Cloud Computing

UNIT III

P KIRAN RAO, Asst Prof, Dept of CSE

Introduction

Cloud computing model is making IT resource available. It also provides many features like less maintaining cost, infrastructure cost, dynamic resource sharing and etc; The major players in the IT industry such as Google, Amazon, Microsoft, Sun, and Yahoo have started offering cloud computing based solutions that cover the entire IT computing stack, from hardware to applications and services.

These industries established public cloud which represents a publicly accessible distributed system hosting the execution of applications and providing services billed on a pay-per-use basis. This type of cloud computing deployment model faces some limitations.

For example

- The public cloud distributed anywhere on the planet, legal issues arise and they simply make it difficult to rely on a virtual public infrastructure for any IT operation.
- Data location and Confidentiality
- Service levels are limited in public cloud for a specific distributed application.

The above limitations can be overcome with the other deployment model i.e Private cloud computing.

The features of private cloud computing:-

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 in their system. Security in-house is easier to maintain and to rely on.

Infrastructure Ensuring Service Level Agreements (SLAs) Quality of service implies that specific operations such as appropriate clustering and failover, data replication, system monitoring and maintenance, disaster recovery, and other uptime services can be commensurate to the application needs. While public clouds 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 virtual public infrastructure.

Private Cloud Limitations

Private clouds cannot easily scale out in the case of peak demand

To overcome above all limitations which are in private, public cloud deployment models these two deployment models are integrated to reach the increased load. Hence, hybrid clouds, which are the result of a private cloud growing and provisioning resources from a public cloud, are likely to be best option for the future in many cases. Hybrid clouds allow exploiting existing IT infrastructures, maintaining sensitive information within the premises, and naturally growing and shrinking by provisioning external resources and releasing them when needed. Securities concerns are then only limited to the public portion of the cloud, which can be used to perform operations with less stringent constraints but that are still part the system workload.

Platform as a Service (PaaS) solutions offer the right tools to implement and deploy hybrid clouds. They provide enterprises with a platform for creating, deploying, and managing distributed applications on top of existing infrastructures. They are in charge of monitoring and managing the infrastructure and acquiring new nodes, and they rely on virtualization technologies in order to scale applications on demand.

