UNIT- II

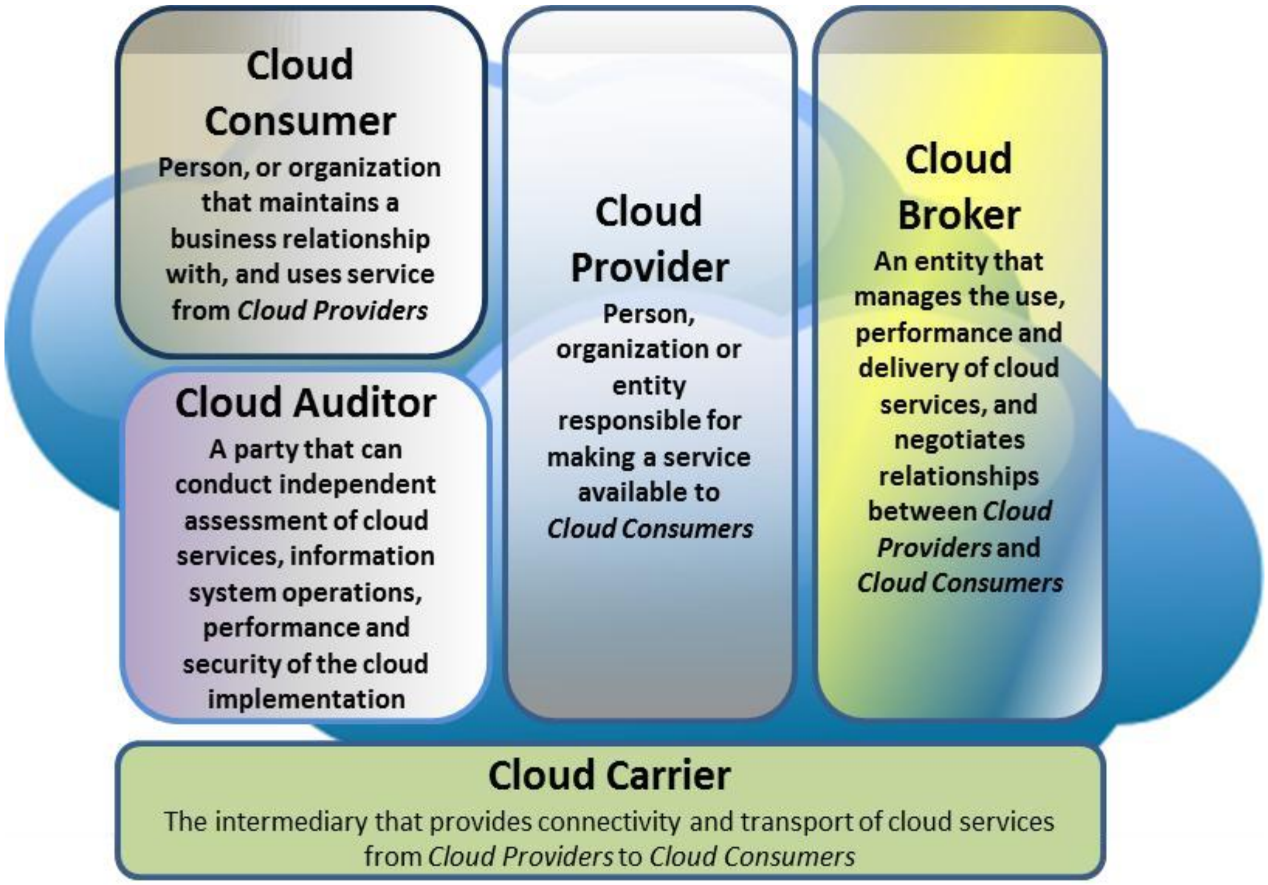
CLOUD REFERENCE MODEL

The NIST cloud computing reference architecture presented in this section is a natural extension to the NIST cloud computing definition.

The NIST cloud computing reference architecture is a generic high-level conceptual model that is a powerful tool for discussing the requirements, structures, and operations of cloud computing. The model is not tied to any specific vendor products, services, or reference implementation. It defines a set of actors, activities, and functions that can be used in the process of developing cloud computing architectures. The NIST cloud computing reference architecture focuses on the requirements of what cloud service provides, not on a design that defines a solution and its implementation.

The Overview of the Reference Architecture describes five major actors with their roles and responsibilities using the newly developing Cloud Computing. The NIST cloud computing reference architecture defines five major actors: cloud consumer, cloud provider, cloud auditor, cloud broker, and cloud carrier. These core individuals have key roles in the realm of cloud computing. Each actor is an entity (a person or an organization) that participates in a transaction or process and/or performs tasks in cloud computing.

Figure **Cloud Actors** briefly lists the five major actors defined in the NIST cloud computing reference architecture

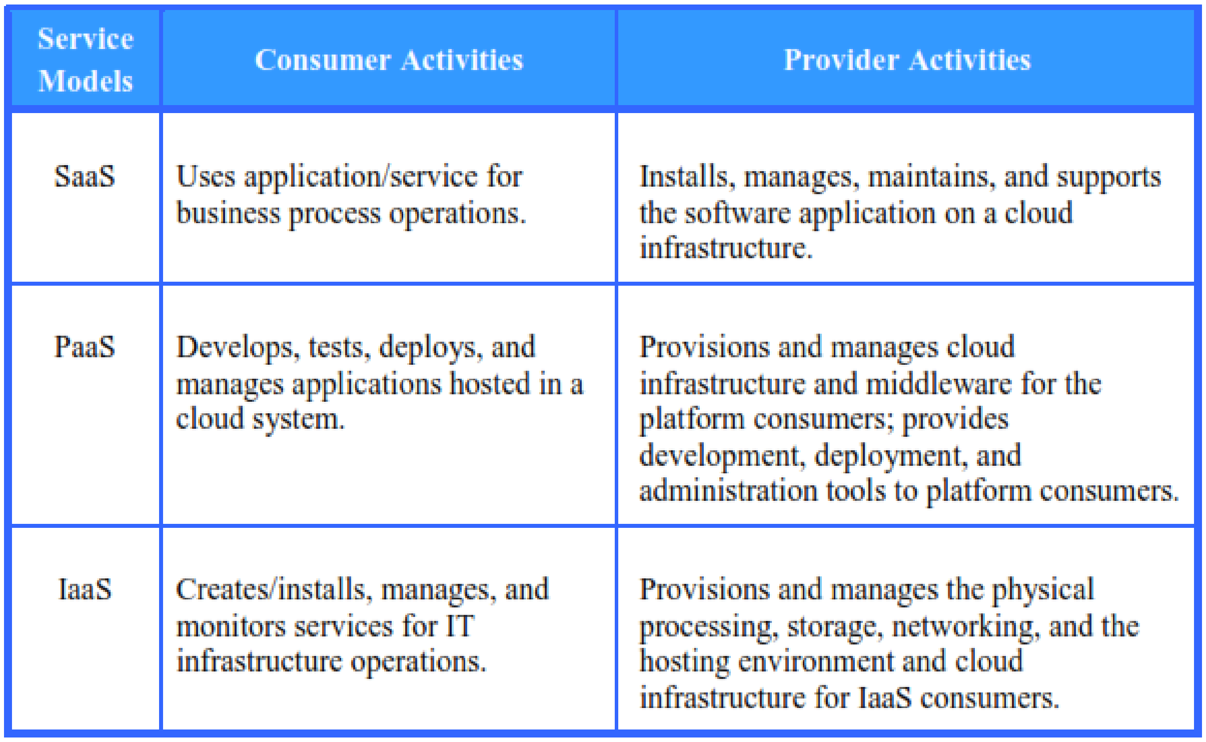

Cloud Consumer
Person, or organization that maintains a business relationship with, and uses service from *Cloud Providers*
Cloud Auditor
A party that can conduct independent assessment of cloud services, information system operations, performance and security of the cloud implementation
Cloud Provider
Person, organization or entity responsible for making a service available to *Cloud Consumers*
Cloud Broker
An entity that manages the use, performance and delivery of cloud services, and negotiates relationships between *Cloud Providers* and *Cloud Consumers*
Cloud Carrier
The intermediary that provides connectivity and transport of cloud services from *Cloud Providers* to *Cloud Consumers*

CLOUD CONSUMER

The cloud consumer is the ultimate stakeholder that the cloud computing service is created to support. A cloud consumer represents a person or organization that maintains a business relationship with, and uses the service from, a cloud provider. A cloud consumer browses the service catalog from a cloud provider, requests the appropriate service, sets up service contracts with the cloud provider, and uses the service. The cloud consumer may be billed for the service provisioned, and needs to arrange payments accordingly. Depending on the services requested, the activities and usage scenarios can be different among cloud consumers.

CLOUD PROVIDER

A cloud provider can be a person, an organization, or an entity responsible for making a service available to cloud consumers. A cloud provider builds the requested software/platform/ infrastructure services, manages the technical infrastructure required for providing the services, provisions the services at agreed-upon service levels, and protects the security and privacy of the services. Cloud Provider: Major Activities, cloud providers undertake different tasks for the provisioning of the various service models.



Service Models	Consumer Activities	Provider Activities
SaaS	Uses application/service for business process operations.	Installs, manages, maintains, and supports the software application on a cloud infrastructure.
PaaS	Develops, tests, deploys, and manages applications hosted in a cloud system.	Provisions and manages cloud infrastructure and middleware for the platform consumers; provides development, deployment, and administration tools to platform consumers.
IaaS	Creates/installs, manages, and monitors services for IT infrastructure operations.	Provisions and manages the physical processing, storage, networking, and the hosting environment and cloud infrastructure for IaaS consumers.

For SaaS, the cloud provider deploys, configures, maintains, and updates the operation of the software applications on a cloud infrastructure so that the services are provisioned at the expected service levels to cloud consumers. The provider of SaaS assumes most of the responsibilities in managing and controlling the applications and the infrastructure, while the cloud consumers have limited administrative control of the applications.

For PaaS, the cloud provider manages the cloud infrastructure for the platform, and provisions tools and execution resources for the platform consumers to develop, test, deploy, and administer applications. Consumers have control over the applications and possibly the hosting environment settings, but cannot access the infrastructure underlying the platform including network, servers, operating systems, or storage.

For IaaS, the cloud provider provisions the physical processing, storage, networking, and other fundamental computing resources, as well as manages the hosting environment and cloud infrastructure for IaaS consumers. Cloud consumers deploy and run applications, have more control over the hosting environment and operating systems, but do not manage or control the underlying cloud infrastructure (e.g., the physical servers, network, storage, hypervisors, etc.).

The activities of cloud providers can be discussed in greater detail from the perspectives of *Service Deployment*, *Cloud Service Management*, *Security* and *Privacy*.

SERVI CE DEPLOYMENT

As per the NIST cloud computing definition, a cloud infrastructure may be operated in one of the following deployment models: *public cloud*, *private cloud*, *community cloud*, or *hybrid cloud*.

A public cloud is one in which the cloud infrastructure and computing resources are made available to the general public over a public network. A public cloud is owned by an organization selling cloud services and serves a diverse pool of clients.

A private clouds, the cloud infrastructure is operated exclusively for a single organization. A private cloud gives the organization exclusive access to and usage of the infrastructure and computational resources. It may be managed either by the organization or by a third party, and may be implemented at the organization’s premise (i.e., *on-site private clouds*) or outsourced to a hosting company (i.e., *outsourced private clouds*).

Similar to private clouds, a community cloud may be managed by the organizations or by a third party, and may be implemented at the customer's location (i.e., *on-site community cloud*) or outsourced to a hosting company (i.e., *outsourced community cloud*). However, a community cloud serves a set of organizations that have common security, privacy, and compliance considerations, rather than serving a single organization as does a private cloud.

A hybrid cloud is a composition of two or more cloud deployment models (private, community, or public) that remain unique entities but are bound together by standardized or proprietary technology that enables data and application portability.

Cloud Service Management includes all of the service-related functions that are necessary for the management and operation of those services required by or proposed to cloud consumers. As illustrated in Figure, cloud service management can be described from the perspective of *business support, provisioning and configuration*, and from the perspective of *portability and interoperability* requirements.

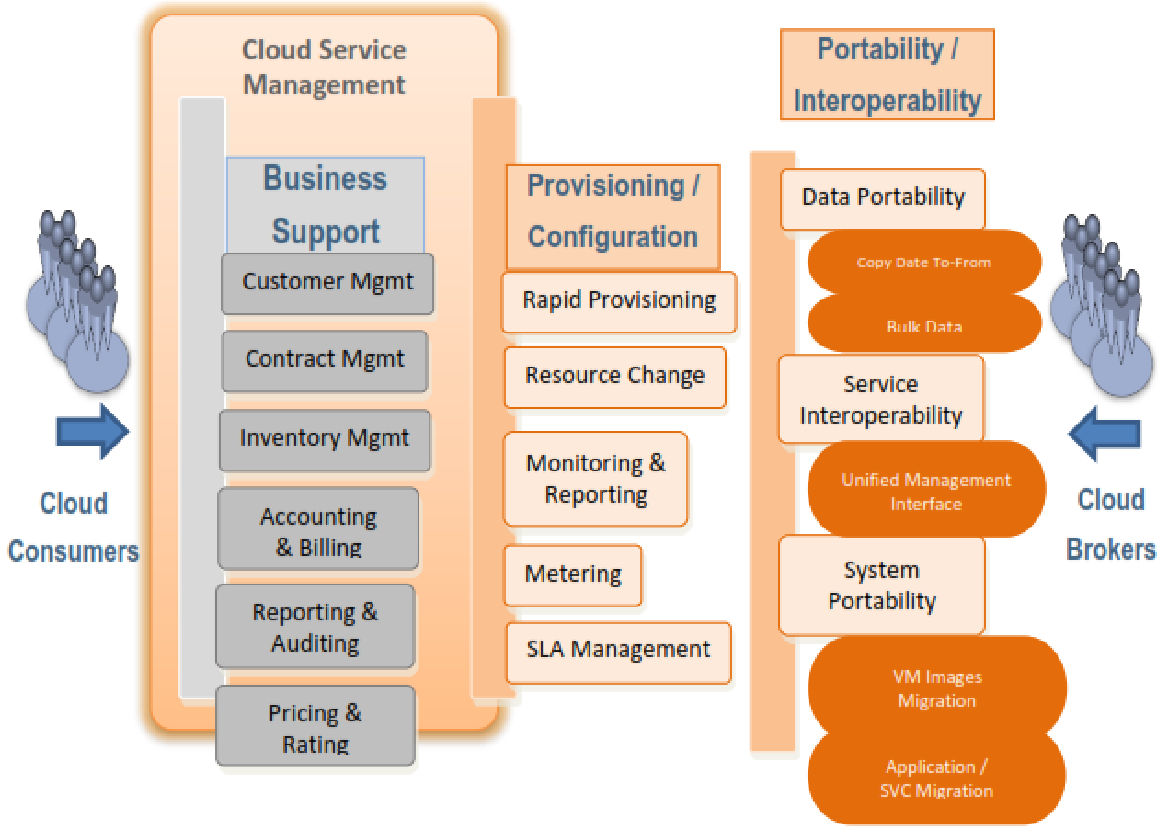


Figure 6 – Cloud Provider: Cloud Service Management

SECURITY & PRIVACY

Cloud providers should protect the assured, proper, and consistent collection, processing, communication, use, and disposition of personal information (PI) and personally identifiable information (PII) in the cloud system. PII is the information that can be used to distinguish or trace an individual's identity, such as name, social security number, biometric records, etc., alone, or when combined with other personal or identifying information that is linked or linkable to a specific individual, such as date and place of birth, mother's maiden name, etc. Though cloud computing provides a flexible solution for shared resources, software, and information, it also poses additional privacy challenges to consumers using the clouds.

CLOUD AUDITOR

A cloud auditor is a party that can conduct independent assessment of cloud services, information system operations, performance, and the security of a cloud computing implementation. A cloud auditor can evaluate the services provided by a cloud provider in terms of security controls, privacy impact, performance, and adherence to service level agreement parameters. Auditing is especially important for federal agencies as "agencies should include a contractual section enabling third parties to assess security controls of cloud providers.

