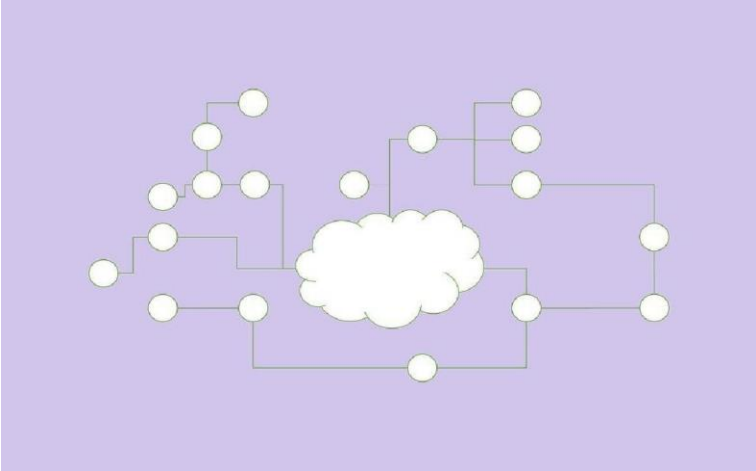
UNIT V CLOUD TECHNOLOGIES AND ADVANCEMENTS 8

Hadoop – MapReduce – Virtual Box -- Google App Engine – Programming Environment for Google App Engine — Open Stack – Federation in the Cloud – Four Levels of Federation – Federated Services and Applications – Future of Federation.

Hadoop Vs. CloudComputing



Hadoop is an open source programming framework based on Java which supports the processing and storage of large volumes of data sets in a distributed computing environment. Hadoop is a part of Apache project, which is sponsored by Apache Software Foundation.It makes it possible to handle thousands of terabytes of data, and to run applications on systems with thousands of commodity hardware nodes. Hadoop has a distributed file system which allows faster transfer of data amongst nodes. It also allows systems to continue operating even if a node stops working.

This reduces the risks of a system failure and unforeseen data loss, even if a significant number of nodes stop operating. Therefore, Hadoop has emerged as a foundation for processing big data tasks like scientific analysis, processing large volumes of sensor data and business and sales planning.

Cloud computing

In simplest terms, it means storing and accessing your data, programs, and files over the internet rather than your PC‘s hard drive. Basically, the cloud is another name for the internet.

Cloud computing has several attractive benefits for end users and businesses. The primary benefits of cloud computing includes:

- **Elasticity** – with cloud computing, businesses only use the resources they require. Organizations can increase their usage as computing needs increase and reduce their usage as the computing needs decrease. This eliminates the need for investing heavily in IT infrastructures which may or may not be used.
- **Self-service provisioning** – users can always use the resources for almost any type of workload on demand. This eliminates the need for IT admins to provide and manage computer resources.
- **Pay-per-use** – compute resources are measures depending on the usage level. This means that users are only charged for the cloud resources they use. Cloud computing deployment models Cloud computing services can be deployed in three models, that is, public, private and hybrid.
- **Public clouds** – Services are usually provided by a third party over the internet. In public clouds, services are sold on demand, mostly by the minute or per hour. Clients pay only for the bandwidth, storage or CPU cycles they use.
- **Private clouds** – This model allows data to be delivered from a business‘ data center to internal users. Private clouds are very flexible and convenient, and they preserve the management, control, and security of the data centers.
- **Hybrid cloud** – This is the combination of both public and private clouds, which involves automation and orchestration between the two models. In hybrid clouds, organizations can run critical operations or sensitive applications on the private cloud and use the public cloud for general workloads.

The differences between Hadoop and Cloud Computing

1. Hadoop is a framework which uses simple programming models to process large data sets across clusters of computers. It is designed to scale up from single servers to thousands of machines, which offer local computation and storage individually. Cloud computing, on the other hand, constitutes various computing concepts, which involve a large number of computers which are usually connected through a real-time communication network.
2. Cloud computing focuses on on-demand, scalable and adaptable service models. Hadoop on the other hand all about extracting value out of volume, variety, and velocity.
3. In cloud computing, Cloud MapReduce is a substitute implementation of MapReduce. The main difference between cloud MapReduce and Hadoop is that Cloud MapReduce doesn‘t provide its own implementation; rather, it relies on the infrastructure offered by different cloud services providers.

4. Hadoop is an _ecosystem‘ of open source software projects which allow cheap computing which is well distributed on industry-standard hardware. On the other hand, cloud computing is a model where processing and storage resources can be accessed from any location via the internet.

Hadoop MapReduce

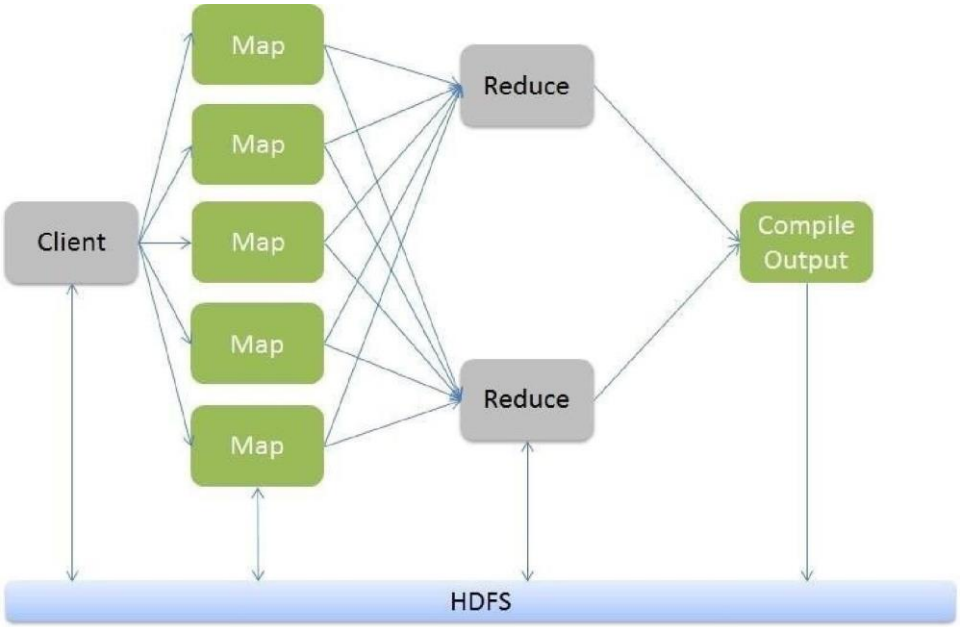
Hadoop MapReduce is a programming paradigm at the heart of Apache Hadoop for providing massive scalability across hundreds or thousands of Hadoop clusters on commodity hardware.