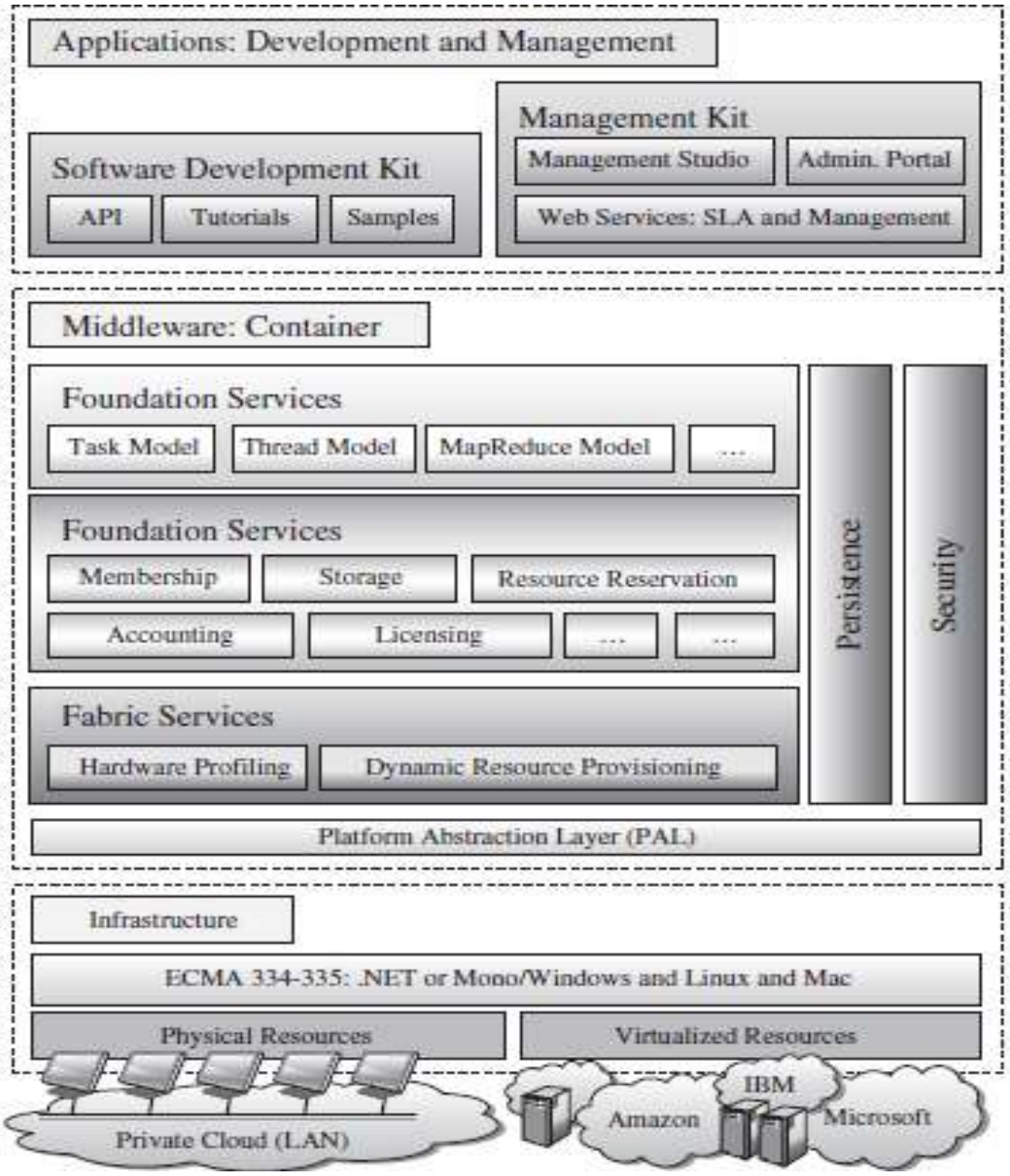
ANEKA CLOUD PLATFORM

- Aneka is a software platform and a framework for developing distributed applications on the cloud.
- Aneka provides developers with a rich set of APIs for transparently exploiting these resources by expressing the application logic with a variety of programming abstractions.
- Aneka can be a public cloud available to anyone through the Internet, a private cloud constituted by a set of nodes with restricted access within an enterprise, or a hybrid cloud where external resources are integrated on demand, thus allowing applications to scale.
- Aneka is essentially an implementation of the PaaS model, and it provides a runtime environment for executing applications by using cloud infrastructure.

Aneka Framework:

- Aneka framework is a layered structured framework.

Application Layer provides Developers to express distributed applications by using the API contained in the Software Development Kit (SDK) or by porting existing legacy applications to the cloud.



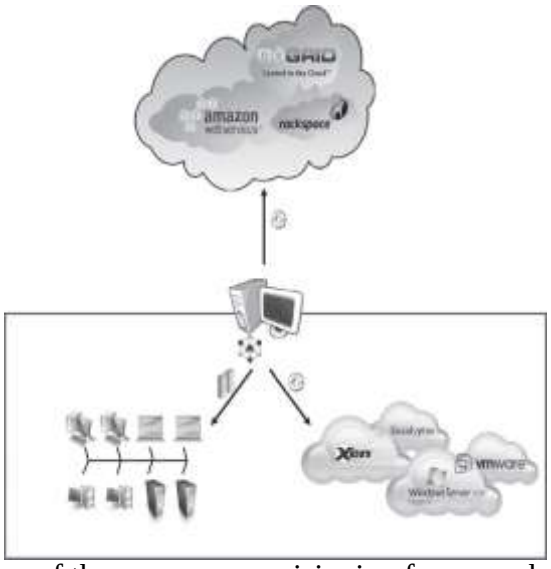
Execution Services: They are responsible for scheduling and executing applications. Each of the programming models supported by Aneka defines specialized implementations of these services for managing the execution of a unit of work defined in the model.

Foundation Services: These are the core management services of the Aneka container. They are in charge of metering applications, allocating resources for execution, managing the collection of available nodes, and keeping the services registry updated.

Fabric Services: They constitute the lowest level of the services stack of Aneka and provide access to the resources managed by the cloud. An important service in this layer is the Resource Provisioning Service, which enables horizontal scaling3 in the cloud. **Resource provisioning** makes Aneka elastic and allows it to grow or to shrink dynamically to meet the QoS requirements of applications.

Aneka also provides a tool for managing the cloud, allowing administrators to easily start, stop, and deploy instances of the Aneka container on new resources and then reconfigure them dynamically to alter the behavior of the cloud.

