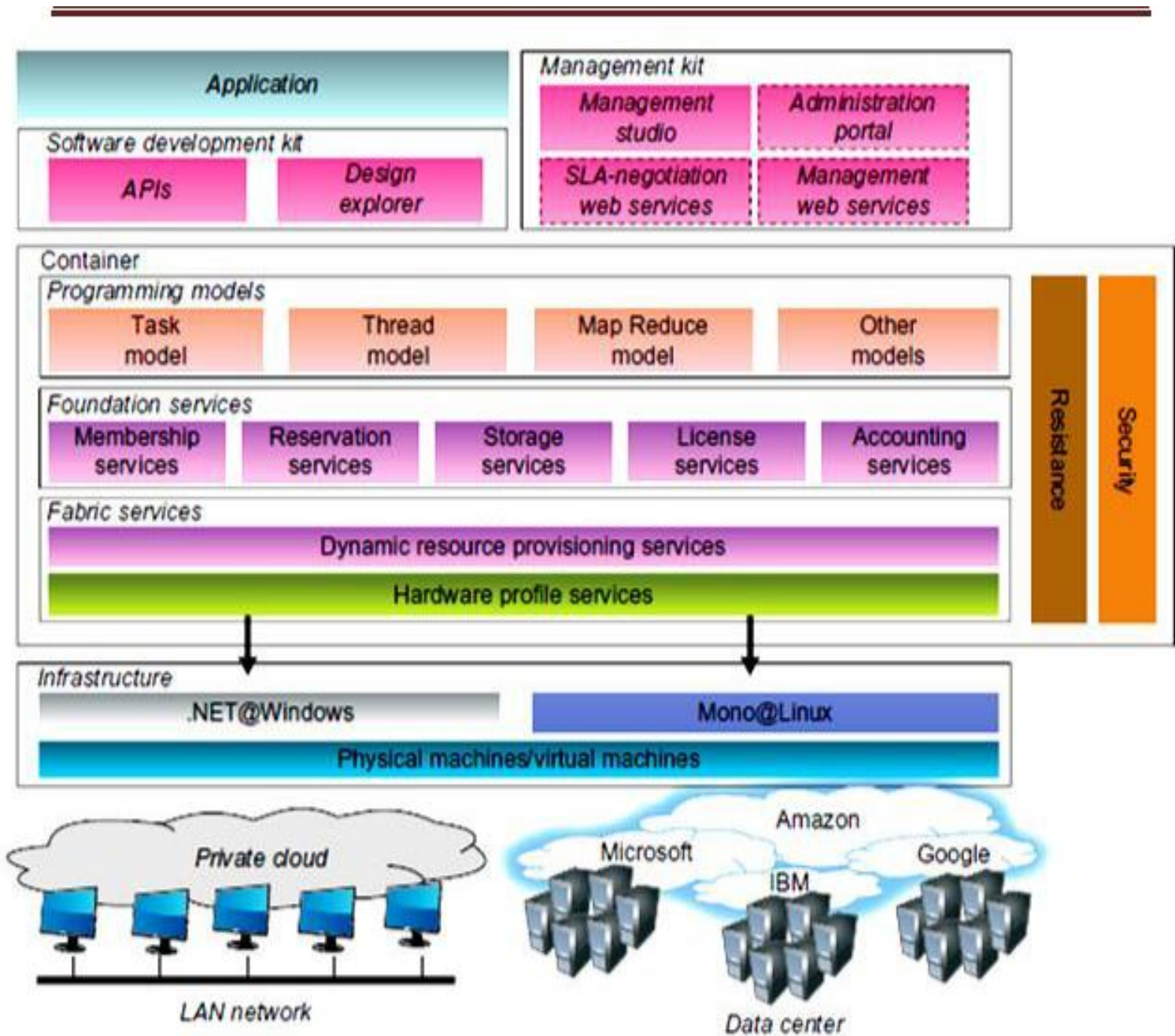
## ➤ Manjrasoft Aneka Cloud

- Aneka 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.
- Aneka acts as a workload distribution and management platform for accelerating applications in both Linux and Microsoft .NET framework environments. Some of the key advantages of Aneka over other workload distribution solutions include:
  - 1) Support of multiple programming and application environments
  - 2) Simultaneous support of multiple runtime environments
  - 3) Rapid deployment tools and framework
  - 4) 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
  - 5) Built on top of the Microsoft .NET framework, with support for Linux environments
- 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.
  - 2) **Accelerate** Aneka supports rapid development and deployment of applications in multiple runtime environments running different operating systems such as Windows or Linux/UNIX. Aneka uses physical machines as much as possible to achieve maximum utilization in local environments.
  - 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. Aneka also has an accounting mechanism and manages priorities and scalability based on SLA/QoS which enables dynamic provisioning.
- Here are three important programming models supported by Aneka for both cloud and traditional parallel applications:
  - 1) Thread programming model, best solution to adopt for leveraging the computing capabilities of multicore nodes in a cloud of computers
  - 2) Task programming model, which allows for quickly prototyping and implementing an independent bag of task applications
  - 3) MapReduce programming model, as discussed

### **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 interaction with the hosting platform is mediated through the Platform Abstraction Layer (PAL), which hides in its implementation all the heterogeneity of the different operating systems.