The MapReduce model processes large unstructured data sets with a distributed algorithm on a Hadoop cluster.

The term MapReduce represents two separate and distinct tasks Hadoop programs perform-Map Job and Reduce Job. Map job scales takes data sets as input and processes them to produce key value pairs. Reduce job takes the output of the Map job i.e. the key value pairs and aggregates them to produce desired results. The input and output of the map and reduce jobs are stored in HDFS.

What is MapReduce?

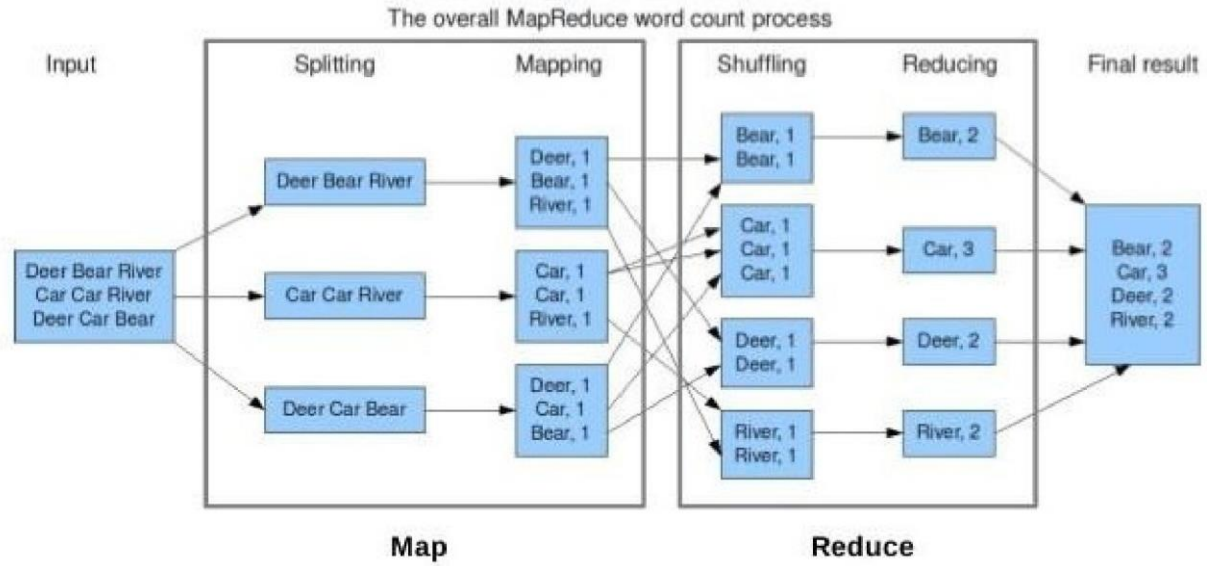
Hadoop MapReduce (Hadoop Map/Reduce) is a software framework for distributed processing of large data sets on computing clusters. It is a sub-project of the Apache Hadoop project. Apache Hadoop is an open-source framework that allows to store and process big data in a distributed environment across clusters of computers using simple programming models. MapReduce is the core component for data processing in Hadoop framework. In layman‘s term Mapreduce helps to split the input data set into a number of parts and run a program on all data parts parallel at once. The term MapReduce refers to two separate and distinct tasks. The first is the map operation, takes a set of data and converts it into another set of data, where individual elements are broken down into tuples (key/value pairs). The reduce operation combines those data tuples based on the key and accordingly modifies the value of the key.



Bear, Deer, River and Car Example

The following word count example explains MapReduce method. For simplicity, let's consider a few words of a text document. We want to find the number of occurrence of each word. First the input is split to distribute the work among all the map nodes as shown in the figure. Then each word is identified and mapped to the number one. Thus the pairs also called as tuples are created. In the first mapper node three words Deer, Bear and River are passed. Thus the output of the node will be three key, value pairs with three distinct keys and value set to one. The mapping process remains the same in all the nodes. These tuples are then passed to the reduce nodes. A partitioner comes into action which carries out shuffling so that all the tuples with same key are sent to same node.

The Reducer node processes all the tuples such that all the pairs with same key are counted and the count is updated as the value of that specific key. In the example there are two pairs with the key _Bear‘ which are then reduced to single tuple with the value equal to the count. All the output tuples are then collected and written in the output file.

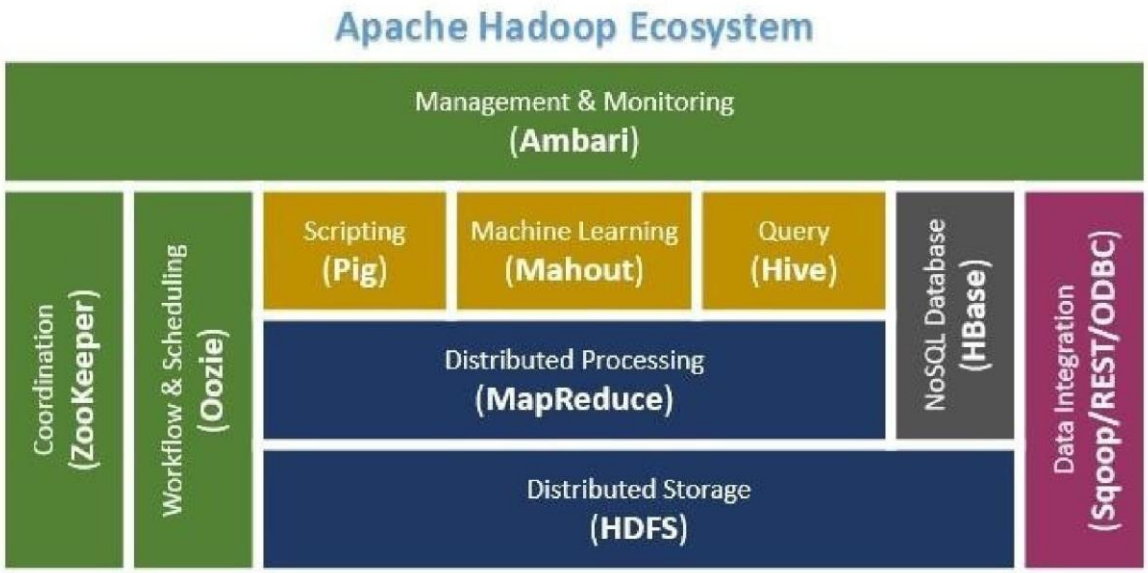


How is MapReduce used?