Aneka resource provisioning



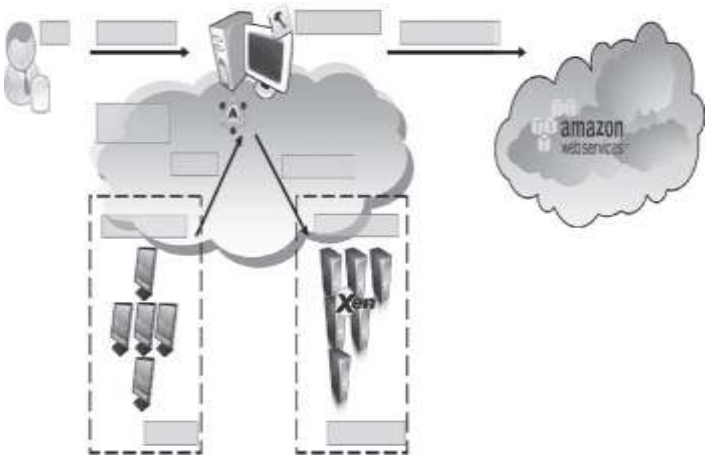
The benefit of cloud computing is the elasticity of resources, services, and applications, which is the ability to automatically scale out based on demand and users' quality of service requests. Aneka as a PaaS not only features multiple programming models allowing developers to easily build their distributed applications, but also provides resource provisioning facilities in a seamless and dynamic fashion. Applications managed by the Aneka container can be dynamically mapped to heterogeneous resources, which can grow or shrink according to the application's needs. This elasticity is achieved by means of the resource provisioning framework, which is composed primarily of services built into the Aneka fabric layer.

Private resources identify computing and storage elements kept in the premises that share similar internal security and administrative policies. Aneka identifies two types of private resources: static and dynamic resources. Static resources are constituted by existing physical workstations and servers that may be idle for a certain period of time. Their membership to the Aneka cloud is manually configured by administrators and does not change over time. Dynamic resources are mostly represented by virtual instances that join and leave the Aneka cloud and are controlled by resource pool managers that provision and release them when needed. Public resources reside outside the boundaries of the enterprise and are provisioned by establishing a

service-level agreement with the external provider. Even in this case we can identify two classes: on-demand and reserved resources. On-demand resources are dynamically provisioned by resource pools for a fixed amount of time (for example, an hour) with no long-term commitments and on a pay-as-you-go basis. Reserved resources are provisioned in advance by paying a low, one-time fee and mostly suited for long-term usage. These resources are actually the same as static resources, and no automation is needed in the resource provisioning service to manage them.

Despite the specific classification previously introduced, resources are managed uniformly once they have joined the Aneka cloud and all the standard operations that are performed on statically configured nodes can be transparently applied to dynamic virtual instances.

Resource Provisioning Scenario



Aneka's Resource Provisioning service is required to acquire these resources from both the private data center managed by Xen Hypervisor and the Amazon public cloud.

A private enterprise maintains a private cloud, which consists of (a) five physical dedicated desktops from its engineering department and (b) a small data center managed by Xen Hypervisor providing virtual machines with the maximum capacity of 12 VMs. In most of the cases, this setting is able to address the computing needs of the enterprise. In the case of peak computing demand, additional resources can be provisioned by leveraging the virtual public infrastructure. For example, a mission critical application could require at least 30 resources to complete within an hour, and the customer is willing to spend a maximum of 5 dollars to achieve this goal. In this case, the Aneka Resource Provisioning service becomes a fundamental infrastructure component to address this scenario.

As shown in above case, once the client has submitted the application, the Aneka scheduling engine detects that the current capacity in terms of resources (5 dedicated nodes) is not enough to satisfy the user's QoS requirement and to complete the application on time. An additional 25 resources must be provisioned. It is the responsibility of the Aneka Resource Provisioning service to acquire these resources from both the private data center managed by Xen Hypervisor and the Amazon public cloud. The provisioning service is configured by default with a cost-effective strategy, which privileges the use of local resources instead of the dynamically provisioned and chargeable ones. The computing needs of the application require

the full utilization of the local data center that provides the Aneka cloud with 12 virtual machines. Such capacity is still not enough to complete the mission critical application in time; and the remaining 13 resources are rented from Amazon for a minimum of one hour, which incurs a few dollars' cost.

Aneka fallows different scenarios for provisioning resources for example instead of provisioning 13 small instances from Amazon, a major number of resources, or more powerful resources, can be rented by spending the entire budget available for the application.

HYBRID CLOUD IMPLEMENTATION

Each cloud computing service provider exposes its own interfaces and protocols. Hence, it is not possible to seamlessly integrate different providers into one single infrastructure. The resource provisioning service implemented in Aneka addresses these issues and abstracts away the differences of providers' implementation.

Support for Heterogeneity. Hybrid clouds are produced by heterogeneous resources such as clusters, public or private virtual infrastructures, and workstations. In particular, for what concerns a virtual machine manager, it must be possible to integrate additional cloud service providers without major changes to the entire system design and codebase. Hence, the specific code related to a particular cloud resource provider should be kept isolated behind interfaces and within pluggable components.