Security controls are the management, operational, and technical safeguards or countermeasures employed within an organizational information system to protect the confidentiality, integrity, and availability of the system and its information. For security auditing, a cloud auditor can make an assessment of the security controls in the information system to determine the extent to which the controls are implemented correctly, operating as intended, and producing the desired outcome with respect to the security requirements for the system. The security auditing should include the verification of the compliance with regulation and security policy.

CLOUD BROKER

A Cloud Broker as an entity that manages the use, performance, and delivery of cloud services, and negotiates relationships between Cloud Providers and Cloud Consumers. As cloud computing evolves, the integration of cloud services may become too complex for cloud Consumers to manage. In such cases, a Cloud Consumer may request cloud services from a Cloud Broker instead of directly contacting a Cloud Provider.

Cloud Brokers provide a single point of entry for managing multiple cloud services. The key defining feature that distinguishes a Cloud Broker from a Cloud Service Provider is the ability to provide a single consistent interface to multiple differing providers, whether the interface is for business or technical purposes. In general, Cloud Brokers provide services in three categories:

Intermediation: A Cloud Broker enhances a given service by improving some specific capability and providing value-added services to cloud Consumers. The improvement can be managing access to cloud services, identity management, performance reporting, enhanced security, etc.

Aggregation: A Cloud Broker combines and integrates multiple services into one or more new services. The Broker provides data and service integration and ensures the secure data movement between the cloud Consumer and multiple cloud Providers.

Arbitrage: Service arbitrage is similar to service aggregation except that the services being combined/consolidated are not fixed. Service arbitrage means a Broker has the flexibility to choose services from multiple service Providers.

CLOUD CARRIER

A cloud carrier acts as an intermediary that provides connectivity and transport of cloud services between cloud consumers and cloud providers. Cloud carriers provide access to consumers through network, telecommunication, and other access devices. For example, cloud consumers can obtain cloud services through network access devices, such as computers, laptops, mobile phones, mobile Internet devices (MIDs), etc. The distribution of cloud services is normally provided by network and telecommunication carriers or a *transport agent*, where a transport agent refers to a business organization that provides physical transport of storage media such as high-capacity hard drives.

A cloud provider will set up service level agreements (SLAs) with a cloud carrier to provide services consistent with the level of SLAs offered to cloud consumers, and may require the cloud carrier to provide dedicated and encrypted connections between cloud consumers and cloud providers.

CLOUD COMPUTING ARCHITECTURE

It is possible to organize all the concrete realizations of cloud computing into a layered view covering the entire stack Figure from hardware appliances to software systems. Often, this layer is implemented using a datacenter in which hundreds and thousands of nodes are stacked together. Cloud infrastructure can be heterogeneous in nature because a variety of resources, such as clusters and even networked PCs, can be used to build it.

The physical infrastructure is managed by the core middleware, the objectives of which are to provide an appropriate runtime environment for applications and to best utilize resources. At the bottom of the stack, virtualization technologies are used to guarantee runtime environment customization, application isolation, sandboxing, and quality of service. Hardware virtualization is most commonly used at this level. Hypervisors manage the pool of resources and expose the distributed infrastructure as a collection of virtual machines. By using virtual machine technology it is possible to finely partition the hardware resources such as CPU and memory and to virtualize specific devices, thus meeting the requirements of users and applications.

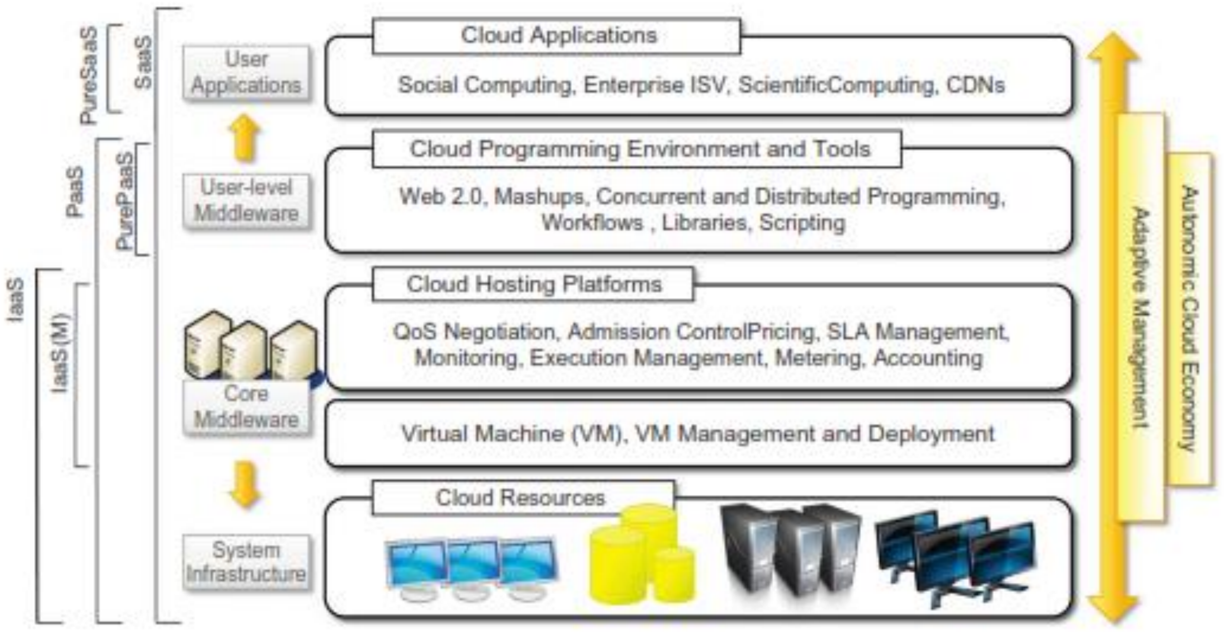

PureSaaS
SaaS
PaaS
PurePaaS
IaaS
IaaS(M)
User Applications
User-level Middleware
Core Middleware
System Infrastructure
Cloud Applications
Social Computing, Enterprise ISV, ScientificComputing, CDNs
Cloud Programming Environment and Tools
Web 2.0, Mashups, Concurrent and Distributed Programming, Workflows , Libraries, Scripting
Cloud Hosting Platforms
QoS Negotiation, Admission ControlPricing, SLA Management, Monitoring, Execution Management, Metering, Accounting
Virtual Machine (VM), VM Management and Deployment
Cloud Resources
Adaptive Management
Autonomic Cloud Economy

Fig. Cloud computing Architecture

The combination of cloud hosting platforms and resources is generally classified as a Infrastructure-as-a-Service (IaaS) solution. IaaS solutions are suitable for designing the system infrastructure but provide limited services to build applications. Such service is provided by cloud programming environments and tools, which form a new layer for offering users a development platform for applications. The range of tools includes Web-based interfaces, command-line tools, and frameworks for concurrent and distributed programming. In this scenario, users develop their applications specifically for the cloud by using the API exposed at the user-level middleware. For this reason, this approach is also known as Platform-as-a-Service (PaaS) because the service offered to the user is a development platform rather than an infrastructure. PaaS solutions generally include the infrastructure as well, which is bundled as part of the service provided to users. In the case of Pure PaaS, only the user-level middleware is offered, and it has to be complemented with a virtual or physical infrastructure.
The top layer of the reference model depicted in Figure contains services delivered at the application level. These are mostly referred to as Software-as-a-Service (SaaS). In most cases these are Web-based applications that rely on the cloud to provide service to end users. The horsepower of the cloud provided by IaaS and PaaS solutions allows independent software vendors to deliver their application services over the Internet. The reference model described in Figure also introduces the concept of everything as a Service (XaaS).

LAYERS / SERVICES OF CLOUDS

Cloud computing services are divided into three classes, according to the abstraction level of the capability provided and the service model of providers, namely: (1) Infrastructure as a Service, (2) Platform as a Service, and (3) Software as a Service. Figure 2 depicts the layered organization of the cloud stack from physical infrastructure to applications.

These abstraction levels can also be viewed as a layered architecture where services of a higher layer can be composed from services of the underlying layer. The reference model explains the role of each layer in an integrated architecture. Cloud development environments are built on top of infrastructure services to offer application development and deployment capabilities; in this level, various programming models, libraries, APIs, and mashup editors enable the creation of a range of business, Web, and scientific applications. Once deployed in the cloud, these applications can be consumed by end users.

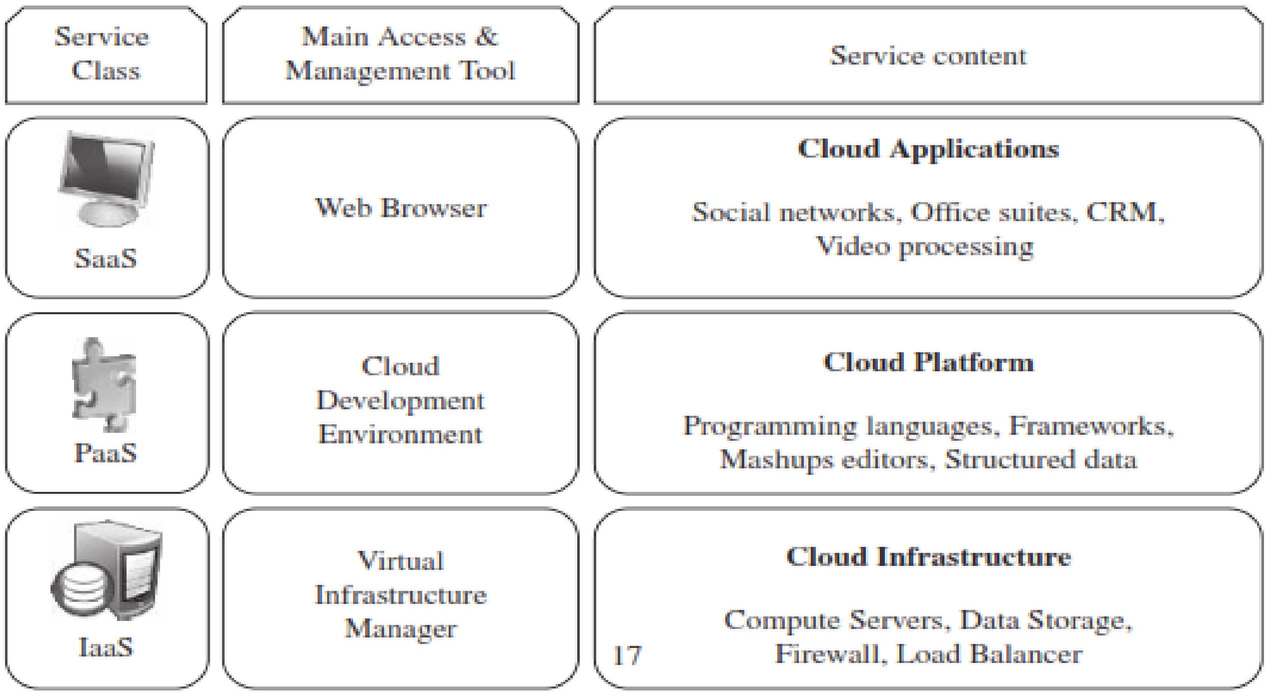

Service Class
Main Access & Management Tool
Service content
SaaS
Web Browser
Cloud Applications
Social networks, Office suites, CRM, Video processing
PaaS
Cloud Development Environment
Cloud Platform
Programming languages, Frameworks, Mashups editors, Structured data
IaaS
Virtual Infrastructure Manager
Cloud Infrastructure
17
Compute Servers, Data Storage, Firewall, Load Balancer

Fig. 2 Cloud computing stack

Infrastructure as a Service

Offering virtualized resources (computation, storage, and communication) on demand is known as Infrastructure as a Service (IaaS). This model allows users to use virtualized IT resources for computing, storage, and networking. In short, the service is performed by rented cloud infrastructure. The user can deploy and run his applications over his chosen OS environment. The user does not manage or control the underlying cloud infrastructure, but has control over the OS, storage, deployed applications, and possibly select networking components.

This IaaS model encompasses *storage as a service*, compute *instances as a service*, and c*ommunication as a service*.

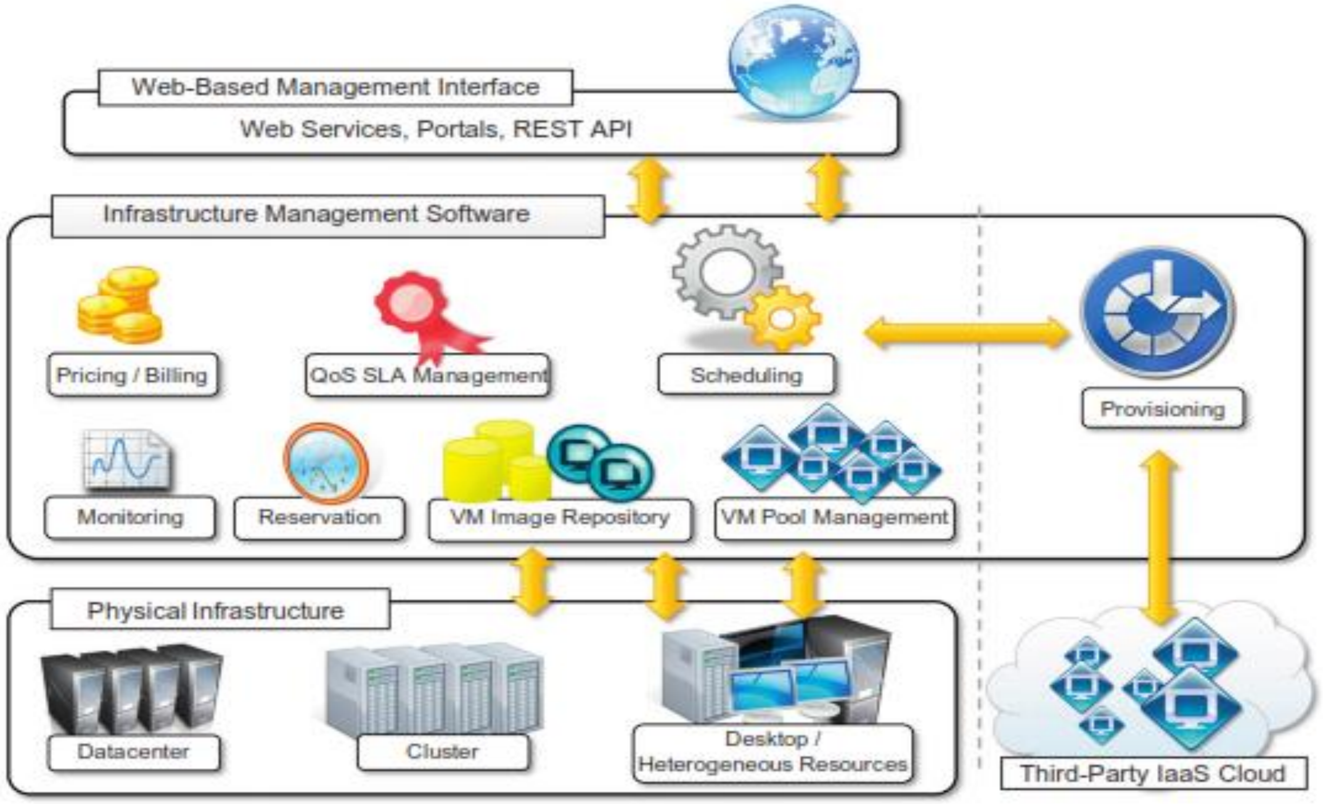


Fig. 3 Infrastructure as service

Figure 3 provides an overall view of the components forming an Infrastructure-as-a-Service solution. It is possible to distinguish three principal layers: the physical infrastructure, the software management infrastructure, and the user interface. At the top layer the user interface provides access to the services exposed by the software management infrastructure. Such an interface is generally based on Web 2.0 technologies: Web services, RESTful APIs, and mash-ups. These technologies allow either applications or final users to access the services exposed by the underlying infrastructure.
The core features of an IaaS solution are implemented in the infrastructure management software layer. In particular, management of the virtual machines is the most important function performed by this layer. A central role is played by the scheduler, which is in charge of allocating the execution of virtual machine instances. The scheduler interacts with the other components that perform a variety of tasks:

- The pricing and billing component takes care of the cost of executing each virtual machine instance and maintains data that will be used to charge the user.
- The monitoring component tracks the execution of each virtual machine instance and maintains data required for reporting and analyzing the performance of the system.
- The reservation component stores the information of all the virtual machine instances that have been executed or that will be executed in the future.
- If support for QoS-based execution is provided, a QoS/SLA management component will maintain a repository of all the SLAs made with the users; together with the monitoring component, this component is used to ensure that a given virtual machine instance is executed with the desired quality of service.
- The VM repository component provides a catalog of virtual machine images that users can use to create virtual instances. Some implementations also allow users to upload their specific virtual machine images.
- AVM pool manager component is responsible for keeping track of all the live instances.
- Finally, if the system supports the integration of additional resources belonging to a third-party IaaS provider, a provisioning component interacts with the scheduler to provide a virtual machine instance that is external to the local physical infrastructure directly managed by the pool.