Various platforms are designed over Hadoop for easier querying, summarization. For instance, Apache Mahout provides machine learning algorithms that are implemented over Hadoop. Apache Hive provides data summarization, query, and analysis over the data stored in HDFS.

Learn Hadoop by working on interesting Big Data and Hadoop Projects for just $9

MapReduce is primarily written in Java, therefore more often than not, it is advisable to learn Java for Hadoop MapReduce.MapReduce libraries have been written in many programming languages. Though it is mainly implemented in Java, there are non-Java interfaces such as Streaming(Scripting Languages), Pipes(C++), Pig, Hive, Cascading. In case of Streaming API, the corresponding jar is included and the mapper and reducer are written in Python/Scripting language. Hadoop which in turn uses MapReduce technique has a lot of use cases. On a general note it is used in scenario of needle in a haystack or for continuous monitoring of a huge system statistics. One such example is monitoring the traffic in a country road network and handling the traffic flow to prevent a jam. One common example is analyzing and storage of twitter data. It is also used in Log analysis which consists of various summations.

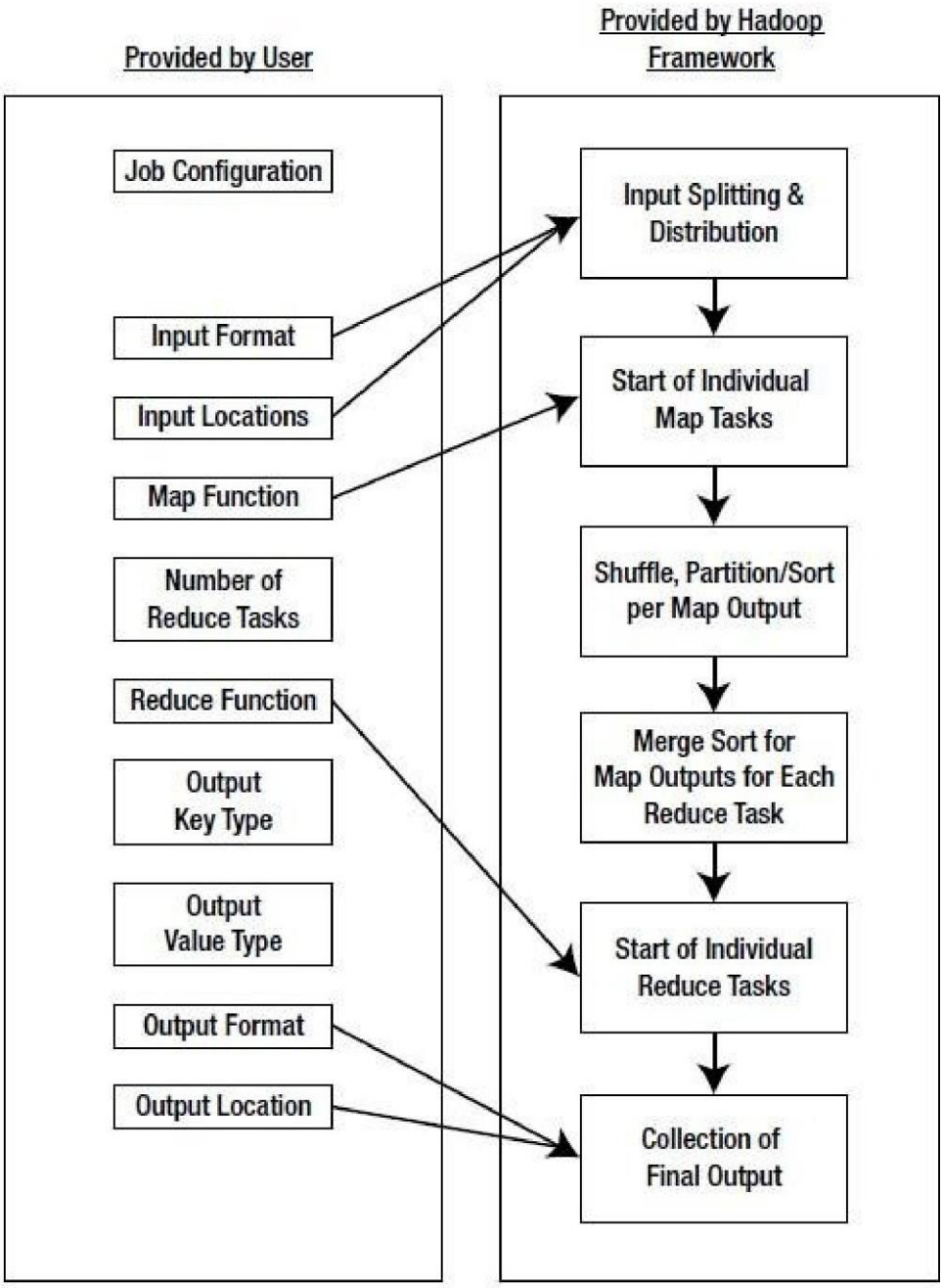


MapReduce Architecture

The figure shown below illustrates the various parameters and modules that can be configured during a MapReduce operation:

JobConf is the framework used to provide various parameters of a MapReduce job to the Hadoop for execution. The Hadoop platforms executes the programs based on configuration set using JobConf. The parameters being Map Function, Reduce Function, combiner , Partitioning function, Input and Output format. Partitioner controls the shuffling of the tuples when being sent from Mapper node to Reducer nodes. The total number of partitions done in the tuples is equal to the number of reduce nodes. In simple terms based on the function output the tuples are transmitted through different reduce nodes.