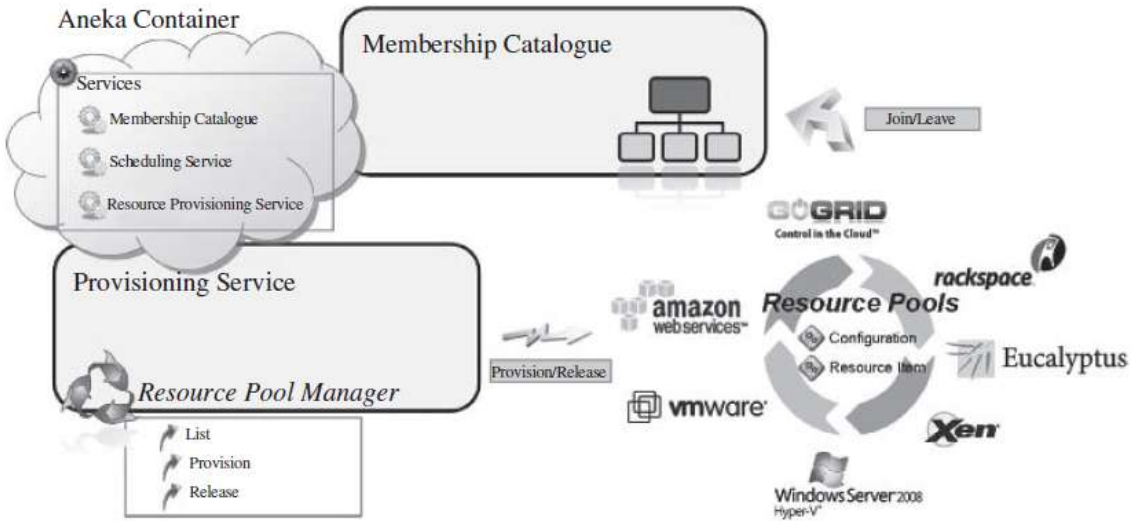
Support for Dynamic and Open Systems. Hybrid clouds change their composition and topology over time. They form as a result of dynamic conditions such as peak demands or specific Service Level Agreements attached to the applications currently in execution. An open and extensible architecture that allows easily plugging new components and rapidly integrating new features is of a great value in this case. Specific enterprise architectural patterns can be considered while designing such software systems. In particular, inversion of control and, more Precisely, dependency injection5 in component-based systems is really helpful.

Support for Basic VM Operation Management. Hybrid clouds integrate virtual infrastructures with existing physical systems. Virtual infrastructures are produced by virtual instances. Hence, software frameworks that support hypervisor-based execution should implement a minimum set of operations. They include requesting a virtual instance, controlling its status, terminating its execution, and keeping track of all the instances that have been requested.

Support for Flexible Scheduling Policies. The heterogeneity of resources that constitute a hybrid infrastructure naturally demands for flexible scheduling policies. Public and private resources can be differently utilized, and the workload should be dynamically partitioned into different streams according to their security and quality of service (QoS) requirements. There is then the need of being able to transparently change scheduling policies over time with a minimum impact on the existing infrastructure and almost now downtimes. Configurable scheduling policies are then an important feature.

Support for Workload Monitoring. Workload monitoring becomes even more important in the case of hybrid clouds where a subset of resources is leased and resources can be dismissed if they are no longer necessary. Workload monitoring is an important feature for any distributed middleware, in the case of hybrid clouds, it is necessary to integrate this feature with scheduling policies that either directly or indirectly governs the management of virtual instances and their leases.

Aneka Hybrid Cloud Architecture



The resource provisioning infrastructure is represented by a collection of resource pools that provide access to resource providers, whether they are external or internal, and managed uniformly through a specific component called a resource pool manager.

Resource Provisioning Service:- This is an Aneka-specific service that implements the service interface and wraps the resource pool manager, thus allowing its integration within the Aneka container.

Resource Pool Manager This manages all the registered resource pools and decides how to allocate resources from those pools. The resource pool manager provides a uniform interface for requesting additional resources from any private or public provider and hides the complexity of managing multiple pools to the Resource Provisioning Service.

Resource Pool This is a container of virtual resources that mostly come from the same resource provider. A resource pool is in charge of managing the virtual resources it contains and eventually releasing them when they are no longer in use. Since each vendor exposes its own

specific interfaces, the resource pool (a) encapsulates the specific implementation of the communication protocol required to interact with it and (b) provides the pool manager with a unified interface for acquiring, terminating, and monitoring virtual resources. The request for additional resources is generally triggered by a scheduler that detects that the current capacity is not sufficient to satisfy the expected quality of services ensured for specific applications. In this case a provisioning request is made to the Resource Provisioning Service. According to specific policies, the pool manager determines the pool instance(s) that will be used to provision resources and will forward the request to the selected pools. Each resource pool will translate the forwarded request by using the specific protocols required by the external provider and provision the resources. Once the requests are successfully processed, the requested number of virtual resources will join the Aneka cloud by registering themselves with the Membership Catalogue Service, which keeps track of all the nodes currently connected to the cloud. Once joined the cloud the provisioned resources are managed like any other node.