Platform as a Service

A cloud easily programmable, known as Platform as a Service (PaaS). A cloud platform offers an environment on which developers create and deploy applications and do not necessarily need to know how many processors or how much memory that applications will be using. Google AppEngine, an example of Platform as a Service, offers a scalable environment for developing and hosting Web applications, which should be written in specific programming languages such as Python or Java, and use the services' own proprietary structured object data store.
Platform-as-a-Service (PaaS) solutions provide a development and deployment platform for running applications in the cloud. They constitute the middleware on top of which applications are built. A general features characterizing the PaaS approach is given in Figure 4.

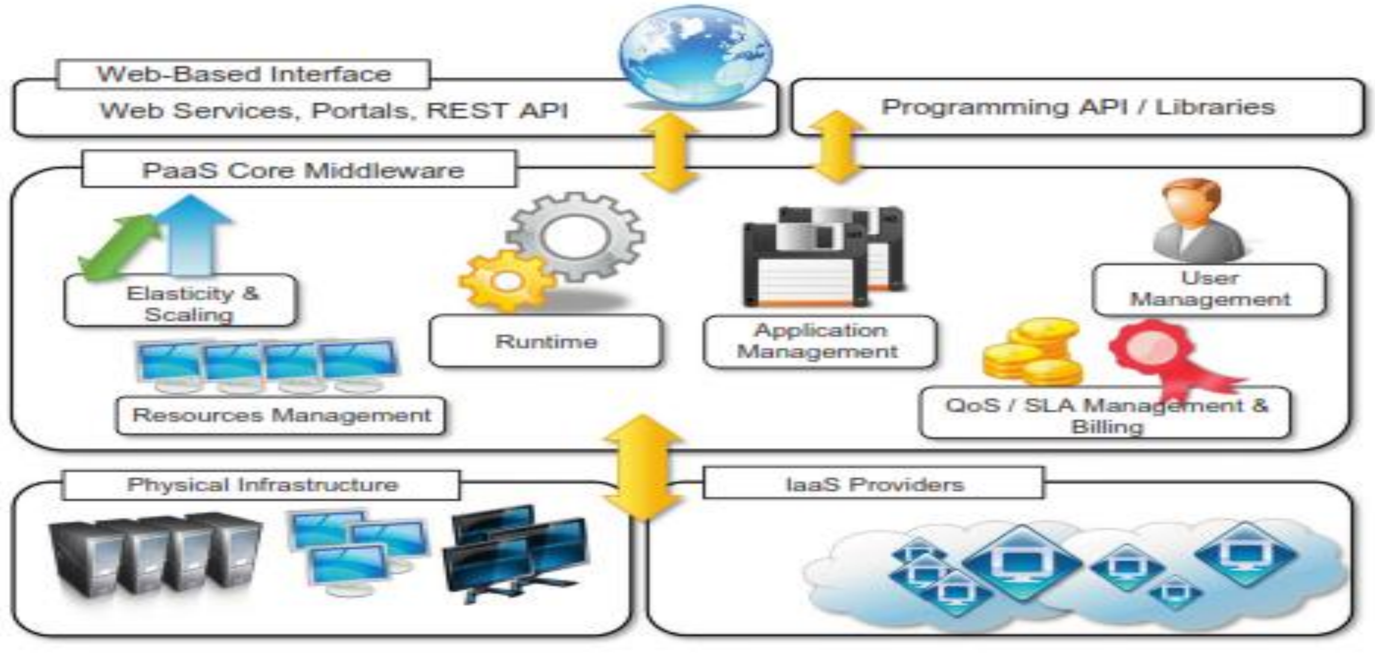


Fig. 4 Platform as a service

Application management is the core functionality of the middleware. PaaS implementations provide applications with a runtime environment and do not expose any service for managing the underlying infrastructure. They automate the process of deploying applications to the infrastructure, configuring application components, provisioning and configuring supporting technologies such as load balancers and databases, and managing system change based on policies set by the user.

The core middleware is in charge of managing the resources and scaling applications on demand or automatically, according to the commitments made with users. From a user point of view, the core middleware exposes interfaces that allow programming and deploying applications on the cloud. These can be in the form of a Web-based interface or in the form of programming APIs and libraries.

There are some essential characteristics that identify a PaaS solution:

- **Runtime framework.** This framework represents the "software stack" of the PaaS model and the most intuitive aspect that comes to people's minds when they refer to PaaS solutions. The runtime framework executes end-user code according to the policies set by the user and the provider.
- **Abstraction.** PaaS solutions are distinguished by the higher level of abstraction that they provide. Whereas in the case of IaaS solutions the focus is on delivering "raw" access to virtual or physical infrastructure, in the case of PaaS the focus is on the applications the cloud must support. This means that PaaS solutions offer a way to deploy and manage applications on the cloud rather than a bunch of virtual machines on top of which the IT infrastructure is built and configured.
- **Automation.** PaaS environments automate the process of deploying applications to the infrastructure, scaling them by provisioning additional resources when needed. This process is performed automatically and according to the SLA made between the customers and the provider. This feature is normally not native in IaaS solutions, which only provide ways to provision more resources.
- **Cloud services.** PaaS offerings provide developers and architects with services and APIs, helping them to simplify the creation and delivery of elastic and highly available cloud applications. These services are the key differentiators among competing PaaS solutions and generally include specific components for developing applications, advanced services for application monitoring, management, and reporting.

Another essential component for a PaaS-based approach is the ability to integrate third-party cloud services offered from other vendors by leveraging service-oriented architecture. Such integration should happen through standard interfaces and protocols. This opportunity makes the development of applications more agile and able to evolve according to the needs of customers and users.

Software as a Service

Applications reside on the top of the cloud stack. Services provided by this layer can be accessed by end users through Web portals. Therefore, consumers are increasingly shifting from locally installed computer programs to on-line software services that offer the same functionally. Traditional desktop applications such as word processing and spreadsheet can now be accessed as a service in the Web. This model of delivering applications, known as Software as a Service (SaaS), alleviates the burden of software maintenance for customers and simplifies development and testing for providers.

It provides a means to free users from complex hardware and software management by offloading such tasks to third parties, which build applications accessible to multiple users through a Web browser. In this scenario, customers neither need install anything on their premises nor have to pay considerable up-front costs to purchase the software and the required licenses. They simply access the application website, enter their credentials and billing details, and can instantly use the application, which, in most of the cases, can be further customized for their needs. On the provider side, the specific details and features of each customer's application are maintained in the infrastructure and made available on demand.

The SaaS model is appealing for applications serving a wide range of users and that can be adapted to specific needs with little further customization. This requirement characterizes SaaS as a "one-to-many" software delivery model, whereby an application is shared across multiple users. It constitutes the perfect candidate for hosted solutions, since the applications delivered to the user are the same, and the applications themselves provide users with the means to shape the applications according to user needs. As a result, SaaS applications are naturally multitenant.

Multitenancy, which is a feature of SaaS compared to traditional packaged software, allows providers to centralize and sustain the effort of managing large hardware infrastructures, maintaining and upgrading applications transparently to the users, and optimizing resources by sharing the costs among the large user base. On the customer side, such costs constitute a minimal fraction of the usage fee paid for the software.
The core characteristics of SaaS:

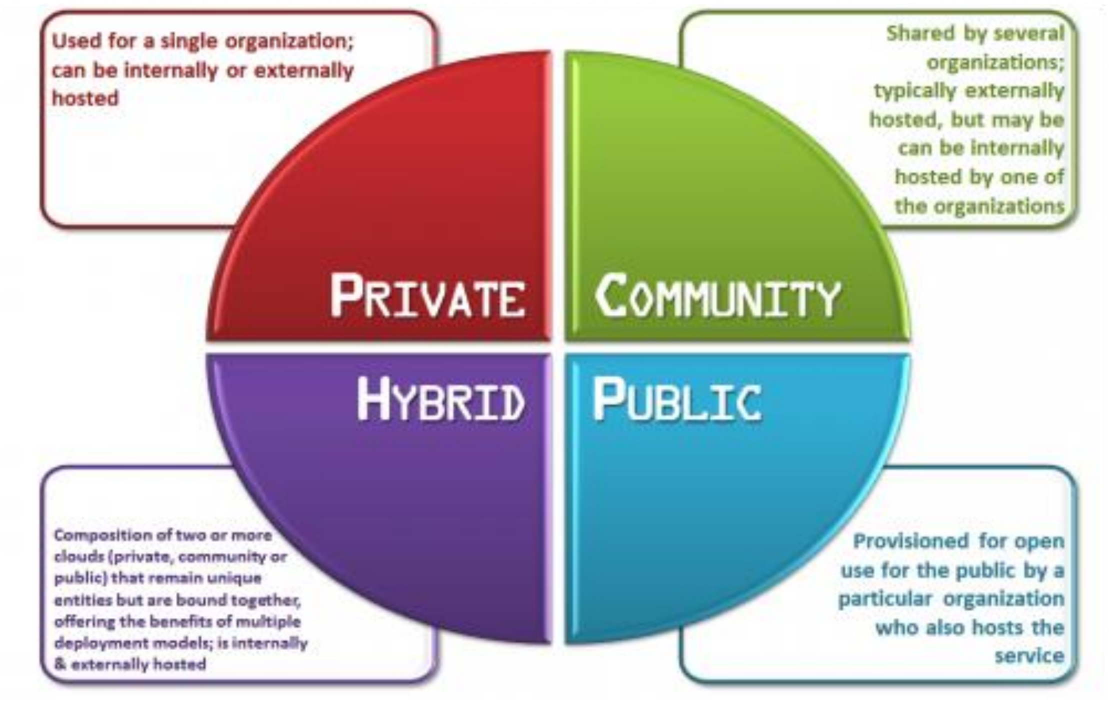
- The product sold to customer is application access.
- The application is centrally managed.
- The service delivered is one-to-many.
- The service delivered is an integrated solution delivered on the contract, which means provided as promised.

Software-as-a-Service applications can serve different needs. CRM, ERP, and social networking applications are definitely the most popular ones. Office automation applications are also an important representative for SaaS applications: Google Documents are example of Web-based applications that aim to address all user needs for documents, spreadsheets, and presentation management. They offer a Web-based interface for creating, managing, and modifying documents that can be easily shared among users and made accessible from anywhere.

Types of clouds

In cloud computing technology, large pools of resources can be connected through private or public networks. A more useful classification is given according to the administrative domain of a cloud: It identifies the boundaries within which cloud computing services are implemented, provides hints on the underlying infrastructure adopted to support such services, and qualifies them.

It is then possible to differentiate four different types of cloud:


Used for a single organization; can be internally or externally hosted
Shared by several organizations; typically externally hosted, but may be can be internally hosted by one of the organizations
PRIVATE
COMMUNITY
HYBRID
PUBLIC
Composition of two or more clouds (private, community or public) that remain unique entities but are bound together, offering the benefits of multiple deployment models; is internally & externally hosted
Provisioned for open use for the public by a particular organization who also hosts the service

- **Public clouds**. The cloud is open to the wider public.
- **Private clouds.** The cloud is implemented within the private premises of an institution and generally made accessible to the members of the institution or a subset of them.
- **Hybrid or heterogeneous clouds**. The cloud is a combination of the two previous solutions and most likely identifies a private cloud that has been augmented with resources or services hosted in a public cloud.

- **Community clouds**. The cloud is characterized by a multi-administrative domain involving different deployment models (public, private, and hybrid), and it is specifically designed to address the needs of a specific industry.

Almost all the implementations of clouds can be classified in this categorization. In the following sections provide brief characterizations of these clouds.

1. Public clouds

Public clouds constitute the first expression of cloud computing. They are a realization of the canonical view of cloud computing in which the services offered are made available to anyone, from anywhere, and at any time through the Internet. From a structural point of view they are a distributed system, most likely composed of one or more datacenters connected together, on top of which the specific services offered by the cloud are implemented. Any customer can easily sign in with the cloud provider, enter her credential and billing details, and use the services offered.

Historically, public clouds were the first class of cloud that were implemented and offered. They offer solutions for minimizing IT infrastructure costs and serve as a viable option for handling peak loads on the local infrastructure. They have become an interesting option for small enterprises, which are able to start their businesses without large up-front investments by completely relying on public infrastructure for their IT needs. What made attractive public clouds compared to the reshaping of the private premises and the purchase of hardware and software was the ability to grow or shrink according to the needs of the related business. By renting the infrastructure or subscribing to application services, customers were able to dynamically upsize or downsize their IT according to the demands of their business.

A fundamental characteristic of public clouds is multitenancy. A public cloud is meant to serve a multitude of users, not a single customer. Any customer requires a virtual computing environment that is separated, and most likely isolated, from other users. This is a fundamental requirement to provide effective monitoring of user activities and guarantee the desired performance and the other QoS attributes negotiated with users. QoS management is a very important aspect of public clouds. A public cloud can offer any kind of service: infrastructure, platform, or applications. For example, Amazon EC2 is a public cloud that provides infrastructure as a service; Google AppEngine is a public cloud that provides an application development platform as a service.

Public clouds can be composed of geographically dispersed datacenters to share the load of users and better serve them according to their locations. For example, Amazon Web Services has datacenters installed in the United States, Europe, Singapore, and Australia; they allow their customers to choose between three different regions: us-west-1, us-east-1,or eu-west-1. Such regions are priced differently and are further divided into availability zones, which map to specific datacenters.

BENEFITS OF PUBLIC CLOUD

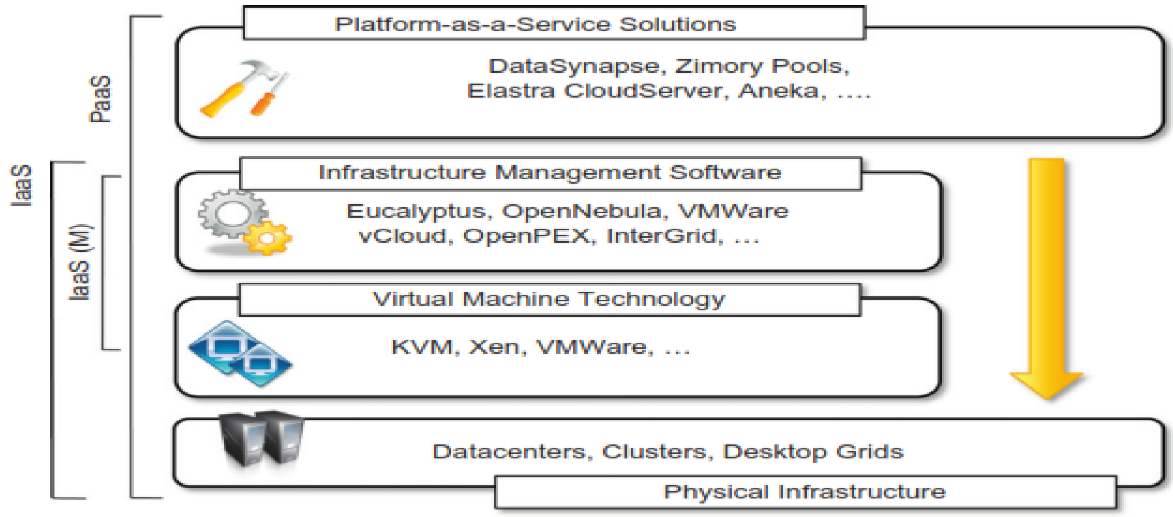
Cost Effective: Since Public Cloud Share Same Resources With Large Number Of Consumer, It Has Low Cost.
Reliability: Since Public Cloud Employs Large Number of Resources from Different Locations, If Any of the Resource Fail, Public Cloud Can Employ Another One.
Flexibility: It Is Also Very Easy To Integrate Public Cloud With Private Cloud, Hence Gives Consumers A Flexible Approach.
Location Independence: Since, Public Cloud Services Are Delivered Through Internet, therefore Ensures Location Independence.
Utility Style Costing: Public Cloud is also based on Pay-Per-Use Model and resources are Accessible whenever Consumer Needs It.
High Scalability: Cloud Resources are Made Available on Demand from A Pool of Resources, I.E., They Can Be Scaled up or down According the Requirement.