Input Format describes the format of the input data for a MapReduce job. Input location specifies the location of the datafile. Map Function/ Mapper convert the data into key value pair. For example let‘s consider daily temperature data of 100 cities for the past 10 years. In this the map function is written such a way that every temperature being mapped to the corresponding city. Reduce Function reduces the set of tuples which share a key to a single tuple with a change in the value. In this example if we have to find the highest temperature recorded in the city the reducer function is written in such a way that it return the tuple with highest value i.e: highest temperature in the city in that sample data.



The number of Map and Reduce nodes can also be defined. You can set Partitioner function which partitions and transfer the tuples which by default is based on a hash function. In other words we can set the options such that a specific set of key value pairs are transferred to a specific reduce task. For example if your key value consists of the year it was recorded, then we can set the parameters such that all the keys of specific year are transferred to a same reduce task. The Hadoop framework consists of a single master and many slaves. Each master has JobTracker and

each slave has TaskTracker. Master distributes the program and data to the slaves. Task tracker, as the name suggests keep track of the task directed to it and relays the information to the JobTracker. The JobTracker monitors all the status reports and re-initiates the failed tasks if any. Combiner class are run on map task nodes. It takes the tuples emitted by Map nodes as input. It basically does reduce operation on the tuples emitted by the map node. It is like a pre- reduce task to save a lot of bandwidth. We can also pass global constants to all the nodes using _Counters'. They can be used to keep track of all events in map and reduce tasks. For example we can pass a counter to calculate the statistics of an event beyond a certain threshold.

Oracle VM VirtualBox (formerly **Sun VirtualBox**, **Sun xVM VirtualBox** and **Innotek VirtualBox**) is a free and open-source hosted hypervisor for x86 virtualization, developed by Oracle Corporation. Created by Innotek, it was acquired by Sun Microsystems in 2008, which was in turn acquired by Oracle in 2010. VirtualBox may be installed on Windows, macOS, Linux, Solaris and OpenSolaris. There are also ports to FreeBSD and Genode. It supports the creation and management of guest virtual machines running Windows, Linux, BSD, OS/2, Solaris, Haiku, and OSx86,as well as limited virtualization of macOS guests on Apple hardware. For some guest operating systems, a "Guest Additions" package of device drivers and system applications is available, which typically improves performance, especially that of graphics.

VirtualBox was first offered by Innotek GmbH from Weinstadt, Germany, under a proprietary software license, making one version of the product available at no cost for personal or evaluation use, subject to the VirtualBox Personal Use and Evaluation License (PUEL). In January 2007, based on counsel by LiSoG, Innotek GmbH released VirtualBox Open Source Edition (OSE) as free and open-source software, subject to the requirements of the GNU General Public License (GPL), version 2.

Innotek GmbH also contributed to the development of OS/2 and Linux support in virtualization and OS/2 ports of products from Connectix which were later acquired by Microsoft. Specifically, Innotek developed the "additions" code in both Windows Virtual PC and Microsoft Virtual Server, which enables various host–guest OS interactions like shared clipboards or dynamic viewport resizing.

Google App Engine (GAE)

Google App Engine (GAE) is a service for developing and hosting Web applications in Google's data centers, belonging to the platform as a service (PaaS) category of cloud computing. Web

applications hosted on GAE are sandboxed and run across multiple servers for redundancy and allowing for scaling of resources according to the traffic requirements of the moment. App Engine automatically allocates additional resources to the servers to accommodate increased load. Google App Engine is Google's platform as a service offering that allows developers and businesses to build and run applications using Google's advanced infrastructure. These applications are required to be written in one of a few supported languages, namely: Java, Python, PHP and Go. It also requires the use of Google query language and that the database used is Google Big Table. Applications must abide by these standards, so applications either must be developed with GAE in mind or else modified to meet the requirements.

GAE is a platform, so it provides all of the required elements to run and host Web applications, be it on mobile or Web. Without this all-in feature, developers would have to source their own servers, database software and the APIs that would make all of them work properly together, not to mention the entire configuration that must be done. GAE takes this burden off the developers so they can concentrate on the app front end and functionality, driving better user experience.

Advantages of GAE include:

- Readily available servers with no configuration requirement
- Power scaling function all the way down to "free" when resource usage is minimal
- Automated cloud computing tools

Introducing Google App Engine

Google App Engine is a web application hosting service. By ―web application,‖ we mean an application or service accessed over the Web, usually with a web browser: storefronts with shopping carts, social networking sites, multiplayer games, mobile applications, survey applications, project management, collaboration, publishing, and all the other things we're discovering are good uses for the Web. App Engine can serve traditional website content too, such as documents and images, but the environment is especially designed for real-time dynamic applications. Of course, a web browser is merely one kind of client: web application infrastructure is well suited to mobile applications, as well.

In particular, Google App Engine is designed to host applications with many simultaneous users. When an application can serve many simultaneous users without degrading performance, we say it *scales*. Applications written for App Engine scale automatically. As more people use the application, App Engine allocates more resources for the application and manages the use of

those resources. The application itself does not need to know anything about the resources it is using.

Unlike traditional web hosting or self-managed servers, with Google App Engine, you only pay for the resources you use. Billed resources include CPU usage, storage per month, incoming and outgoing bandwidth, and several resources specific to App Engine services. To help you get started, every developer gets a certain amount of resources for free, enough for small applications with low traffic.

App Engine is part of Google Cloud Platform, a suite of services for running scalable applications, performing large amounts of computational work, and storing, using, and analyzing large amounts of data. The features of the platform work together to host applications efficiently and effectively, at minimal cost. App Engine‘s specific role on the platform is to host web applications and scale them automatically. App Engine apps use the other services of the platform as needed, especially for data storage.

An App Engine web application can be described as having three major parts: application instances, scalable data storage, and scalable services. In this chapter, we look at each of these parts at a high level. We also discuss features of App Engine for deploying and managing web applications, and for building websites integrated with other parts of Google Cloud Platform.