A release request is triggered by the scheduling service when provisioned resources are no longer in use. Such a request is then forwarded to the interested resources pool (with a process similar to the one described in the previous paragraph) that will take care of terminating the resources when more appropriate. A general guideline for pool implementation is to keep provisioned resources active in a local pool until their lease time expires. By doing this, if a new request arrives within this interval, it can be served without leasing additional resources from the public infrastructure. Once a virtual instance is terminated, the Membership Catalogue Service will detect a disconnection of the corresponding node and update its registry accordingly.

The current implementation of Aneka allows customizing the Resource Provisioning Infrastructure by specifying the following elements:

Resource Provisioning Service. The default implementation provides a lightweight component that generally forwards the requests to the resource Pool Manager. A possible extension of the system can be the implementation of a distributed resource provisioning service that can operate at this level or at the Resource Pool Manager level.

Resource Pool Manager. The default implementation provides the basic management features required for resource and provisioning request forwarding.

Resource Pools. The Resource Pool Manager exposes a collection of resource pools that can be used. It is possible to add any implementation that is compliant to the interface contract exposed by the Aneka provisioning API, thus adding a heterogeneous open-ended set of external providers to the cloud.

Provisioning Policy. Scheduling services can be customized with resource provisioning aware algorithms that can perform scheduling of applications by taking into account the required QoS.

Use Case—the Amazon EC2 Resource Pool

Amazon EC2 is one of the most popular cloud resource providers.

It provides a Web service interface for accessing, managing, and controlling virtual machine instances. The Web-service-based interface simplifies the integration of Amazon EC2 with any application. This is the case of Aneka, for which a simple Web service client has been developed to allow the interaction with EC2.

In order to interact with Amazon EC2, several parameters are required:

User Identity This represents the account information used to authenticate withAmazonEC2.The identity is constituted by a pair of encrypted keys that are the access key and the secret key. These keys can be obtained from the Amazon Web services portal once the user has signed in, and they are required to perform any operation that involves Web service access.

Resource Identity. The resource identity is the identifier of a public or a private Amazon Machine Image (AMI) that is used as template from which to create virtual machine instances.

Resource Capacity. This specifies the different type of instance that will be deployed byAmazonEC2. Instance types vary according to the number of cores, the amount of memory, and other settings that affect the performance of the virtual machine instance. Several types of images are available, those commonly used are: small, medium, and large. The capacity of each type of resource has been predefined by Amazon and is charged differently.

This information is maintained in the EC2ResourcePoolConfiguration class and need to be provided by the administrator in order to configure the pool. Hence, the implementation of EC2ResourcePool is forwarding the request of the pool manager to EC2 by using the Web service client and the configuration information previously described. It then stores the metadata of each active virtual instance for further use. In order to utilize at best the virtual machine instances provisioned from EC2, the pool implements a cost-effective optimization strategy.

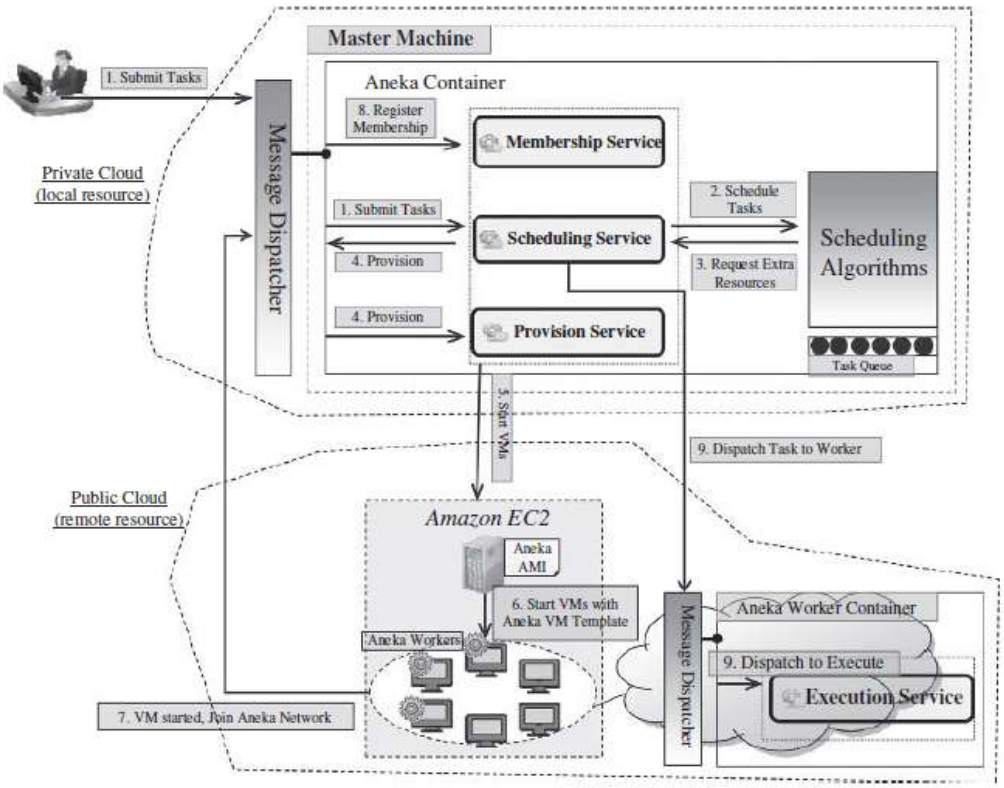
Implementation Steps for Aneka Resource Provisioning Service