DISADVANTAGES OF PUBLIC CLOUD

Low Security: In public cloud model**,** data is hosted off-site and resources are shared publicly, therefore does not ensure higher level of security.
Less Customizable: It is comparatively less customizable than private cloud.

2. Private clouds

Private clouds are virtual distributed systems that rely on a private infrastructure and provide internal users with dynamic provisioning of computing resources. Instead of a pay-as-you-go model as in public clouds, there could be other schemes in place, taking into account the usage of the cloud and proportionally billing the different departments or sections of an enterprise. Private clouds have the advantage of keeping the core business operations in-house by relying on the existing IT infrastructure and reducing the burden of maintaining it once the cloud has been set up.
In this scenario, security concerns are less critical, since sensitive information does not flow out of the private infrastructure. Moreover, existing IT resources can be better utilized because the private cloud can provide services to a different range of users. Another interesting opportunity that comes with private clouds is the possibility of testing applications and systems at a comparatively lower price rather than public clouds before deploying them on the public virtual infrastructure. The key advantages of using a private cloud computing infrastructure:


Platform-as-a-Service Solutions
DataSynapse, Zimory Pools, Elastra CloudServer, Aneka, ...
Infrastructure Management Software
Eucalyptus, OpenNebula, VMWare vCloud, OpenPEX, InterGrid, ...
Virtual Machine Technology
KVM, Xen, VMWare, ...
Datacenters, Clusters, Desktop Grids
Physical Infrastructure
PaaS
IaaS
IaaS (M)

Private cloud hardware and software stack

- **Customer information protection.** Despite assurances by the public cloud leaders about security, few provide satisfactory disclosure or have long enough histories with their cloud offerings to provide warranties about the specific level of security put in place on their systems. In-house security is easier to maintain and rely on.
- **Infrastructure ensuring SLAs**. Quality of service implies specific operations such as appropriate clustering and failover, data replication, system monitoring and maintenance, and disaster recovery, and other uptime services can be commensurate to the application needs. Although public cloud vendors provide some of these features, not all of them are available as needed.
- **Compliance with standard procedures and operations.** If organizations are subject to third-party compliance standards, specific procedures have to be put in place when deploying and executing applications. This could be not possible in the case of the virtual public infrastructure.

From an architectural point of view, private clouds can be implemented on more heterogeneous hardware: They generally rely on the existing IT infrastructure already deployed on the private premises. This could be a datacenter, a cluster, an enterprise desktop grid, or a combination of them.

BENEFITS OF PRIVATE CLOUD

Higher Security and Privacy: Private cloud operations are not available to general public and resources are shared from distinct pool of resources, therefore, ensures high security and privacy.
More Control: Private clouds have more control on its resources and hardware than public cloud because it is accessed only within an organization.
Cost and Energy Efficiency: Private cloud resources are not as cost effective as public clouds but they offer more efficiency than public cloud.

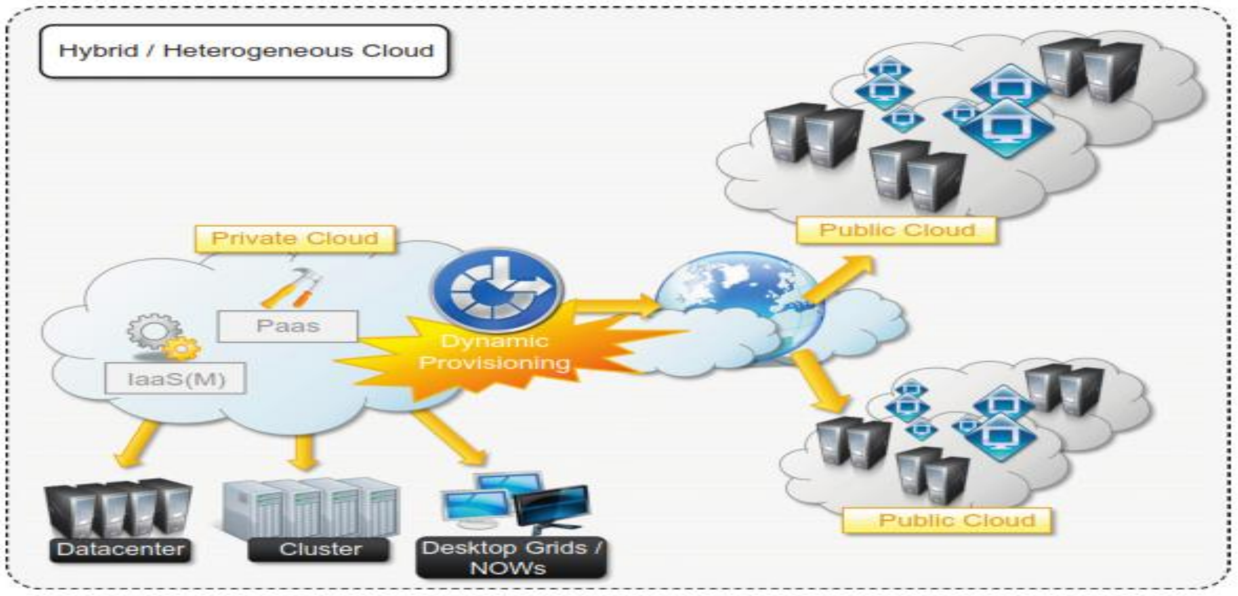
DISADVANTAGES OF PRIVATE CLOUD

Restricted Area: Private cloud is only accessible locally and is very difficult to deploy globally.
Inflexible Pricing: In order to fulfill demand, purchasing new hardware is very costly.
Limited Scalability: Private cloud can be scaled only within capacity of internal hosted resources.
Additional Skills: In order to maintain cloud deployment, organization requires more skilled and expertise.

3. Hybrid clouds

Public clouds are large software and hardware infrastructures that have a capability that is huge enough to serve the needs of multiple users, but they suffer from security threats and administrative pitfalls. Although the option of completely relying on a public virtual infrastructure is appealing for companies that did not incur IT capital costs and have just started considering their IT needs (i.e., start-ups), in most cases the private cloud option prevails because of the existing IT infrastructure.

Private clouds are the perfect solution when it is necessary to keep the processing of information within an enterprise's premises or it is necessary to use the existing hardware and software infrastructure. One of the major drawbacks of private deployments is the inability to scale on demand and to efficiently address peak loads. In this case, it is important to leverage capabilities of public clouds as needed. Hence, a hybrid solution could be an interesting opportunity for taking advantage of the best of the private and public worlds. This led to the development and diffusion of hybrid clouds.



Hybrid/ Heterogeneous cloud

Hybrid clouds allow enterprises to exploit existing IT infrastructures, maintain sensitive information within the premises, and naturally grow and shrink by provisioning external resources and releasing them when they're no longer needed. Security concerns are then only limited to the public portion of the cloud that can be used to perform operations with less stringent constraints but that are still part of the system workload.
Figure provides a general overview of a hybrid cloud: It is a heterogeneous distributed system resulting from a private cloud that integrates additional services or resources from one or more public clouds. For this reason they are also called heterogeneous clouds. As depicted in the diagram, dynamic provisioning is a fundamental component in this scenario. Hybrid clouds address scalability issues by leveraging external resources for exceeding capacity demand. These resources or services are temporarily leased for the time required and then released. This practice is also known as *cloud bursting*.

BENEFITS OF HYBRID CLOUD

Scalability: It offers both features of public cloud scalability and private cloud scalability.
Flexibility: It offers both secure resources and scalable public resources.
Cost Efficiencies: Public cloud are more cost effective than private, therefore hybrid cloud can have this saving.
Security: Private cloud in hybrid cloud ensures higher degree of security.

DISADVANTAGES OF HYBRID CLOUD

Networking Issues: Networking becomes complex due to presence of private and public cloud.

Security Compliance: It is necessary to ensure that cloud services are compliant with organization's security policies.

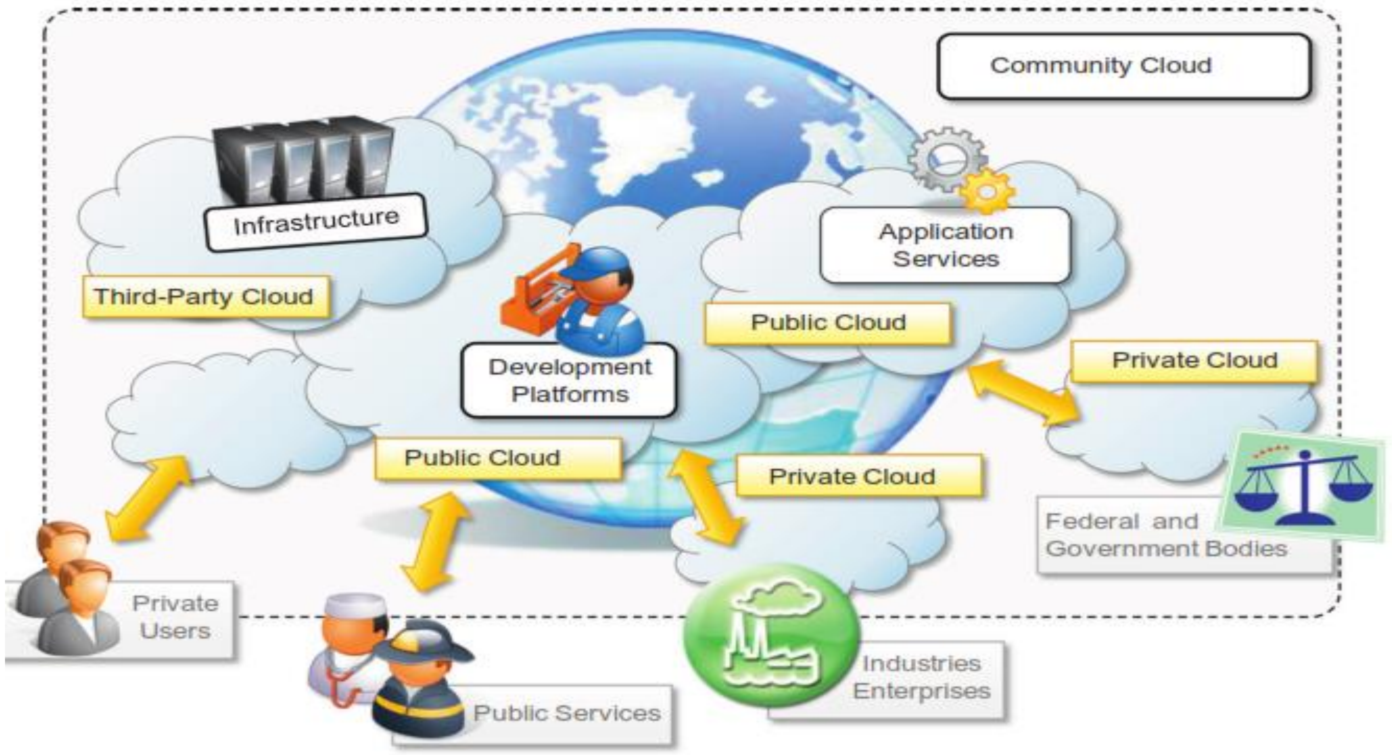
Infrastructural Dependency: The hybrid cloud model is dependent on internal IT infrastructure, therefore it is necessary to ensure redundancy across data centers.

4. Community clouds

Community cloud is the subset of public cloud, Community clouds are distributed systems created by integrating the services of different clouds to address the specific needs of an industry, a community, or a business sector. The infrastructure is shared by several organizations and supports a specific community that has shared concerns (e.g., mission, security requirements, policy, and compliance considerations). It may be managed by the organizations or a third party and may exist on premise or off premise.

Figure provides a general view of the usage scenario of community clouds, together with reference architecture. The users of a specific community cloud fall into a well-identified community, sharing the same concerns or needs; they can be government bodies, industries, or even simple users, but all of them focus on the same issues for their interaction with the cloud. This is a different scenario than public clouds, which serve a multitude of users with different needs. Community clouds are also different from private clouds, where the services are generally delivered within the institution that owns the cloud.

From an architectural point of view, a community cloud is most likely implemented over multiple administrative domains. This means that different organizations such as government bodies, private enterprises, research organizations, and even public virtual infrastructure providers contribute with their resources to build the cloud infrastructure.



Community Cloud

Candidate sectors for community clouds are as follows:

- **Media industry**. In the media industry, companies are looking for low-cost, agile, and simple solutions to improve the efficiency of content production. Most media productions involve an extended ecosystem of partners. In particular, the creation of digital content is the outcome of a collaborative process that includes movement of large data, massive compute-intensive rendering tasks, and complex workflow executions. Community clouds can provide a shared environment where services can facilitate business-to-business collaboration and offer the horsepower in terms of aggregate bandwidth, CPU, and storage required to efficiently support media production.

- **Healthcare industry**. In the healthcare industry, there are different scenarios in which community clouds could be of use. In particular, community clouds can provide a global platform on which to share information and knowledge without revealing sensitive data maintained within the private infrastructure. The naturally hybrid deployment model of community clouds can easily support the storing of patient-related data in a private cloud while using the shared infrastructure for noncritical services and automating processes within hospitals.
- **Energy and other core industries**. In these sectors, community clouds can bundle the comprehensive set of solutions that together vertically address management, deployment, and orchestration of services and operations. Since these industries involve different providers, vendors, and organizations, a community cloud can provide the right type of infrastructure to create an open and fair market.
- **Public sector**. Legal and political restrictions in the public sector can limit the adoption of public cloud offerings. Moreover, governmental processes involve several institutions and agencies and are aimed at providing strategic solutions at local, national, and international administrative levels. They involve business-to-administration, citizen-to-administration, and possibly business-to-business processes. Some examples include invoice approval, infrastructure planning, and public hearings. A community cloud can constitute the optimal venue to provide a distributed environment in which to create a communication platform for performing such operations.
- **Scientific research**. Science clouds are an interesting example of community clouds. In this case, the common interest driving different organizations sharing a large distributed infrastructure is scientific computing.

BENEFITS OF COMMUNITY CLOUD
Cost Effective: Community cloud offers same advantage as that of private cloud at low cost. Sharing between Organizations Community cloud provides an infrastructure to share cloud resources and capabilities among several organizations.
Security: Community cloud is comparatively more secure than the public cloud.

ISSUES IN COMMUNITY CLOUD

- Since all data is housed at one location, one must be careful in storing data in community cloud because it might be accessible by others.
- It is also challenging to allocate responsibilities of governance, security and cost.

DATA-CENTER DESIGN AND INTERCONNECTION NETWORKS

A data center is often built with a large number of servers through a huge interconnection network.