**Figure: Architecture and components of Aneka.**

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. These services provide added value to the infrastructure and are of use to system administrators and developers. Within this category we can list storage management, resource reservation, reporting, accounting, billing, services monitoring, and licensing. Services in this level operate across the range of supported application models.
- ❖ **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. They leverage

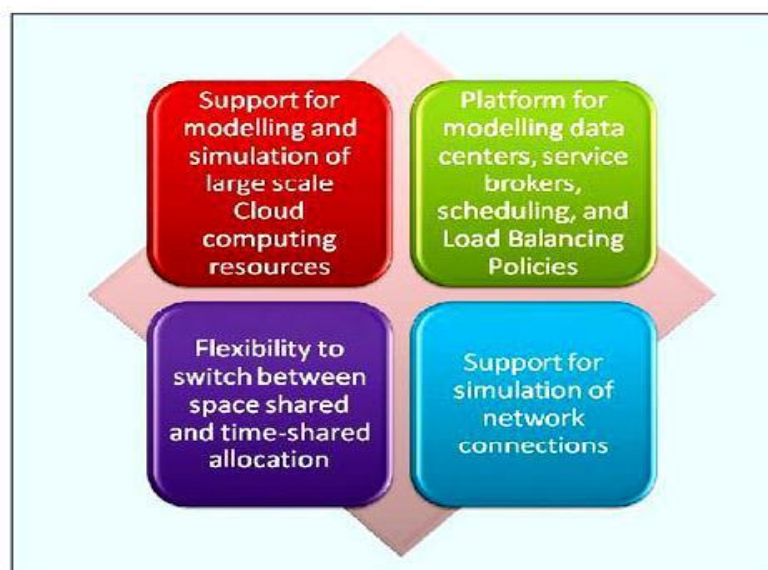
foundation and fabric services for several tasks of application execution such as elastic scalability, data transfer, and performance monitoring, accounting, and billing. At this level, Aneka expresses its true potential in supporting different application models and distributed programming patterns.

### **CloudSim**

- **CloudSim** is a library for the simulation of cloud scenarios. It provides essential classes for describing data centres, computational resources, virtual machines, applications, users, and policies for the management of various parts of the system such as scheduling and provisioning.
- CloudSim provides a generalised and extensible simulation framework that enables seamless modelling and simulation of app performance. By using CloudSim, developers can focus on specific systems design issues that they want to investigate, without getting concerned about details related to cloud-based infrastructures and services.

#### **An introduction to CloudSim**

- ✓ CloudSim is a simulation tool that allows cloud developers to test the performance of their provisioning policies in a repeatable and controllable environment, free of cost.
- ✓ Using these components, it is easy to evaluate new strategies governing the use of clouds, while considering policies, scheduling algorithms, load balancing policies, etc.
- ✓ It can also be used to assess the competence of strategies from various perspectives such as cost, application execution time, etc. It also supports the evaluation of Green IT policies. It can be used as a building block for a simulated cloud environment and can add new policies for scheduling, load balancing and new scenarios.
- ✓ It is flexible enough to be used as a library that allows you to add a desired scenario by writing Java program.
- ✓ By using CloudSim, organisations, R&D centres and industry-based developers can test the performance of a newly developed application in a controlled and easy to set-up environment. The prominent features offered by CloudSim are given in Figure 1.



## **Architecture of CloudSim**

The CloudSim layer provides support for modelling and simulation of cloud environments including dedicated management interfaces for memory, storage, bandwidth and VMs. It also provisions hosts to VMs, application execution management and dynamic system state monitoring. A cloud service provider can implement customised strategies at this layer to study the efficiency of different policies in VM provisioning.

The user code layer exposes basic entities such as the number of machines, their specifications, etc, as well as applications, VMs, number of users, application types and scheduling policies.

The main components of the CloudSim framework,

**Regions:** It models geographical regions in which cloud service providers allocate resources to their customers. In cloud analysis, there are six regions that correspond to six continents in the world.

**Data centres:** It models the infrastructure services provided by various cloud service providers. It encapsulates a set of computing hosts or servers that are either heterogeneous or homogeneous in nature, based on their hardware configurations.

**Data centre characteristics:** It models information regarding data centre resource configurations.

**Hosts:** It models physical resources (compute or storage).

**The user base:** It models a group of users considered as a single unit in the simulation, and its main responsibility is to generate traffic for the simulation.

**Cloudlet:** It specifies the set of user requests. It contains the application ID, name of the user base that is the originator to which the responses have to be routed back, as well as the size of the request execution commands, and input and output files. It models the cloud-based application services. CloudSim categorises the complexity of an application in terms of its computational requirements. Each application service has a pre-assigned instruction length and data transfer overhead that it needs to carry out during its life cycle.

**Service broker:** The service broker decides which data centre should be selected to provide the services to the requests from the user base.

**VMM allocation policy:** It models provisioning policies on how to allocate VMs to hosts.

**VM scheduler:** It models the time or space shared, scheduling a policy to allocate processor cores to VMs.