The general steps of resource provisioning on demand in Aneka are the following:

The application submits its tasks to the scheduling service, which, in turns, adds the tasks into the scheduling queue.

The scheduling algorithm finds an appropriate match between a task and a resource. If the algorithm could not find enough resources for serving all the tasks, it requests extra resources from the scheduling service.

The scheduling service will send a Resource Provision Message to provision service and will ask provision service to get X number of resources as determined by the scheduling algorithm.



Upon receiving the provision message, the provision service will delegate the provision request to a component called resource pool manager, which is responsible for managing various resource pools. A resource pool is a logical view of a cloud resource provider, where the virtual machines can be provisioned at runtime. Aneka resource provisioning supports multiple resource pools such as Amazon EC2 pool and Citrix Xen server pool.

The resource pool manager knows how to communicate with each pool and will provision the requested resources on demand. Based on the requests from the provision service, the pool manager starts X virtual machines by utilizing the predefined virtual machine template already configured to run Aneka containers.

A worker instance of Aneka will be configured and running once a virtual resource is started. All the work instances will then connect to the Aneka master machine and will register themselves with Aneka membership service.

The scheduling algorithm will be notified by the membership service once those work instances join the network, and it will start allocating pending tasks to them immediately.

Once the application is completed, all the provisioned resources will be released by the provision service to reduce the cost of renting the virtual machine.

Comet Cloud

Introduction

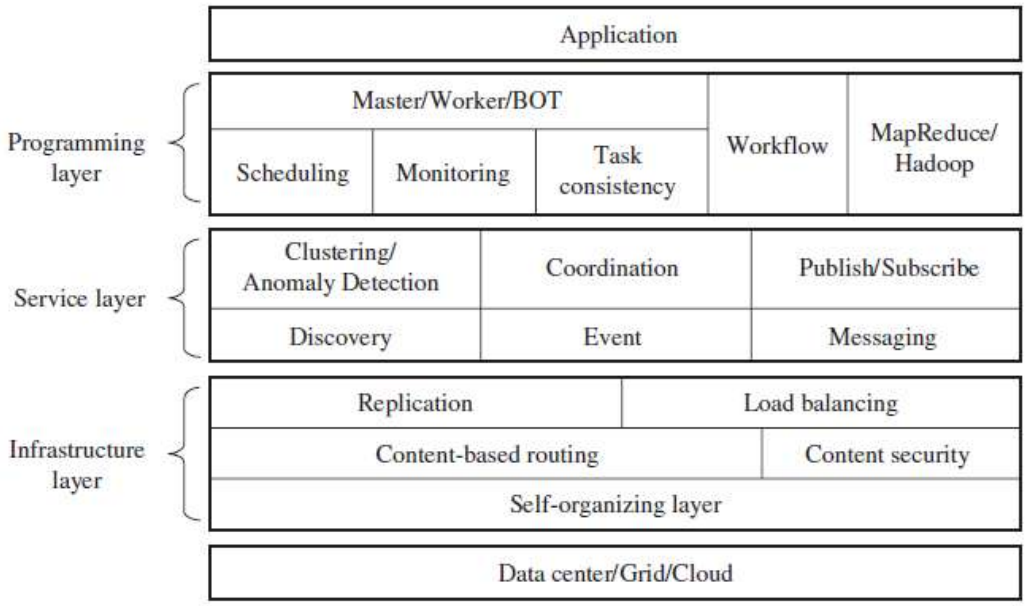
Clouds typically have highly dynamic demands for resources with highly heterogeneous and dynamic workloads. For example, the workloads associated with the application can be quite dynamic, in terms of both the number of tasks processed and the computation requirements of each task. Comet Cloud is to realize a virtual computational cloud with resizable computing capability, which integrates local computational environments and public cloud services on-demand, and provide abstractions and mechanisms to support a range of programming paradigms and applications requirements.

Comet Cloud enables policy-based autonomic cloud bridging and cloud bursting. Autonomic cloud bridging enables on-the-fly integration of local computational environments and public cloud services (such as Amazon EC2 and Eucalyptus, and autonomic cloud bursting enables dynamic application scale-out to address dynamic workloads, spikes in demands, and other extreme requirements.