Warehouse-Scale Data-Center Design
"The cloud is built on massive datacenters". Figure shows a data center that is as large as a shopping mall (11 times the size of a football field) under one roof. Such a data center can house 400,000 to 1 million servers. The data centers are built economics of scale—meaning lower unit cost for larger data centers.
A small data center could have 1,000 servers. The larger the data center, the lower the operational cost. The approximate monthly cost to operate a huge 400-server data center is estimated by network cost \$13/Mbps; storage cost \$0.4/GB; and administration costs. These unit costs are greater than those of a 1,000-server data center. The network cost to operate a small data center is about seven times greater and the storage cost is 5.7 times greater. Microsoft has about 100 data centers, large or small, which are distributed around the globe.

Large Scale Data center

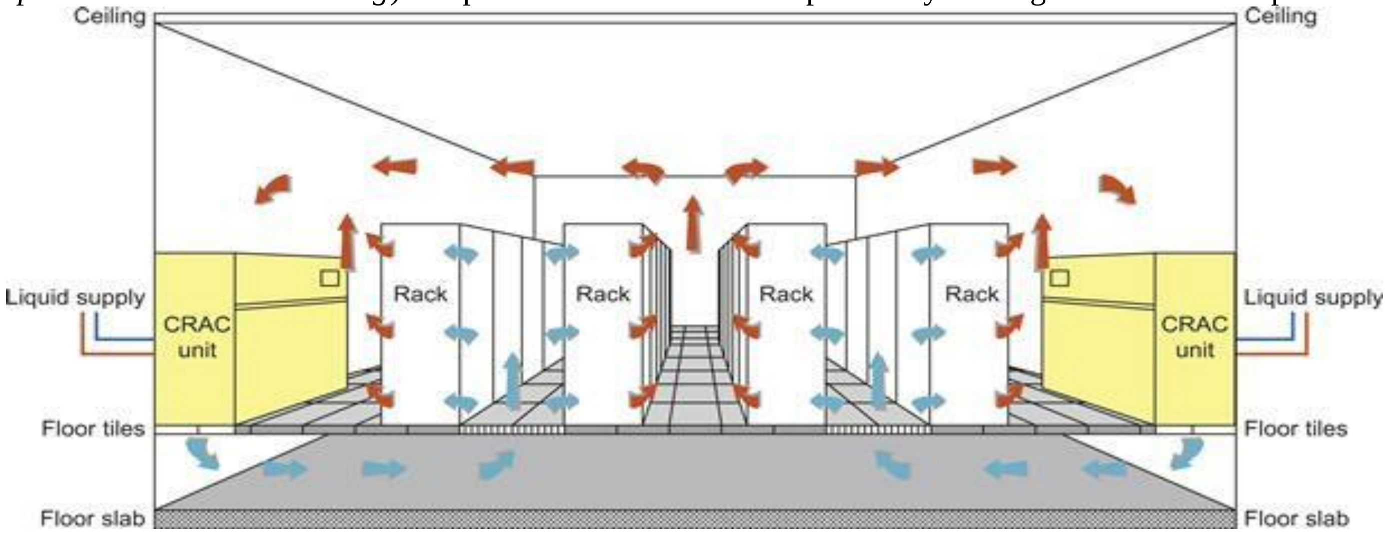
Data-Center Construction Requirements

Most data centers are built with commercially available components. An off-the-shelf server consists of a number of processor sockets, each with a multicore CPU and its internal cache hierarchy, local shared and coherent DRAM, and a number of directly attached disk drives. The DRAM and disk resources within the rack are accessible through first-level rack switches and all resources in all racks are accessible via a cluster level switch. Consider a data center built with 2,000 servers, each with 8 GB of DRAM and four 1 TB disk drives. Each group of 40 servers is connected through a 1 Gbps link to a rack-level switch that has an additional eight 1 Gbps ports used for connecting the rack to the cluster-level switch.

It was estimated that the bandwidth available from local disks is 200 MB/s, whereas the bandwidth from off-rack disks is 25 MB/s via shared rack uplinks. The total disk storage in the cluster is almost 10 million times larger than local DRAM. A large application must deal with large discrepancies in latency, bandwidth, and capacity. In a very large-scale data center, components are relatively cheaper. The components used in data centers are very different from those in building supercomputer systems. With a scale of thousands of servers, concurrent failure, either hardware failure or software failure, of 1 percent of nodes is common. Many failures can happen in hardware; for example, CPU failure, disk I/O failure, and network failure. It is even quite possible that the whole data center does not work in the case of a power crash. Also, some failures are brought on by software. The service and data should not be lost in a failure situation. Reliability can be achieved by redundant hardware. The software must keep multiple copies of data in different locations and keep the data accessible while facing hardware or software errors.

Cooling System of a Data-Center Room

Shows the layout and cooling facility of a warehouse in a data center. The datacenter room has raised floors for hiding cables, power lines, and cooling supplies. The cooling system is somewhat simpler than the power system. The raised floor has a steel grid resting on stanchions about 2–4 ft above the concrete floor. The under-floor area is often used to route power cables to racks, but its primary use is to distribute cool air to the server rack. The CRAC (*computer room air conditioning*) unit pressurizes the raised floor plenum by blowing cold air into the plenum.



The cooling system in a raised-floor data center with hot-cold air circulation supporting water heat exchange facilities

The cold air escapes from the plenum through perforated tiles that are placed in front of server racks. Racks are arranged in long aisles that alternate between cold aisles and hot aisles to avoid mixing hot and cold air. The hot air produced by the servers circulates back to the intakes of the CRAC units that cool it and then exhaust the cool air into the raised floor plenum again. Typically, the incoming coolant is at 12–14°C and the warm coolant returns to a chiller. Newer data centers often insert a cooling tower to pre-cool the condenser water loop fluid. Water-based free cooling uses cooling towers to dissipate heat. The cooling towers use a separate cooling loop in which water absorbs the coolant's heat in a heat exchanger.

DATA-CENTER INTERCONNECTION NETWORKS

A critical core design of a data center is the interconnection network among all servers in the data-center cluster. This network design must meet five special requirements: low latency, high bandwidth, low cost, *message-passing interface* (MPI) communication support, and fault tolerance. The design of an inter-server network must satisfy both point-to-point and collective communication patterns among all server nodes. Specific design considerations are given in the following sections.

Application Traffic Support

The network topology should support all MPI communication patterns. Both point-to point and collective MPI communications must be supported. The network should have high bisection bandwidth to meet this requirement. For example, one-to-many communications are used for supporting distributed file access. One can use one or a few servers as metadata master servers which need to communicate with slave server nodes in the cluster. To support the Map Reduce programming paradigm, the network must be designed to perform the map and reduce functions at a high speed. In other words, the underlying network structure should support various network traffic patterns demanded by user applications.

Network Expandability

The interconnection network should be expandable. With thousands or even hundreds of thousands of server nodes, the cluster network interconnection should be allowed to expand once more servers are added to the data center. The network topology should be restructured while facing such expected growth in the future. Also, the network should be designed to support load balancing and data movement among the servers. None of the links should become a bottleneck that slows down application performance.
The topology of the interconnection should avoid such bottlenecks. The most critical issue regarding expandability is support of modular network growth for building data-center containers. One single data-center container contains hundreds of servers and is considered to be the building block of large-scale data centers. Cable connections are then needed among multiple data-center containers. Data centers are not built by piling up servers in multiple racks today. Instead, datacenter owners buy server containers while each container contains several hundred or even thousands of server nodes. The owners can just plug in the power supply, outside connection link, and cooling water, and the whole system will just work. This is quite efficient and reduces the cost of purchasing and maintaining servers.

Fault Tolerance and Graceful Degradation

The interconnection network should provide some mechanism to tolerate link or switch failures. In addition, multiple paths should be established between any two server nodes in a data center. Fault tolerance of servers is achieved by replicating data and computing among redundant servers. Similar redundancy technology should apply to the network structure. Both software and hardware network redundancy apply to cope with potential failures. One the software side, the software layer should be aware of network failures.
Packet forwarding should avoid using broken links. The network support software drivers should handle this transparently without affecting cloud operations. In case of failures, the network structure should degrade gracefully amid limited node failures. Hot-swappable components are desired. There should be no critical paths or critical points which may become a single point of failure that pulls down the entire system. The network structure is often divided into two layers. The lower layer is close to the end servers, and the upper layer establishes the backbone connections among the server groups or sub-clusters. This hierarchical interconnection approach appeals to building data centers with modular containers.

Modular Data Center in Shipping Containers
A modern data center is structured as a shipyard of server clusters housed in truck-towed containers. Figure shows the housing of multiple sever racks in a truck-towed container in the SGI ICE Cube modular data center. Inside the container, hundreds of blade servers are housed in racks surrounding the container walls. An array of fans forces the heated air generated by the server racks to go through a heat exchanger, which cools the air for the next rack (detail in callout) on a continuous loop. The SGI ICE Cube container can house 46,080 processing cores or 30 PB of storage per container.

sgi
sgi
ICE Cube®
The world's most efficient modular data center
Built with the latest Intel® Xeon® processors

This container-based data center was motivated by demand for lower power consumption, higher computer density, and mobility to relocate data centers to better locations with lower electricity costs, better cooling water supplies, and cheaper housing for maintenance engineers. Sophisticated cooling technology enables up to 80% reduction in cooling costs compared with traditional warehouse data centers. Both chilled air circulation and cold water are flowing through the heat exchange pipes to keep the server racks cool and easy to repair.
Data centers usually are built at a site where leases and utilities for electricity are cheaper, and cooling is more efficient.

Container Data-Center Construction
The data-center module is housed in a truck-towable container. The modular container design includes the network, computer, storage, and cooling gear. One needs to increase cooling efficiency by varying the water and airflow with better airflow management. Another concern is to meet seasonal load requirements. The construction of a container based data center may start with one system (server), then move to a rack system design, and finally to a container system. This staged development may take different amounts of time and demand increasing costs.
The container must be designed to be weatherproof and easy to transport. The modular data-center approach supports many cloud service applications. For example, the health care industry will benefit by installing a data center at all clinic sites.

Data-Center Management Issues
Here are basic requirements for managing the resources of a data center. These suggestions have resulted from the design and operational experiences of many data centers in the IT and service industries.

- **Making common users happy** The data center should be designed to provide quality service to the majority of users.
- **Controlled information flow** Information flow should be streamlined. Sustained services and high availability (HA) are the primary goals.
- **Multiuser manageability** The system must be managed to support all functions of a data center, including traffic flow, database updating, and server maintenance.
- **Scalability to prepare for database growth** The system should allow growth as workload increases. The storage, processing, I/O, power, and cooling subsystems should be scalable.
- **Reliability in virtualized infrastructure** Failover, fault tolerance, and VM live migration should be integrated to enable recovery of critical applications from failures or disasters.
- **Low cost to both users and providers** The cost to users and providers of the cloud system built over the data centers should be reduced, including all operational costs.
- **Security enforcement and data protection** Data privacy and security defense mechanisms must be deployed to protect the data center against network attacks and system interrupts and to maintain data integrity from user abuses or network attacks.
- **Green information technology** Saving power consumption and upgrading energy efficiency are in high demand when designing and operating current and future data centers.

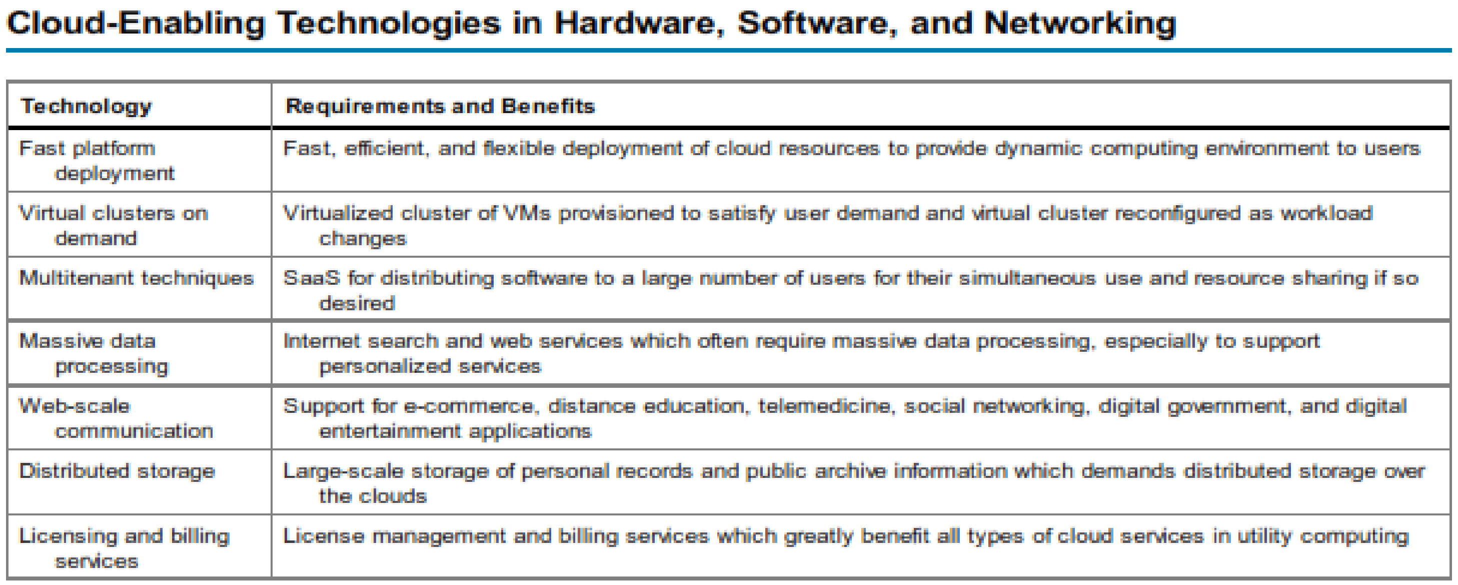
Architectural Design of Compute and Storage Clouds

Cloud Platform Design Goals

Scalability, virtualization, efficiency, and reliability are four major design goals of a cloud computing platform. Clouds support Web 2.0 applications. Cloud management receives the user request, finds the correct resources, and then calls the provisioning services which invoke the resources in the cloud. The cloud management software needs to support both physical and virtual machines. Security in shared resources and shared access of data centers also pose another design challenge.
The platform needs to establish a very large-scale HPC infrastructure. The hardware and software systems are combined to make it easy and efficient to operate. System scalability can benefit from cluster architecture. If one service takes a lot of processing power, storage capacity, or network traffic, it is simple to add more servers and bandwidth. System reliability can benefit from this architecture. Data can be put into multiple locations. For example, user e-mail can be put in three disks which expand to different geographically separate data centers. In such a situation, even if one of the data centers crashes, the user data is still accessible. The scale of the cloud architecture can be easily expanded by adding more servers and enlarging the network connectivity accordingly.

Enabling Technologies for Clouds

Cloud computing are the ubiquity of broadband and wireless networking, falling storage costs, and progressive improvements in Internet computing software. Cloud users are able to demand more capacity at peak demand, reduce costs, experiment with new services, and remove unneeded capacity, whereas service providers can increase system utilization via multiplexing, virtualization, and dynamic resource provisioning. Clouds are enabled by the progress in hardware, software, and networking technologies summarized in Table.