The Runtime Environment

An App Engine application responds to web requests. A web request begins when a client, typically a user‘s web browser, contacts the application with an HTTP request, such as to fetch a web page at a URL. When App Engine receives the request, it identifies the application from the domain name of the address, either a custom domain name you have registered and configured for use with the app, or an *.appspot.com* subdomain provided for free with every app. App Engine selects a server from many possible servers to handle the request, making its selection based on which server is most likely to provide a fast response. It then calls the application with the content of the HTTP request, receives the response data from the application, and returns the response to the client.From the application‘s perspective, the runtime environment springs into existence when the request handler begins, and disappears when it ends. App Engine provides several methods for storing data that persists between requests, but these mechanisms live outside of the runtime environment. By not retaining state in the runtime environment

between requests—or at least, by not expecting that state will be retained between requests—App Engine can distribute traffic among as many servers as it needs to give every request the same treatment, regardless of how much traffic it is handling at one time.

In the complete picture, App Engine allows runtime environments to outlive request handlers, and will reuse environments as much as possible to avoid unnecessary initialization. Each instance of your application has local memory for caching imported code and initialized data structures. App Engine creates and destroys instances as needed to accommodate your app‘s traffic. If you enable the multithreading feature, a single instance can handle multiple requests concurrently, further utilizing its resources.

Application code cannot access the server on which it is running in the traditional sense. An application can read its own files from the filesystem, but it cannot write to files, and it cannot read files that belong to other applications. An application can see environment variables set by App Engine, but manipulations of these variables do not necessarily persist between requests. An application cannot access the networking facilities of the server hardware, although it can perform networking operations by using services.

In short, each request lives in its own —sandbox.‖ This allows App Engine to handle a request with the server that would, in its estimation, provide the fastest response. For web requests to the app, there is no way to guarantee that the same app instance will handle two requests, even if the requests come from the same client and arrive relatively quickly. Sandboxing also allows App Engine to run multiple applications on the same server without the behavior of one application affecting another. In addition to limiting access to the operating system, the runtime environment also limits the amount of clock time and memory a single request can take. App Engine keeps these limits flexible, and applies stricter limits to applications that use up more resources to protect shared resources from ‗runaway‖ applications.

A request handler has up to 60 seconds to return a response to the client. While that may seem like a comfortably large amount for a web app, App Engine is optimized for applications that respond in less than a second. Also, if an application uses many CPU cycles, App Engine may

slow it down so the app isn‘t hogging the processor on a machine serving multiple apps. A CPU-intensive request handler may take more clock time to complete than it would if it had exclusive use of the processor, and clock time may vary as App Engine detects patterns in CPU usage and allocates accordingly.

Google App Engine provides four possible runtime environments for applications, one for each of four programming languages: Java, Python, PHP, and Go. The environment you choose depends on the language and related technologies you want to use for developing the application.

The Python environment runs apps written in the Python 2.7 programming language, using a custom version of CPython, the official Python interpreter. App Engine invokes a Python app using WSGI, a widely supported application interface standard. An application can use most of Python‘s large and excellent standard library, as well as rich APIs and libraries for accessing services and modeling data. Many open source Python web application frameworks work with App Engine, such as Django, web2py, Pyramid, and Flask. App Engine even includes a lightweight framework of its own, called webapp. Similarly, the Java, PHP, and Go runtime environments offer standard execution environments for those languages, with support for standard libraries and third-party frameworks.

All four runtime environments use the same application server model: a request is routed to an app server, an application instance is initialized (if necessary), application code is invoked to handle the request and produce a response, and the response is returned to the client. Each environment runs application code within sandbox restrictions, such that any attempt to use a feature of the language or a library that would require access outside of the sandbox returns an error.

You can configure many aspects of how instances are created, destroyed, and initialized. How you configure your app depends on your need to balance monetary cost against performance. If you prefer performance to cost, you can configure your app to run many instances and start new ones aggressively to handle demand. If you have a limited budget, you can adjust the limits that control how requests queue up to use a minimum number of instances.

I haven‘t said anything about which operating system or hardware configuration App Engine uses. There are ways to figure this out with a little experimentation, but in the end it doesn‘t matter: the runtime environment is an abstraction *above* the operating system that allows App Engine to manage resource allocation, computation, request handling, scaling, and load distribution without the application‘s involvement. Features that typically require knowledge of the operating system are either provided by services outside of the runtime environment, provided or emulated using standard library calls, or restricted in sensible ways within the definition of the sandbox.

Everything stated above describes how App Engine allocates application instances dynamically to scale with your application‘s traffic. In addition to a flexible bank of instances serving your primary traffic, you can organize your app into multiple ―modules.‖ Each module is addressable individually using domain names, and can be configured with its own code, performance characteristics, and scaling pattern— including the option of running a fixed number of always-on instances, similar to traditional servers. In practice, you usually use a bank of dynamically scaling instances to handle your ―frontend‖ traffic, then establish modules as ―backends‖ to be accessed by the frontends for various purposes.

Basic architecture

OpenStack Glance has a client-server architecture that provides a REST API to the user through which requests to the server can be performed.

A Glance Domain Controller manages the internal server operations that is divided into layers. Specific tasks are implemented by each layer.

All the file (Image data) operations are performed using glance_store library, which is responsible for interaction with external storage back ends and (or) local filesystem(s). The glance_store library provides a uniform interface to access the backend stores. Glance uses a central database (Glance DB) that is shared amongst all the components in the system and is sql-based by default. Other types of database backends are somewhat supported and used by operators but are not extensively tested upstream.