Comet Cloud is based on a decentralized coordination substrate, and it supports highly heterogeneous and dynamic cloud/grid infrastructures, integration of public/private clouds, and cloudbursts.

COMETCLOUD ARCHITECTURE

Comet Cloud is composed of a programming layer, a service layer, and an infrastructure layer.



The **infrastructure layer** uses the group of self-organizing overlay and the Squid information discovery and content-based routing substrate. The routing engine supports flexible **content-based routing** and complex querying using partial keywords, wildcards, or ranges. It also guarantees that all peer nodes with data elements that match a query/ message will be located. Nodes providing resources in the overlay have different roles and, accordingly, different access privileges based on their credentials and capabilities. This layer also provides replication and load balancing services, and it handles dynamic joins and leaves of nodes as well as node failures. Every node keeps the replica of its successor node's state, and it reflects changes to this replica whenever its successor notifies it of changes. It also notifies its predecessor of any changes to its state. If a node fails, the predecessor node merges the replica into its state and then makes a replica of its new successor. If a new node joins, the joining node's predecessor updates its replica to reflect the joining node's state, and the successor gives its state information to the joining node. To maintain load balancing, load should be redistributed among the nodes whenever a node joins and leaves.

The **service layer** provides a range of services to supports autonomics at the programming and application level. This layer supports the Linda-like tuple space coordination model, and it provides a virtual shared-space abstraction as well as associative access primitives.
The basic coordination primitives are listed below:

out (ts, t): a non blocking operation that inserts tuple t into space ts.
in (ts, t'): a blocking operation that removes a tuple t matching template t' from the space ts and returns it.

rd (ts, t'): a blocking operation that returns a tuple t matching template t' from the space ts. The tuple is not removed from the space.
The out is for inserting a tuple into the space, and in and rd are for reading a tuple from the space are implemented in removes the tuple after read, and rd only reads the tuple.

The above uniform operators do not distinguish between local and remote spaces, and consequently the Comet is naturally suitable for context-transparent applications. However, this abstraction does not maintain geographic locality between peer nodes and may have a detrimental effect on the efficiency of the applications imposing context-awareness for example mobile applications. These applications require that context locality be maintained in addition to content locality; that is, they impose requirements for context-awareness. To address this issue, Comet Cloud supports dynamically constructed transient spaces that have a specific scope definition. The global space is accessible to all peer nodes and acts as the default coordination platform. Membership and authentication mechanisms are adopted to restrict access to the transient spaces. This layer also provides asynchronous messaging and evening services. Finally, on-line clustering services support autonomic management and enable self-monitoring and control. Events describing the status or behavior of system components are clustered, and the clustering is used to detect anomalous behaviors.

The **programming layer** provides the basic framework for application development and management. It supports a range of paradigms including the master/worker/BOT. Masters generate tasks and workers consume them. Masters and workers can communicate via virtual shared space or using a direct connection. Scheduling and monitoring of tasks are supported by the application framework. The task consistency service handles lost tasks. Even though replication is provided by the infrastructure layer, a task may be lost due to network congestion. In this case, since there is no failure, infrastructure level replication may not be able to handle it. This can be handled by the master, for example, by waiting for the result of each task for a predefined time interval and, if it does not receive the result back, regenerating the lost task. If the master receives duplicate results for a task, it selects the first one and ignores other subsequent results.

Comet Space

In Comet, a tuple is a simple XML string, where the first element is the tuple's tag and is followed by an ordered list of elements containing the tuple's fields. Each field has a name followed by its value. The tag, field names, and values must be actual data for a tuple and can contain wildcards ("*") for a template tuple.

A tuple in Comet can be retrieved if it exactly or approximately matches a template tuple. Exact matching requires the tag and field names of the template tuple to be specified without any wildcard, as in Linda.

For example

```
<contact>
   <name> Smith </name>
   <phone> 7324451000 </phone>
   <email> smith@gmail.com </email>
   <dep> ece </dep>
</contact>
```

(a)

```
<contact>
   <name> Smith </name>
   <phone> 7324451000 </phone>
   <email> * </email>
   <dep> * </dep>
</contact>
```

(b)

```
<contact>
   <na*> Smith </na*>
   <*>
   <*>
   <dep> ece </dep>
</contact>
```

(c)