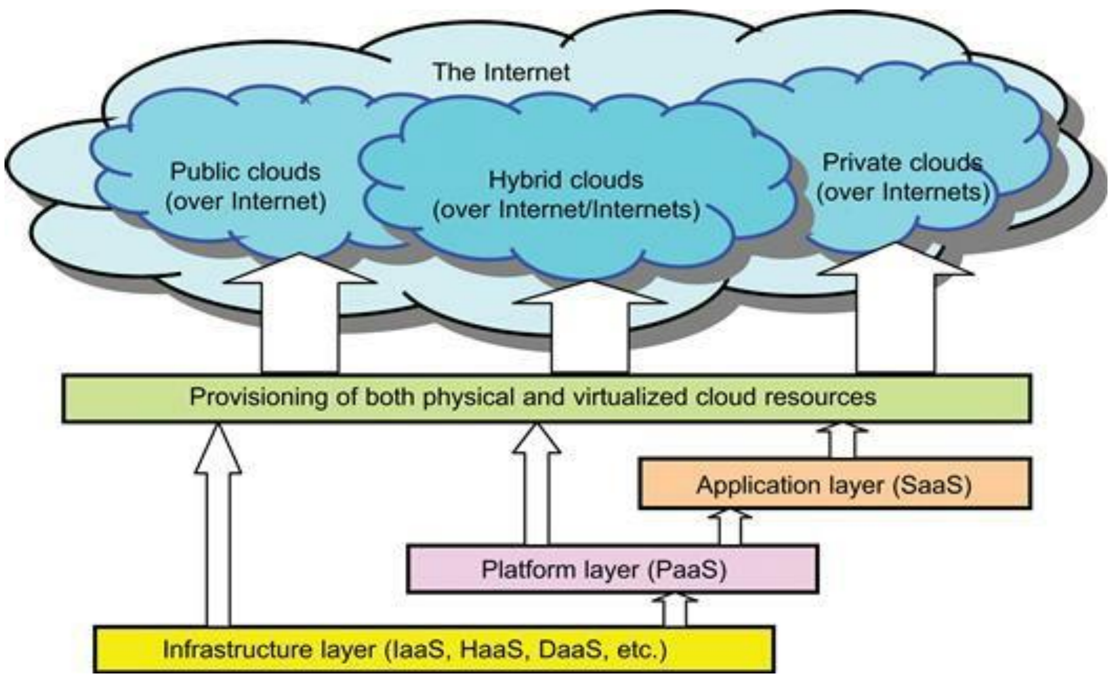

Cloud-Enabling Technologies in Hardware, Software, and Networking

Technology	Requirements and Benefits
Fast platform deployment	Fast, efficient, and flexible deployment of cloud resources to provide dynamic computing environment to users
Virtual clusters on demand	Virtualized cluster of VMs provisioned to satisfy user demand and virtual cluster reconfigured as workload changes
Multitenant techniques	SaaS for distributing software to a large number of users for their simultaneous use and resource sharing if so desired
Massive data processing	Internet search and web services which often require massive data processing, especially to support personalized services
Web-scale communication	Support for e-commerce, distance education, telemedicine, social networking, digital government, and digital entertainment applications
Distributed storage	Large-scale storage of personal records and public archive information which demands distributed storage over the clouds
Licensing and billing services	License management and billing services which greatly benefit all types of cloud services in utility computing

These technologies play instrumental roles in making cloud computing a reality. Most of these technologies are mature today to meet increasing demand. In the hardware area, the rapid progress in multicore CPUs, memory chips, and disk arrays has made it possible to build faster data centers with huge amounts of storage space. Resource virtualization enables rapid cloud deployment and disaster recovery.

Layered Cloud Architectural Development

The architecture of a cloud is developed at three layers: infrastructure, platform, and application, as demonstrated in Figure. These three development layers are implemented with virtualization and standardization of hardware and software resources provisioned in the cloud. The services to public, private, and hybrid clouds are conveyed to users through networking support over the Internet and intranets involved. It is clear that the infrastructure layer is deployed first to support IaaS services. This infrastructure layer serves as the foundation for building the platform layer of the cloud for supporting PaaS services. In turn, the platform layer is a foundation for implementing the application layer for SaaS applications.


The Internet
Public clouds (over Internet)
Hybrid clouds (over Internet/Internets)
Private clouds (over Internets)
Provisioning of both physical and virtualized cloud resources
Application layer (SaaS)
Platform layer (PaaS)
Infrastructure layer (IaaS, HaaS, DaaS, etc.)

Layered architectural development of the

Cloud platform for IaaS, PaaS, and SaaS

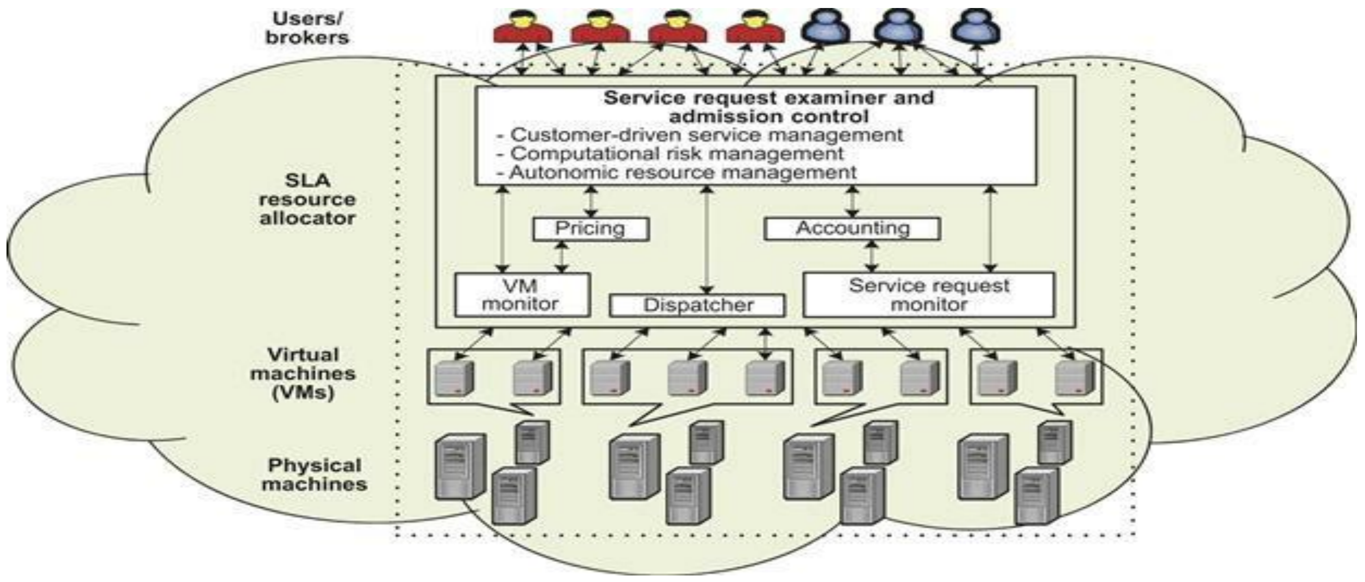
The infrastructure layer is built with virtualized compute, storage, and network resources. The abstraction of these hardware resources is meant to provide the flexibility demanded by users. Internally, virtualization realizes automated provisioning of resources and optimizes the infrastructure management process. The platform layer is for general-purpose and repeated usage of the collection of software resources. This layer provides users with an environment to develop their applications, to test operation flows, and to monitor execution results and performance. The platform should be able to assure users that they have scalability, dependability, and security protection. In a way, the virtualized cloud platform serves as a "system middleware" between the infrastructure and application layers of the cloud.

The application layer is formed with a collection of all needed software modules for SaaS applications. Service applications in this layer include daily office management work, such as information retrieval, document processing, and calendar and authentication services. The application layer is also heavily used by enterprises in business marketing and sales, consumer relationship management (CRM), financial transactions, and supply chain management. It should be noted that not all cloud services are restricted to a single layer. Many applications may apply resources at mixed layers. After all, the three layers are built from the bottom up with a dependence relationship.

Market-Oriented Cloud Architecture

As consumers rely on cloud providers to meet more of their computing needs, they will require a specific level of QoS to be maintained by their providers, in order to meet their objectives and sustain their operations. Cloud providers consider and meet the different QoS parameters of each individual consumer as negotiated in specific SLAs. To achieve this, the providers cannot deploy traditional system-centric resource management architecture. Instead, market-oriented resource management is necessary to regulate the supply and demand of cloud resources to achieve market equilibrium between supply and demand.

The designer needs to provide feedback on economic incentives for both consumers and providers. The purpose is to promote QoS-based resource allocation mechanisms. In addition, clients can benefit from the potential cost reduction of providers, which could lead to a more competitive market, and thus lower prices.


Users/ brokers
SLA resource allocator
Service request examiner and admission control
- Customer-driven service management
- Computational risk management
- Autonomic resource management
Pricing
Accounting
VM monitor
Dispatcher
Service request monitor
Virtual machines (VMs)
Physical machines

Market-oriented cloud architecture to expand/shrink leasing of resources with variation in QoS/demand from users

- Users or brokers acting on user's behalf submit service requests from anywhere in the world to the data center and cloud to be processed.
- The SLA resource allocator acts as the interface between the data center/cloud service provider and external users/brokers. It requires the interaction of the following mechanisms to support SLA-oriented resource management. When a service request is first submitted the service request examiner interprets the submitted request for QoS requirements before determining whether to accept or reject the request.
- The request examiner ensures that there is no overloading of resources whereby many service requests cannot be fulfilled successfully due to limited resources. It also needs the latest status information regarding resource availability (from the VM Monitor mechanism) and workload processing (from the Service Request Monitor mechanism) in order to make resource allocation decisions effectively.
- The Pricing mechanism decides how service requests are charged. For instance, requests can be charged based on submission time (peak/off-peak), pricing rates (fixed/changing), or availability of resources (supply/demand).
- The Accounting mechanism maintains the actual usage of resources by requests so that the final cost can be computed and charged to users.
- In addition, the maintained historical usage information can be utilized by the Service Request Examiner and Admission Control mechanism to improve resource allocation decisions.

ARCHITECTURAL DESIGN CHALLENGES

Six open challenges in cloud architecture development.

Challenge 1—Service Availability and Data Lock-in Problem

The management of a cloud service by a single company is often the source of single points of failure. To achieve HA, one can consider using multiple cloud providers. Even if a company has multiple data centers located in different geographic regions, it may have common software infrastructure and accounting systems. Therefore, using multiple cloud providers may provide more protection from failures. Another availability obstacle is distributed *denial of service* (DDoS) attacks. Criminals threaten to cut off the incomes of SaaS providers by making their services unavailable. Some utility computing services offer SaaS providers the opportunity to defend against DDoS attacks by using quick scale-ups.

Challenge 2—Data Privacy and Security Concerns

Current cloud offerings are essentially public (rather than private) networks, exposing the system to more attacks. Many obstacles can be overcome immediately with well understood technologies such as encrypted storage, virtual LANs, and network middle boxes (e.g., firewalls, packet filters). For example, you could encrypt your data before placing it in a cloud. Many nations have laws requiring SaaS providers to keep customer data and copyrighted material within national boundaries. Traditional network attacks include buffer overflows, DoS attacks, spyware, malware, rootkits, Trojan horses, and worms.

Challenge 3—Unpredictable Performance and Bottlenecks
Multiple VMs can share CPUs and main memory in cloud computing, but I/O sharing is problematic. For example, to run 75 EC2 instances with the STREAM benchmark requires a mean bandwidth of 1,355 MB/second. However, for each of the 75 EC2 instances to write 1 GB files to the local disk requires a mean disk write bandwidth of only 55 MB/second.
This demonstrates the problem of I/O interference between VMs. One solution is to improve I/O architectures and operating systems to efficiently virtualize interrupts and I/O channels. Internet applications continue to become more data-intensive.

Challenge 4—Distributed Storage and Widespread Software Bugs
The database is always growing in cloud applications. The opportunity is to create a storage system that will not only meet this growth, but also combine it with the cloud advantage of scaling arbitrarily up and down on demand. This demands the design of efficient distributed SANs. Data centers must meet programmers' expectations in terms of scalability, data durability, and HA. Data consistence checking in SAN-connected data centers is a major challenge in cloud computing.
Large-scale distributed bugs cannot be reproduced, so the debugging must occur at a scale in the production data centers. No data center will provide such a convenience. One solution may be a reliance on using VMs in cloud computing. The level of virtualization may make it possible to capture valuable information in ways that are impossible without using VMs.

Challenge 5—Cloud Scalability, Interoperability, and Standardization
The pay-as-you-go model applies to storage and network bandwidth; both are counted in terms of the number of bytes used. Computation is different depending on virtualization level. GAE automatically scales in response to load increases and decreases; users are charged by the cycles used. AWS charges by the hour for the number of VM instances used, even if the machine is idle. The opportunity here is to scale quickly up and down in response to load variation, in order to save money, but without violating SLAs.
Open Virtualization Format (OVF) describes an open, secure, portable, efficient, and extensible format for the packaging and distribution of VMs. It also defines a format for distributing software to be deployed in VMs. In terms of cloud standardization, we suggest the ability for virtual appliances to run on any virtual platform. We also need to enable VMs to run on heterogeneous hardware platform hypervisors. This requires hypervisor-agnostic VMs.

Challenge 6—Software Licensing and Reputation Sharing
Many cloud computing providers originally relied on open source software because the licensing model for commercial software is not ideal for utility computing. The primary opportunity is either for open source to remain popular or simply for commercial software companies to change their licensing structure to better fit cloud computing. One can consider using both pay-for-use and bulk-use licensing schemes to widen the business coverage. One customer's bad behavior can affect the reputation of the entire cloud.
Another legal issue concerns the transfer of legal liability. Cloud providers want legal liability to remain with the customer, and vice versa. This problem must be solved at the SLA level.

CLOUD PROGRAMMING & SOFTWARE

Parallel and Distributed Programming Paradigms
A parallel and distributed program as a parallel program running on a set of computing engines or a distributed computing system. The term carries the notion of two fundamental terms in computer science: distributed computing system and parallel computing. A 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.
Parallel computing is the simultaneous use of more than one computational engine (not necessarily connected via a network) to run a job or an application. For instance, parallel computing may use either a distributed or a nondistributed computing system such as a multiprocessor platform. 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

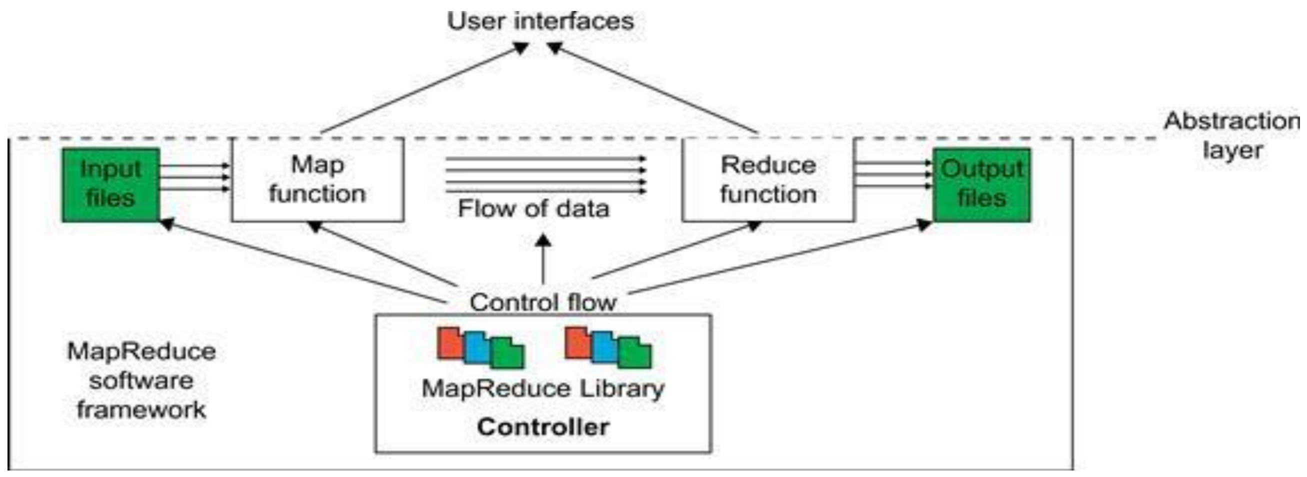
computing systems' standpoint, it increases throughput and resource utilization. Running a parallel program on a distributed computing system, however, could be a very complicated process.