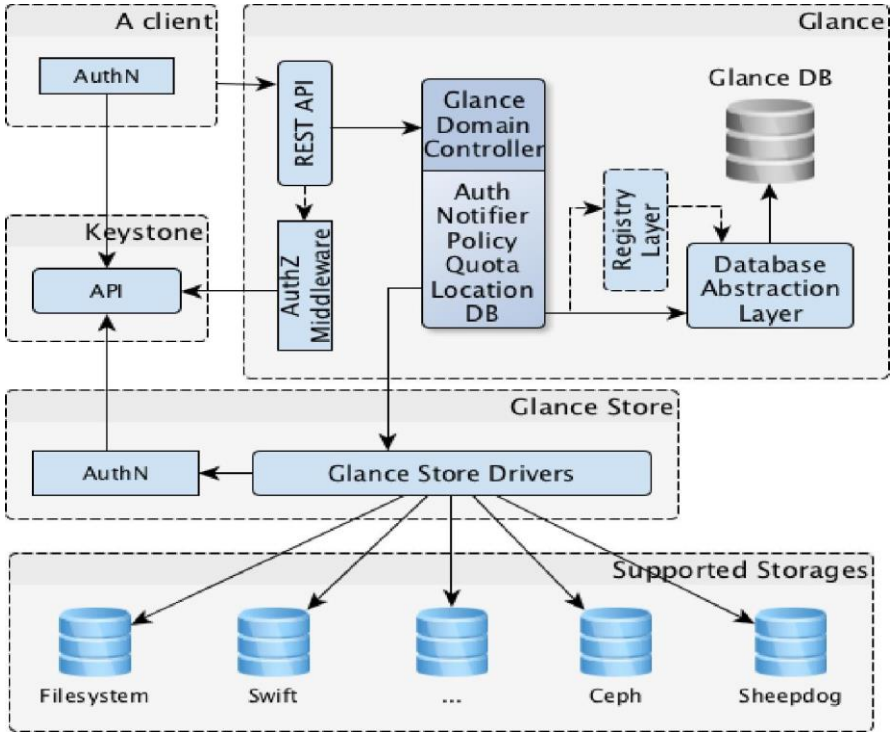


Image 1. OpenStack Glance Architecture : Following components are present in the Glance architecture:

- **A client** - any application that makes use of a Glance server.
- **REST API** - Glance functionalities are exposed via REST.
- **Database Abstraction Layer (DAL) - an application programming interface** (API) that unifies the communication between Glance and databases.
- **Glance Domain Controller - middleware that implements the main** Glance functionalities such as authorization, notifications, policies, database connections.
- **Glance Store - used to organize interactions between Glance and various** data stores.
- **Registry Layer - optional layer that is used to organise secure** communication between the domain and the DAL by using a separate service.

Cloud Federation

Intercloud researchers have shown the most interest in cloud federations because it enables power-efficient, cost-effective, dynamic sharing of idle cloud resources and services. Federation members can sign service-level agreements (SLAs) to ensure QoS and availability.

The federation should

- have a defined marketing system that describes the cost of utilizing resources and services and that helps to valorize use,
- feature efficient geographic dispersion by allocating resources close to users to eliminate network problems that could interrupt service access, and
- follow rules in a federal-level agreement (FLA) describing the cooperation and relationship among participating clouds.

We disagree with the research literature's frequent interchangeable use of the terms ―cloud federation‖ and ―intercloud.‖ In federations, cloud organizations participate voluntarily after signing an FLA. In an intercloud organization, no private or public cloud is necessarily aware of its participation. Also, interclouds are based on open standards that provide interfaces for interoperability. Cloud federations use a broker to translate and connect CSPs‘ own interfaces.

Cloud Federation Architecture

For federations or interclouds to work properly, heterogeneous clouds must be able to interoperate. However, this can be difficult to achieve. For example, participating clouds might use different techniques to describe the services they offer. Users, however, need a mechanism to provide common access to available services. Thus, the cloud federation's architecture must employ interface standards, a service broker that translates between interfaces and provides updates on offered services and users‘ status changes, or a combination of the two.[3,8]

Cloud federations most often use brokerages. The common object request broker architecture (CORBA) and object request broker (ORB) middleware were initially the most popular approaches.[9] However, the advent of XML-based technologies such as SOAP has provided the ability to use the same language in the descriptions of all services, thereby avoiding the need for translation.

FIGURE 1. shows a cloud federation architecture with the broker playing a central role and the CSPs at the edges communicating mainly through the broker. The brokering system is in the cloud and matches the available federation resources with user demand, taking into consideration participants‘ SLAs. To achieve this, the broker must understand the various ways that each cloud describes its available resources and services[2,3] and then combine the gathered information

seamlessly for the user. In some cases, the broker could provide users with resource and service pricing information, as well as bill them. For the federation to function properly, all interested parties must sign an FLA that specifies interconnection rules and describes each participant's responsibilities and permissible behaviors, along with the financial, administrative, or other penalties for violating its terms. The parties can leave the federation when they want, as long as they follow FLA procedures.

Advantages and Limitations of Cloud Federation

Advantages

- Federation performance is guaranteed by the dynamic resource allocation—or *elasticity*—that lets clouds ask for other participants‘ idle resources or services when their own are exhausted. This achieves both uninterrupted service delivery and resource scalability, the latter being the result of the seamless, transparent operation between clouds for the delivery of an agreed-upon QoS level.
- Federations also enable the geographic dispersion of resources, efficiently locating some near users[10] but also allowing participants to access more distant resources in case of local outages. This enables efficient commercialization of the offered resources and lower prices than single- cloud services can charge.[11]
- And because the FLA clearly describes what each participant is offering, as well as the federation's rules, it ensures the commitment of the involved parties to the operation's performance.

Limitations

- Although federation mechanisms can provide the agreed-upon performance, constant monitoring and increased security mechanisms are required to guard against accidents and malicious users.
- Selecting which services a federation will offer is not trivial because they will have to come from multiple providers that have different cloud characteristics and that offer varying QoS levels.
- Thus, federation participations should deploy a service-selection mechanism, preferably automated, that uses a predefined set of criteria regarding the QoS that providers offer. Or

they could dynamically negotiate SLAs to address user needs.

- Federation members could also address the lack of a common repository for available services via peer-to-peer approaches using a distributed hash table overlay network for service discovery.[12] They could also utilize an intercloud root,[13] which produces an abstract view of a global catalog of federation services and resources offered in the connected clouds.
- The mobility of virtual machines (VMs), which are common in cloud services, is important for providing uninterrupted performance and expected QoS levels. Hosts must meet requirements for factors such as memory use, state, status of running processes and applications, and LAN connectivity to be able to migrate a live VM from one physical node to another without disrupting network traffic. This is particularly critical in real-time services. In cloud environments, this migration could be challenging for VMs belonging to different clouds that have never shared resources and thus have no knowledge about each other's networking configurations. Thus, it's important to re-create the originating cloud's networking and communication environment in the destination cloud quickly enough to avoid excessive delays.
- Federation participants must address data portability, focusing particularly on issues such as security and privacy, because services belonging to one CSP must frequently access data stored in another cloud.