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. In other words, upon identifying parallelism in the structure of the program, it can be divided into parts to be run on different workers. Different parts may process different data or a copy of the same data.
• **Data partitioning** This splits the input or intermediate data into smaller pieces. Similarly, upon identification of parallelism in the input data, it can also be divided into pieces to be processed on different workers. Data pieces may be processed by different parts of a program or a copy of the same program.
• **Mapping** This assigns the either smaller parts of a program or the smaller pieces of data to underlying resources. This process aims to appropriately assign such parts or pieces to be run simultaneously on different workers and is usually handled by resource allocators in the system.
• **Synchronization** Because different workers may perform different tasks, synchronization and coordination among workers is necessary so that race conditions are prevented and data dependency among different workers is properly managed. Multiple accesses to a shared resource by different workers may raise race conditions, whereas data dependency happens when a worker needs the processed data of other workers.
• **Communication** Because data dependency is one of the main reasons for communication among workers, communication is always triggered when the intermediate data is sent to workers.
• **Scheduling** For a job or program, when the number of computation parts (tasks) or data pieces is more than the number of available workers, a scheduler selects a sequence of tasks or data pieces to be assigned to the workers. It is worth noting that the resource allocator performs the actual mapping of the computation or data pieces to workers, while the scheduler only picks the next part from the queue of unassigned tasks based on a set of rules called the scheduling policy. For multiple jobs or programs, a scheduler selects a sequence of jobs or programs to be run on the distributed computing system. In this case, scheduling is also necessary when system resources are not sufficient to simultaneously run multiple jobs or programs.

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. 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.



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 welldefined interfaces in the form of two functions: *Map* and *Reduce*. These two main functions can be overridden by the user to achieve specific objectives. Figure shows the MapReduce framework with data flow and control flow.

Therefore, the user overrides the *Map* and *Reduce* functions first and then invokes the provided *MapReduce* (*Spec*, *& Results*) function from the library to start the flow of data. The MapReduce function, *MapReduce* (*Spec*, *& Results*), takes an important parameter which is a specification object, the *Spec*. This object is first initialized inside the user's program, and then the user writes code to fill it with the names of input and output files, as well as other optional tuning parameters. This object is also filled with the name of the *Map* and *Reduce* functions to identify these user-defined functions to the MapReduce library.

The overall structure of a user's program containing the Map, Reduce, and the Main functions is given below. 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 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. 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 (Figure 2).

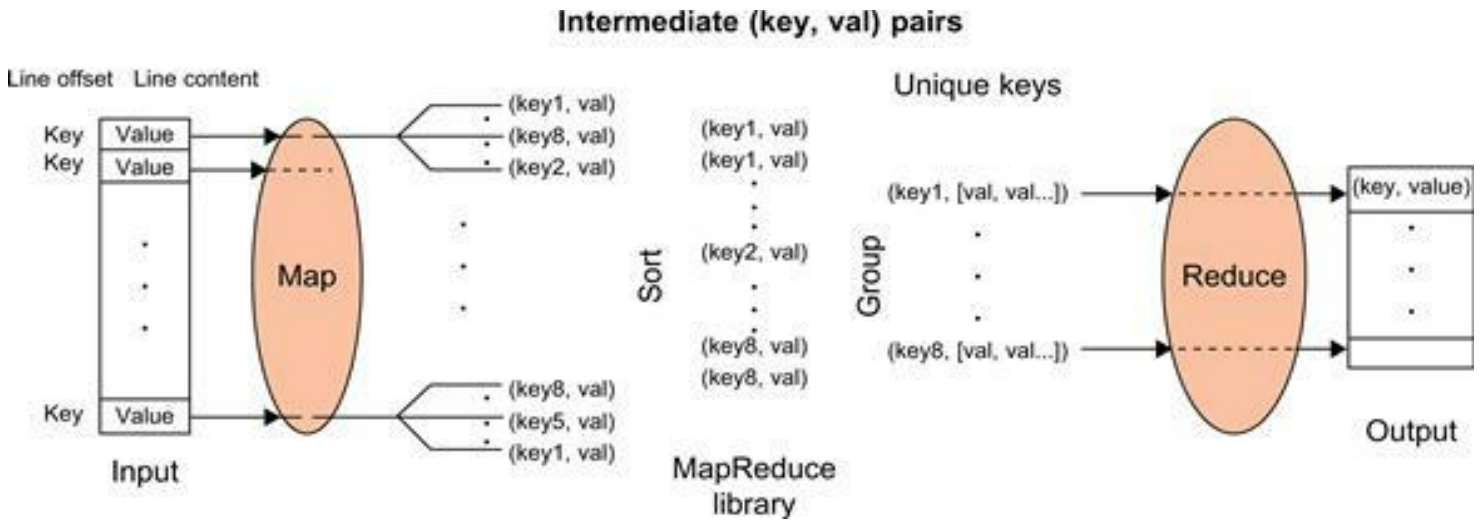


FIGURE 2 MapReduce logical data flow in 5 processing stages over successive (key, value) pairs.

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])*. In fact, 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.
To clarify the data flow in a sample MapReduce application, one of the well-known MapReduce problems, namely word count, to count the number of occurrences of each word in a collection of documents is presented here. Figure 3 demonstrates the data flow of the word-count problem for a simple input file containing only two lines as follows: (1) "most people ignore most poetry" and (2) "most poetry ignores most people." In this case, 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)*.

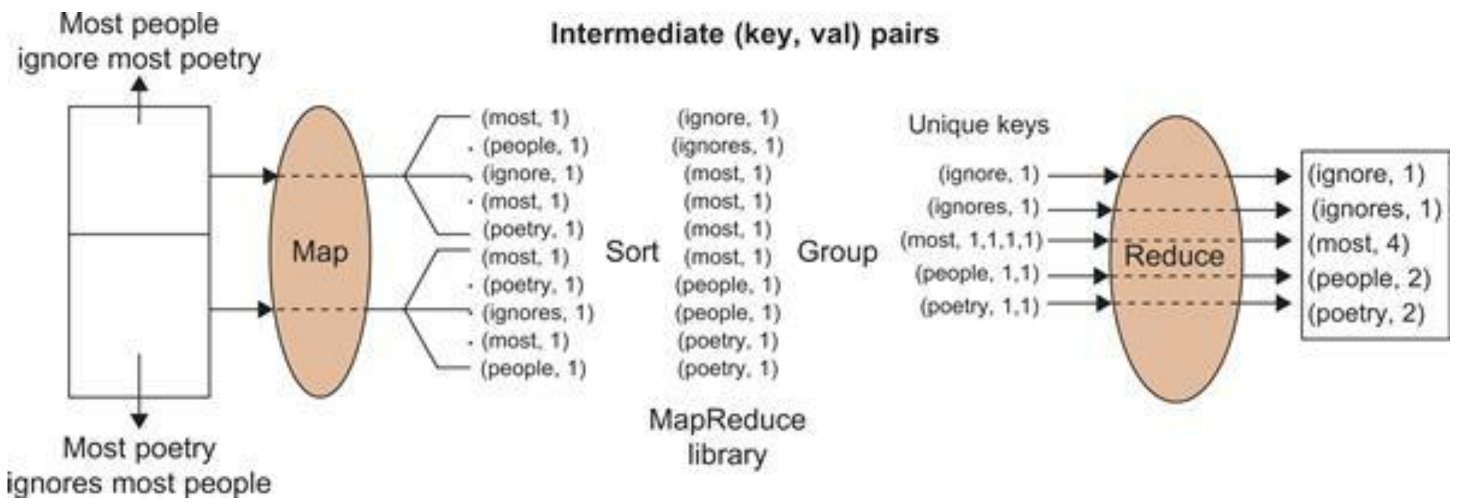


FIGURE 3

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:

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

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. It then groups the values of all occurrences of the same key. 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 Problems
As mentioned earlier, 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. Then the intermediate (key, value) pairs as the output of the *Map* function will be automatically found. The following three examples explain how to define keys and values in such problems:

Problem 1: Counting the number of occurrences of each word in a collection of documents.
Solution: unique "key": each word, intermediate "value": number of occurrences
Problem 2: Counting the number of occurrences of words having the same size, or the same number of letters, in a collection of documents
Solution: unique "key": each word, intermediate "value": size of the word

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

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. Therefore, the MapReduce framework meticulously handles all partitioning, mapping, synchronization, communication, and scheduling details of such data flows. Summarize this steps in the following steps:-
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. Therefore, the MapReduce library only generates copies of a user program 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. Therefore, 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** Each map worker reads its corresponding portion of the input data, namely the input data split, and sends it to its *Map* function. Although a map worker may run more than one *Map* function, which means it has been assigned more than one input data split, each worker is usually assigned one input split only.

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. As mentioned in our discussion of logical data flow, the MapReduce framework sorts and groups the data before it is processed by the *Reduce* function. Similarly, the MapReduce framework will also sort and group the local data on each map worker if the user invokes the *Combiner* function.

7. ***Partitioning* function** The MapReduce data flow, the intermediate (key, value) pairs with identical keys are grouped together because all values inside each group should be processed by only one *Reduce* function to generate the final result. However, in real implementations, since there are *M* map and *R* reduce tasks, intermediate (key, value) pairs with the same key might be produced by different map tasks, although they should be grouped and processed together by one *Reduce* function only.
Therefore, 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 1 reads the data of region 1 of all map workers, all (key, value) pairs with the same key will be gathered by reduce worker 1 accordingly (Figure 4). To implement this technique, a *Partitioning* function could simply be a hash function (e.g., *Hash*(*key*) *mod R*) that forwards the data into particular regions. It is also worth noting that the locations of the buffered data in these *R* partitions are sent to the master for later forwarding of data to the reduce workers.

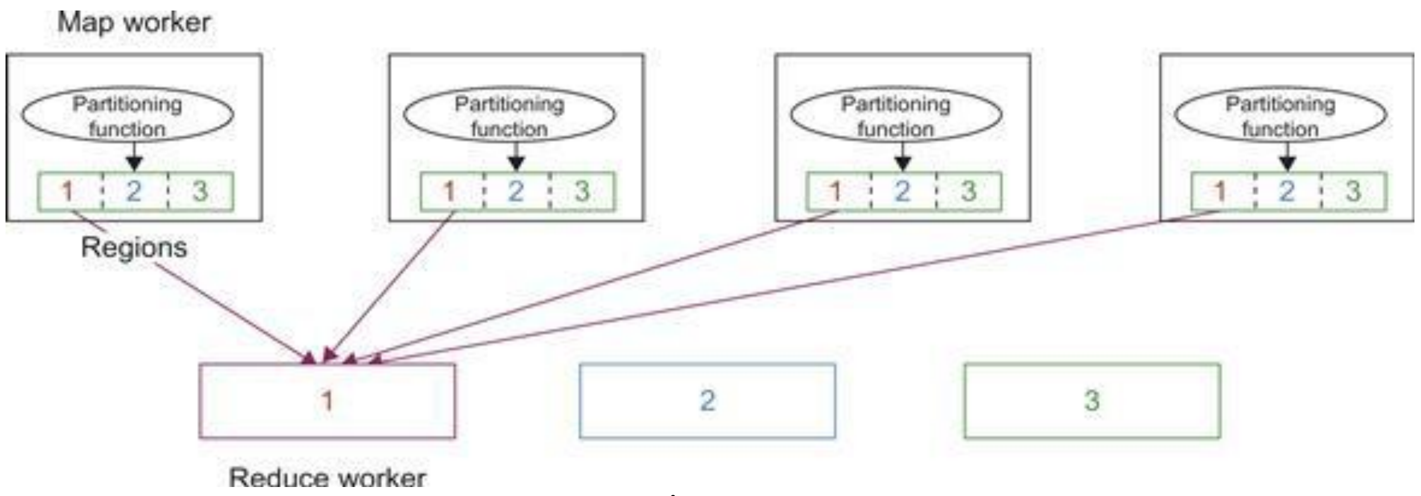


Figure 4

Figure 5 shows the data flow implementation of all data flow steps.
The following are two networking steps:
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 1, already notified of the location of region 1 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.

Steps 10 and 11 correspond to the reduce worker domain:
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. Note that the buffered data is sorted and grouped because the number of unique keys produced by a map worker may be more than *R* regions in which more than one key exists in each region of a map worker (Figure 4).

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.

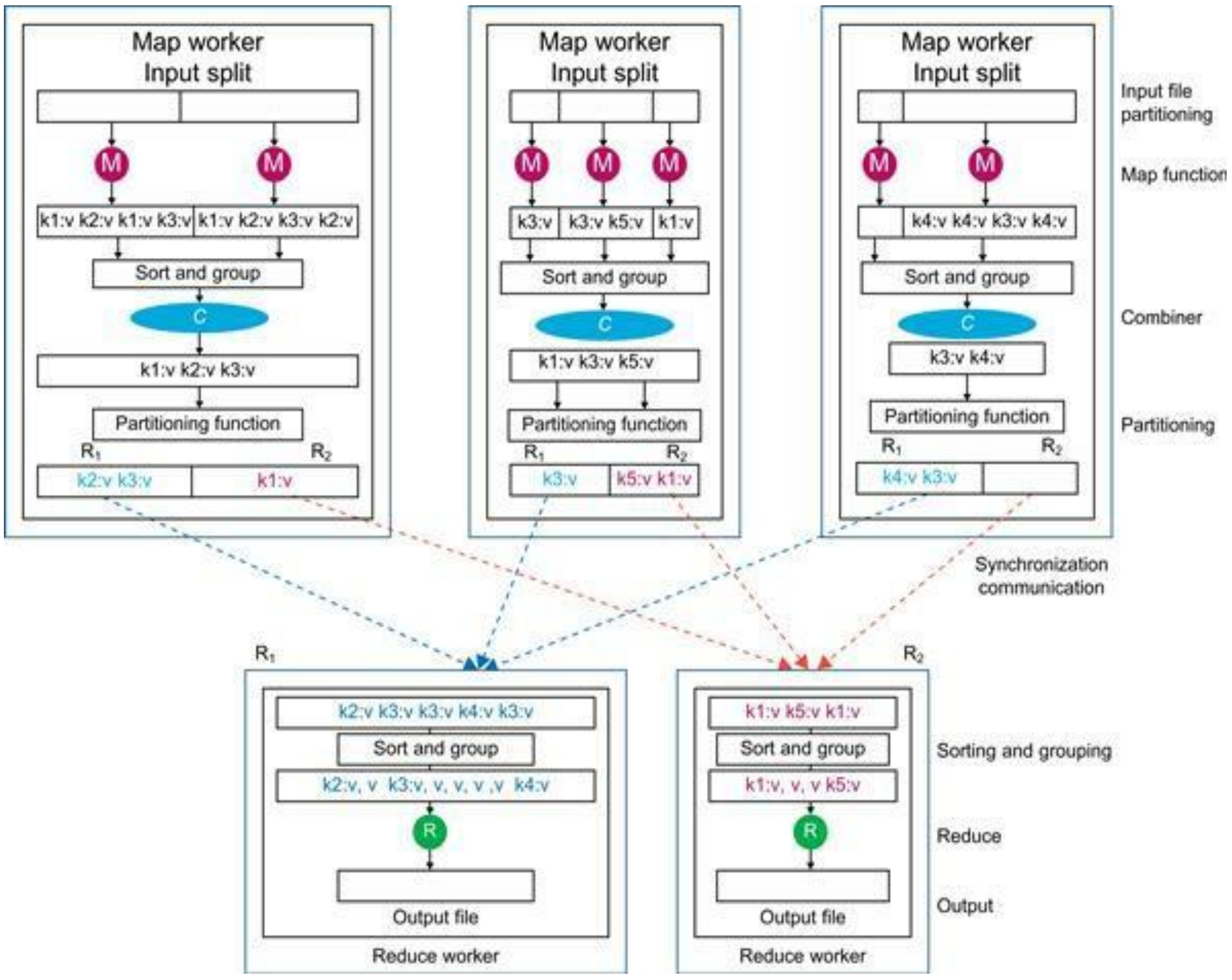
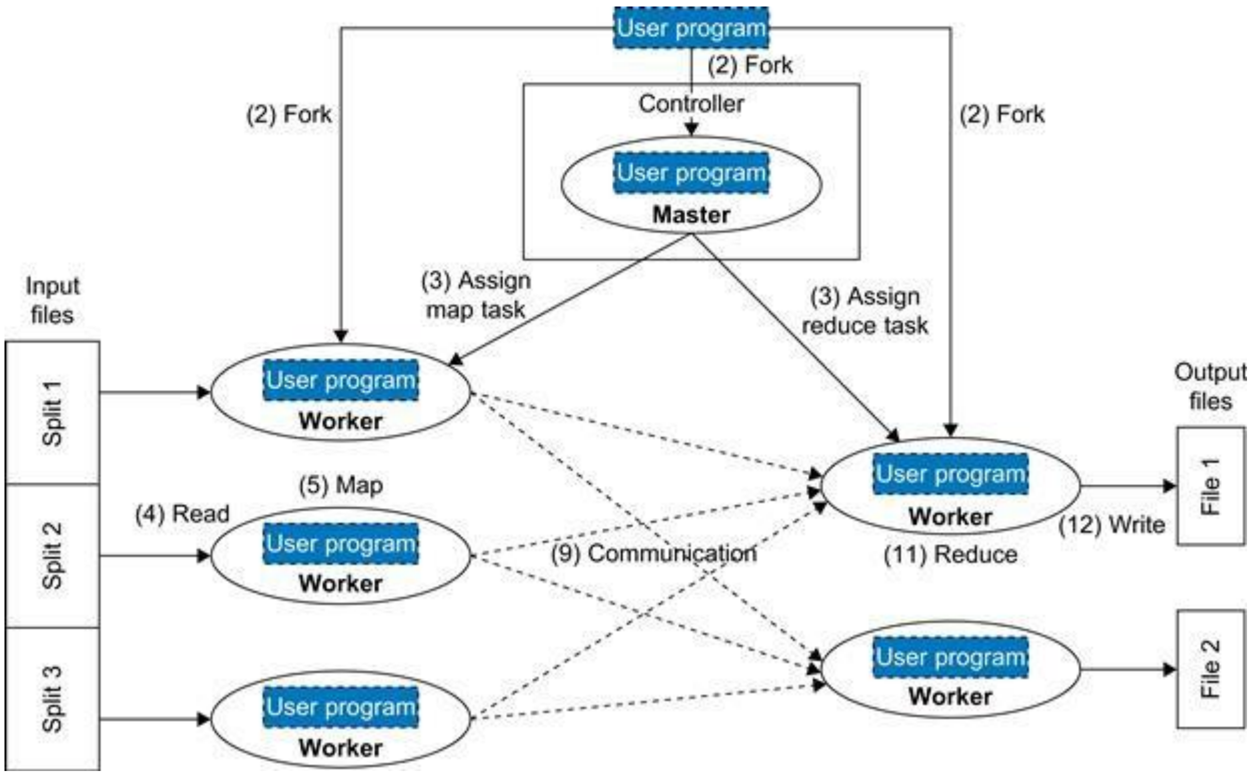


Figure 5



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

Hadoop

Hadoop is an open source implementation of MapReduce coded and released in Java by Apache. 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.

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

Features of HDFS

HDFS Fault Tolerance: One of the main aspects of HDFS is its fault tolerance characteristic. Since Hadoop is designed to be deployed on low-cost hardware by default, a hardware failure in this system is considered to be common rather than an exception. Therefore, Hadoop considers the following issues to fulfill reliability requirements of the file system.

• **Block replication** To reliably store data in HDFS, file blocks are replicated in this system. In other words, HDFS stores a file as a set of blocks and each block is replicated and distributed across the whole cluster. The replication factor is set by the user and is three by default.

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

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

HDFS High-Throughput Access to Large Data Sets (Files): Because HDFS is primarily designed for batch processing rather than interactive processing, data access throughput in HDFS is more important than latency. Also, because applications run on HDFS typically have large data sets, individual files are broken into large blocks (e.g., 64 MB) to allow HDFS to decrease the amount of metadata storage required per file.
This provides two advantages: The list of blocks per file will shrink as the size of individual blocks increases, and by keeping large amounts of data sequentially within a block, HDFS provides fast streaming reads of data.

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

• **Reading a file** To read a file in HDFS, a user sends an "open" request to the NameNode to get the location of file blocks. For each file block, the NameNode returns the address of a set of DataNodes containing replica information for the requested file. The number of addresses depends on the number of block replicas.

Upon receiving such information, the user calls the *read* function to connect to the closest DataNode containing the first block of the file. After the first block is streamed from the respective DataNode to the user, the established connection is terminated and the same process is repeated for all blocks of the requested file until the whole file is streamed to the user.

• **Writing to a file** To write a file in HDFS, a user sends a "create" request to the NameNode to create a new file in the file system namespace. If the file does not exist, the NameNode notifies the user and allows him to start writing data to the file by calling the *write* function. The first block of the file is written to an internal queue termed the data queue while a data streamer monitors its writing into a DataNode. Since each file block needs to be replicated by a predefined factor, the data streamer first sends a request to the NameNode to get a list of suitable DataNodes to store replicas of the first block.
The steamer then stores the block in the first allocated DataNode. Afterward, the block is forwarded to the second DataNode by the first DataNode. The process continues until all allocated DataNodes receive a replica of the first block from the previous DataNode. Once this replication process is finalized, the same process starts for the second block and continues until all blocks of the file are stored and replicated on the file system.

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. Shows the MapReduce engine architecture cooperating with HDFS. Similar to HDFS, the MapReduce engine also has a master/slave architecture consisting of a single JobTracker as the master and a number of TaskTrackers as the slaves (workers). 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.

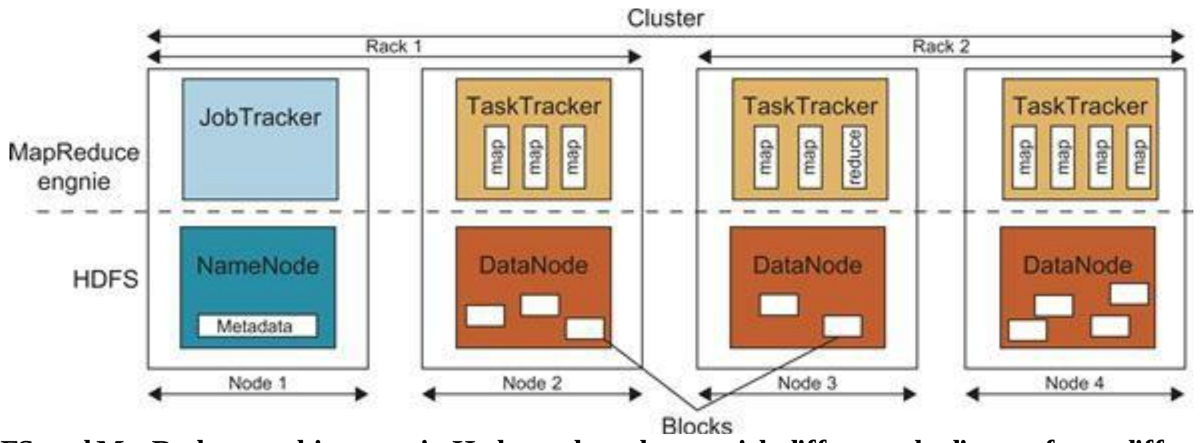


Fig. HDFS and MapReduce architecture in Hadoop where boxes with different shadings refer to different functional nodes applied to different blocks of data.

Each TaskTracker node has a number of simultaneous execution slots, each executing either a map or a reduce task. Slots are defined as the number of simultaneous threads supported by CPUs of the TaskTracker node. For example, a TaskTracker node with N CPUs, each supporting M threads, has $M * N$ simultaneous execution slots. It is worth noting that each data block is processed by one map task running on a single slot. Therefore, there is a one-to-one correspondence between map tasks in a TaskTracker and data blocks in the respective DataNode.

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. Figure depicts the data flow of running a MapReduce job in Hadoop.

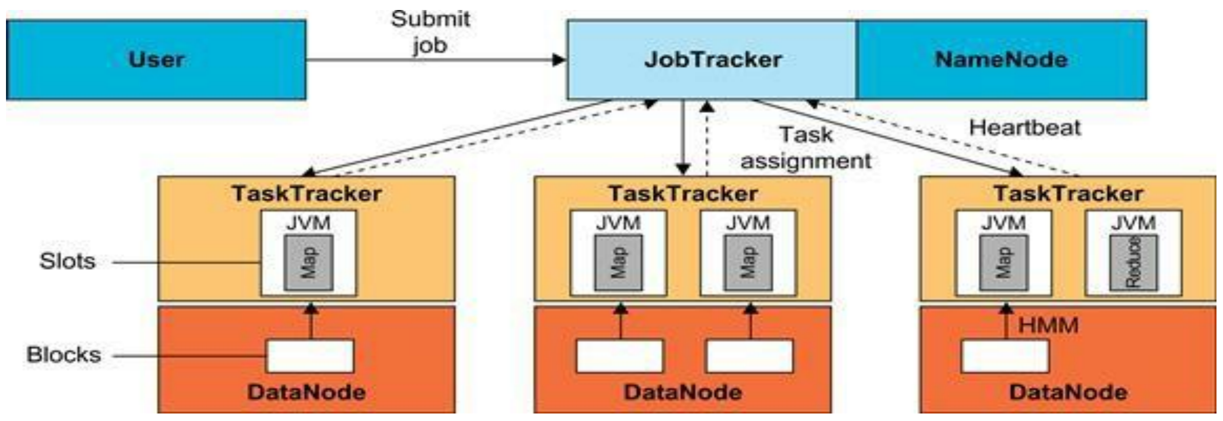


FIGURE Data Flow In Running A MapReduce Job At Various Task Trackers Using The Hadoop Library.

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.

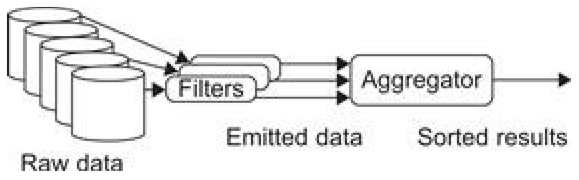
Task execution The control flow to execute a task (either map or reduce) starts inside the TaskTracker by copying the job JAR file to its file system. Instructions inside the job JAR file are executed after launching a Java Virtual Machine (JVM) to run its map or reduce task.

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.

HIGH LEVEL LANGUAGE FOR CLOUD

Sawzall High-Level Languages

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. As with MapReduce, 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. Sawzall was developed by Rob Pike with an initial goal of processing Google's log files. In this regard it was hugely successful and changed a batch daylong enterprise into an interactive session, enabling new approaches to using such data to be developed. shows the overall model of data flow and processing procedures in the Sawzall framework.



The overall flow of filtering, aggregating, and collating in Sawzall.

First the data is partitioned and processed locally with an on-site processing script. The local data is filtered to get the necessary information. The aggregator is used to get the final results based on the emitted data. Many of Google's applications fit this model. Users write their applications using the Sawzall scripting language. 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 High level Language

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

Pig is a high-level platform for creating MapReduce programs used with Hadoop. The language for this platform is called Pig Latin. Pig Latin abstracts the programming from the Java MapReduce idiom into a notation which makes MapReduce programming high level.

At the present time, Pig's infrastructure layer consists of a compiler that produces sequences of Map-Reduce programs, for which large-scale parallel implementations already exist. Pig's language layer currently consists of a textual language called Pig Latin.

Key properties of Pig Latin:

Ease of programming. It is trivial to achieve parallel execution of simple, "embarrassingly parallel" data analysis tasks. Complex tasks comprised of multiple interrelated data transformations are explicitly encoded as data flow sequences, making them easy to write, understand, and maintain.

Optimization opportunities. The way in which tasks are encoded permits the system to optimize their execution automatically, allowing the user to focus on semantics rather than efficiency.

Extensibility. Users can create their own functions to do special-purpose processing.

PROGRAMMING THE GOOGLE APP ENGINE

Figure summarizes some key features of GAE programming model for two supported languages: Java and Python. A client environment that includes an *Eclipse* plug-in for Java allows you to debug your GAE on your local machine. Also, the GWT Google Web Toolkit is available for Java web application developers. Developers can use this, or any other language using a JVM based interpreter or compiler, such as JavaScript or Ruby. 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.

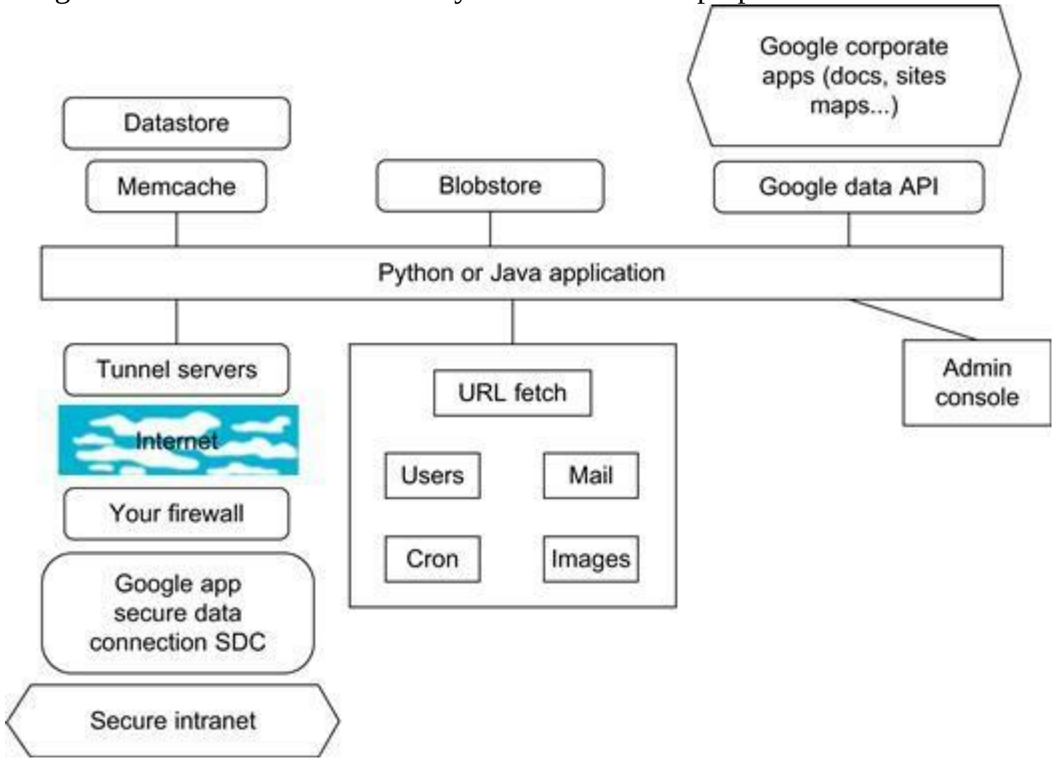


FIGURE Programming Environment for Google AppEngine.

Java offers Java Data Object (JDO) and Java Persistence API (JPA) interfaces implemented by the open source Data Nucleus Access platform, while Python has a SQL-like query language called GQL. The data store is strongly consistent and uses optimistic concurrency control.

An update of an entity occurs in a transaction that is retried a fixed number of times if other processes are trying to update the same entity simultaneously. Your application can execute multiple data store operations in a single transaction which either all succeed or all fail together. The data store implements transactions across its distributed network using "entity groups." A transaction manipulates entities within a single group. Entities of the same group are stored together for efficient execution of transactions. Your GAE application can assign entities to groups when the entities are created. The performance of the data store can be enhanced by in-memory caching using the *memcache*, which can also be used independently of the data store.

Recently, Google added the *blobstore* which is suitable for large files as its size limit is 2 GB. There are several mechanisms for incorporating external resources. The *Google SDC Secure Data Connection* can tunnel through the Internet and 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 high-speed Google infrastructure that retrieves web pages for many other Google products. There are dozens of Google "corporate" facilities including maps, sites, groups, calendar, docs, and YouTube, among others. These support the *Google Data API* which can be used inside GAE.

An application can use Google Accounts for *user* authentication. Google Accounts handles user account creation and sign-in, and a user that already has a Google account (such as a Gmail account) can use that account with your app. GAE provides the ability to manipulate image data using a dedicated *Images* service which can resize, rotate, flip, crop, and enhance images. An application can perform tasks outside of responding to web requests. Your application can perform these tasks on a schedule that you configure, such as on a daily or hourly basis using "cron jobs," handled by the *Cron* service.