Four Levels of Federation

Federation of Clouds As in traditional scheduling, where most systems try to achieve the best trade-off between the users‘ demands and the system policies and objectives, there are conflicting performance goals between the end users and the Cloud providers. While users focus on optimizing the performance of a single application or workflow, such as application throughput and user perceived response time, Cloud providers aim to obtain the best system throughput, use resources efficiently, or consume less energy. Efficient brokering policies will try to satisfy the user requirements and Clouds‘ global performance at the same time. Thereby, Cloud federation introduces new avenues of research into brokering policies such as those techniques based on ensuring the required QoS level (e.g., through advance reservation techniques) or those aiming at optimizing the energy efficiency. Furthermore, the layered service model proposed in this paper

enables the isolation between brokering policies in federated Clouds at different layers which can be implemented following different approaches. Existing work in Cloud brokering focuses on the federation of Clouds mainly at the IaaS layer such as those strategies based on match-making on top of Clouds [9], advanced reservations or energy efficiency . More detailed information of these strategies can be found the main objectives and parameters of brokering at different layers of Cloud federation. Some objectives, such as cost-effectiveness, are desired across all layers, though with different pricing methods. In the following subsections we discuss in more detail the possibilities and characteristics of brokering at different layers of federation, starting from SaaS layer that is the closest layer to the users.

Brokering at the SaaS layer

Brokering at the SaaS layer is mainly based on the user‘s requirements and Service Level Agreements (SLA) between different Cloud providers. Cloud provider that implements the SaaS layers should guarantee a given level of service for a set of application requirements. The application‘s requirements can be generic and/or specific. Generic requirements do not depend on the characteristics of the application and can be used for many types of applications. Some examples are: response time (or completion time), cost (cost of running the application), and level of security. Specific requirements deal with the characteristics and input parameters of the application. Taking WRF as a use case, some specific application requirements are: application version, geographic region, or resolution of the simulation. The main objectives of brokering in the SaaS federated Cloud, which are on three different dimensions: QoS, cost and functionality/availability.

However, the actual brokering policies should address more concrete objectives that would consider different goals and also both generic and specific application requirements/input parameters. Possible optimization goals include: • Using only generic application requirements: lowest price for a given completion time, shortest completion time for a given budget, highest security level for a given budget. • Using both generic and specific applications requirements (WRF): shortest completion time for a given simulation resolution, higher simulation resolution for a given budget and completion time

In order to achieve the objectives listed above, federated Clouds will have to handle and exchange information at the SaaS layer such as: estimated application completion time, cost of running the application, cost of software licenses, available software/versions or limitations (e.g., for the use case of WRF, the maximum simulation resolution). Based on information and the objectives described above, different strategies can be considered. Some examples are: • Forwarding: if the originator Cloud cannot accommodate the request or another Cloud can provide better costeffectiveness, the request can be forwarded to another Cloud domain of the federated Cloud. Benchmarking or modeling the applications on the Clouds‘ resources may be used to estimate the cost/completion time for a given application, but this is a transparent process at the federation level (each Cloud can have its own mechanisms). • Negotiation: one Cloud may take care of jobs from another Cloud upon agreement. The negotiation can be based on information from both past and future events. For example, a job request might be forwarded to a Cloud at higher cost but doing so may significantly optimize the energy efficiency (e.g., switching down servers and/or CRAC units). Other considerations could be taken into account during the negotiation such as the Cloud reputation (e.g., based on SLA violation rate).

Brokering at the PaaS layer

Brokering at the PaaS layer is mainly based on the application‘s requirements in terms of deployment (e.g., compiler framework) and runtime support (e.g., libraries). Since compiling tools, libraries and runtime environments can be from different vendors and with different characteristics, they can have different licensing conditions, prices and even different functionality and performance. Furthermore, additional characteristics such as fault tolerance or platform security issues can be considered in brokering policies at the PaaS layer. Given the use case of WRF the parameters are based on MPI, such as the MPI compiler characteristics, runtime environment for MPI applications and their associated costs and limitations (e.g., licenses for specific MPI runtime). The main goals of brokering a federated Cloud at the PaaS layer are focused on improving the applications‘ environments, including:

- Functionality/availability: brokering over multiple Clouds increases the probability of provisioning with more specialized compilers or execution environments.
- Optimize applications: in some sense, the objective is maximizing the potential of the applications to obtain better performance. Policies can decide using specific compilers or runtime

in order to obtain, for example, more efficient binaries for a given Cloud.

- Fault tolerance and security: when choosing a specific execution environment from different Clouds, fault tolerance and security are attractive secondary goals that may add value to a given decision or they can be primary goals if the nature of the application requires of them. In order to meet the objectives such as those described above, different Clouds must handle and exchange information related to the compiling frameworks such as vendor, capabilities, versions, compatibility or licensing costs, and information related to the runtime characteristics and limitations such as MPI implementation version, vendor, specific libraries or number of MPI processes supported. Brokering policies at the PaaS layer will try to find the best trade off between the optimization goals discussed above and the limitations from the other layers such as the cost. As a matter of example, a brokering policy may decide to compile the WRF application using an expensive compiling framework if the possible optimizations may result in lower completion time in the associated execution framework. Also, the decision can be using a higher number of MPI processes in order to maintain the QoS delivered to the users.

Brokering at the IaaS layer

When addressing federation at the IaaS layer, we consider Cloud infrastructures to be hybrid, integrating different types of resource classes such as public and private Clouds from distributed locations. As the infrastructure is dynamic and can contain a wide array of resource classes with different characteristics and capabilities, it is important to be able to dynamically provision the appropriate mix of resources based on the objectives and requirements of the application. Furthermore, application requirements and resource state may change, for example, due to workload surges, system failures or emergency system maintenance, and as a result, it is necessary to adapt the provisioning to match these changes in resource and application workload.

Brokering functions in federated Clouds at the IaaS layer can be decomposed into two aspects: resource provisioning and resource adaptation. In resource provisioning, the most appropriate mix of resource classes and the number of nodes of each resource class are estimated so as to match the requirements of the application and to ensure that the user objectives (e.g., throughput) and constraints (e.g., precision) are satisfied. Note that re- provisioning can be expensive in terms of time and other costs, and as a result, identifying the best possible initial provisioning is important.

For example, if the initial estimate of required resources is not sufficient, additional nodes can be launched. However, this would involve additional delays due to, for example, time spent to create and configure new instances. At runtime, delays can be caused by, for example, failures, premature job termination, performance fluctuation, performance degradation due to increasing user requests, etc.

As a result, it is necessary to continuously monitor the application execution and adapt resources to ensure that user objectives and constraints are satisfied. Resource adaption is, therefore, responsible for provisioning resources dynamically and on runtime. Examples are assigning more physical CPUs to a given VM to speed up an application, or migrating VMs in order to reduce the resource sharing or optimize the energy efficiency. The goals of brokering methods and policies in federated Clouds at the IaaS layer can be found in different domains. Some examples are listed as follows:

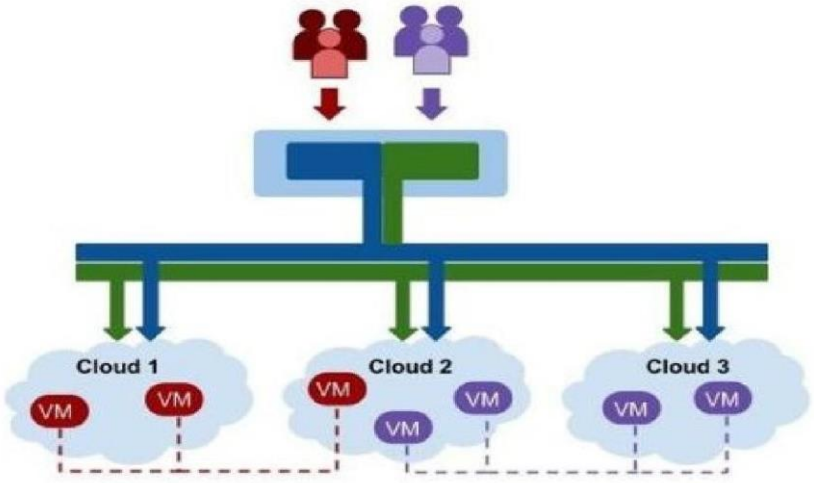
- **Cost-effectiveness**: federated Clouds provide a larger amount of resources, which may help improve cost-effectiveness. This include improvement for both the user and the provider such as, for a given cost, reducing the time to completion, increasing the system throughput or optimizing the resource utilization.
- **Acceleration:** federated Clouds can be used as accelerators to reduce the application time to completion by, for example, using Cloud resources to exploit an additional level of parallelism by offloading appropriate tasks to Cloud resources, given budget constraints.
- **Conservation:** federated Clouds can be used to conserve allocations, within the appropriate runtime and budget constraints.
- **Resilience:** federated Clouds can be used to handle unexpected situations such as an unanticipated downtime, inadequate allocations or failures of working nodes. Additional Cloud resources can be requested to alleviate the impact of the unexpected situations and meet user objectives.
- **Energy efficiency:** federated Clouds can facilitate optimizing the energy efficiency of Clouds by, for example, workload consolidation, thermal-aware placement or delegating part of the workload to external Clouds in order to optimize the energy-efficiency of a given Cloud.

Future of Federation

The next big evolution for the internet is Cloud Computing, where everyone from individuals to major corporations and governments move their data storage and processing into remote data centres. Although Cloud Computing has grown, developed and evolved very rapidly over the last half decade, Cloud Federation continues being an open issue in current cloud market.

Cloud Federation would address many existing limitations in cloud computing:

- Cloud end-users are often tied to a unique cloud provider, because of the different APIs, image formats, and access methods exposed by different providers that make very difficult for an average user to move its applications from one cloud to another, so leading to a vendor lock-in problem.
- Many SMEs have their own on-premise private cloud infrastructures to support the internal computing necessities and workloads. These infrastructures are often over-sized to satisfy peak demand periods, and avoid performance slow-down. Hybrid cloud (or cloud bursting) model is a solution to reduce the on-premise infrastructure size, so that it can be dimensioned for an average load, and it is complemented with external resources from a public cloud provider to satisfy peak demands.
- Many big companies (e.g. banks, hosting companies, etc.) and also many large institutions maintain several distributed data-centers or server-farms, for example to serve to multiple geographically distributed offices, to implement HA, or to guarantee server proximity to the end user. Resources and networks in these distributed data-centers are usually configured as non- cooperative separate elements, so that usually every single service or workload is deployed in a unique site or replicated in multiple sites.
- Many educational and research centers often deploy their own computing infrastructures, that usually do not cooperate with other institutions, except in same punctual situations (e.g. in joint projects or initiatives). Many times, even different departments within the same institution maintain their own non-cooperative infrastructures.



This Study Group will evaluate the main challenges to enable the provision of federated cloud infrastructures, with special emphasis on inter-cloud networking and security issues:

-Security and Privacy

-Interoperability and Portability

-Performance and Networking Cost

It is important to bring perspectives from Europe and USA in order to define the basis for an open cloud market, addressing barriers to adoption and meeting regulatory, legal, geographic, trust and performance constraints.

This group will directly contribute to the first two key actions of the European Cloud Strategy ‖Unleashing the Potential of Cloud Computing in Europe‖.

-The first key action aims at –Cutting through the Jungle of Standards‖ to help the adoption of cloud computing by encouraging compliance of cloud services with respect to standards and thus providing evidence of compliance to legal and audit obligations. These standards aim to avoid customer lock in by promoting interoperability, data portability and reversibility.

-The second key action –Safe and Fair Contract Terms and Conditions‖ aims to protect the cloud consumer from insufficiently specific and balanced contracts with cloud providers that do not

–provide for liability for data integrity, confidentiality or service continuity‖. The cloud consumer is often presented with "take-it-or-leave-it standard contracts that are of the cost saving for the provider but is often undesirable for the user‖. The commission aims to develop with

–stakeholders model terms for cloud computing service level agreements for contracts‖.