AUTONOMIC BEHAVIOR OF COMETCLOUD

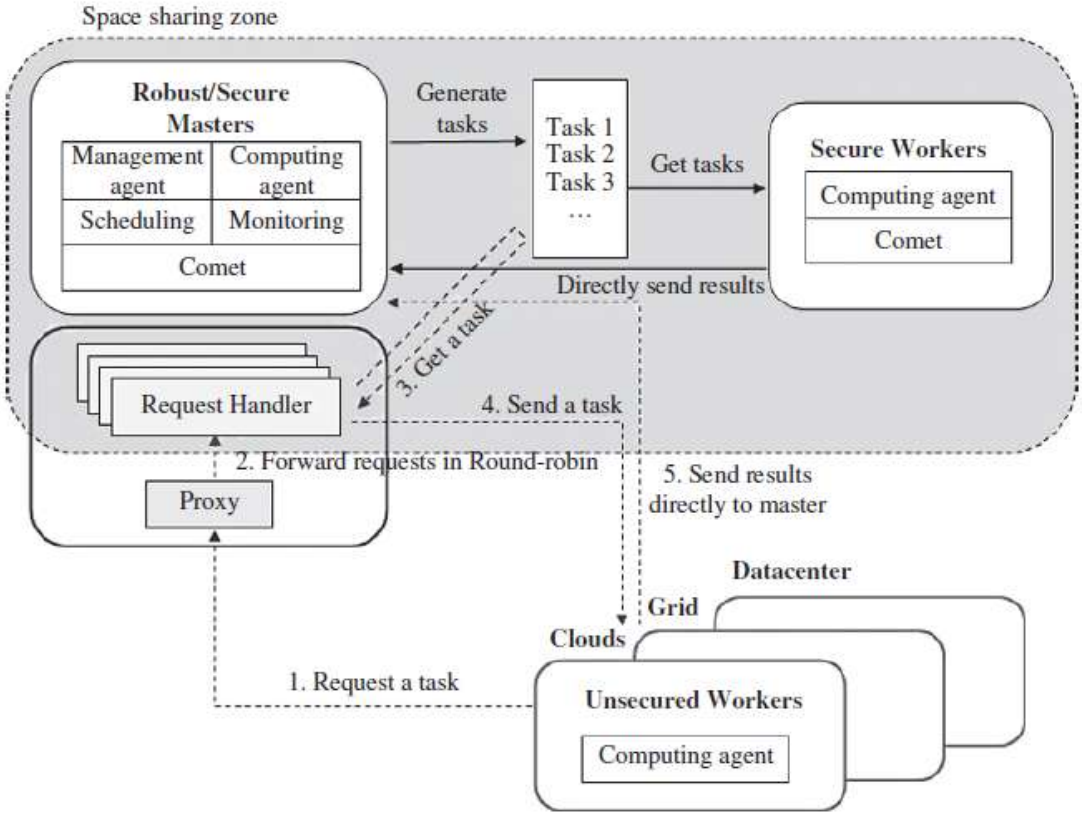
Autonomic Cloudbursting

Autonomic cloudbursts in CometCloud is presented in the following figure. CometCloud considers three types of clouds based on perceived security/trust and assigns capabilities accordingly.

The first is a highly trusted, robust, and secure cloud, usually composed of trusted/secure nodes within an enterprise, which is typically used to host masters and other key (management, scheduling, monitoring) roles. These nodes are also used to store states.

The second type of cloud is one composed of nodes with such credentials—that is, the cloud of secure workers.

The final type of cloud consists of casual workers.



Casual workers are not part of the space but can access the space through the proxy and a request handler to obtain work units as long as they present required credentials. Nodes can be added or deleted from any of these clouds by purpose. If the space needs to be scale-up to store dynamically growing workload as well as requires more computing capability, then autonomic cloudbursts target secure worker to scale up. But only if more computing capability is required, then unsecured workers are added.

Key motivations for autonomic cloudbursts include:

Load Dynamics Application workloads can vary significantly. This includes the number of application tasks as well the computational requirements of a task. The computational environment must dynamically grow (or shrink) in response to these dynamics while still maintaining strict deadlines.

Accuracy of the Analytics The required accuracy of risk analytics depends on a number of highly dynamic market parameters and has a direct impact on the computational demand.

Collaboration of Different Groups Different groups can run the same application with different dataset policies . Here, policy means user's SLA bounded by their condition such as time frame,

budgets, and economic models. As collaboration groups join or leave the work, the computational environment must grow or shrink to satisfy their SLA.

Economics. Application tasks can have very heterogeneous and dynamic priorities and must be assigned resources and scheduled accordingly. Budgets and economic models can be used to dynamically provision computational resources based on the priority and criticality of the application task.

Failures. The computation must be able to manage failures without impacting application quality of service, including deadlines and accuracies.

Autonomic Cloudbridging

Autonomic cloudbridging is meant to connect CometCloud and a virtual cloud which consists of public cloud, data center, and grid by the dynamic needs of the application.

The clouds in the virtual cloud are heterogeneous and have different types of resources and cost policies, besides, the performance of each cloud can change over time by the number of current users.

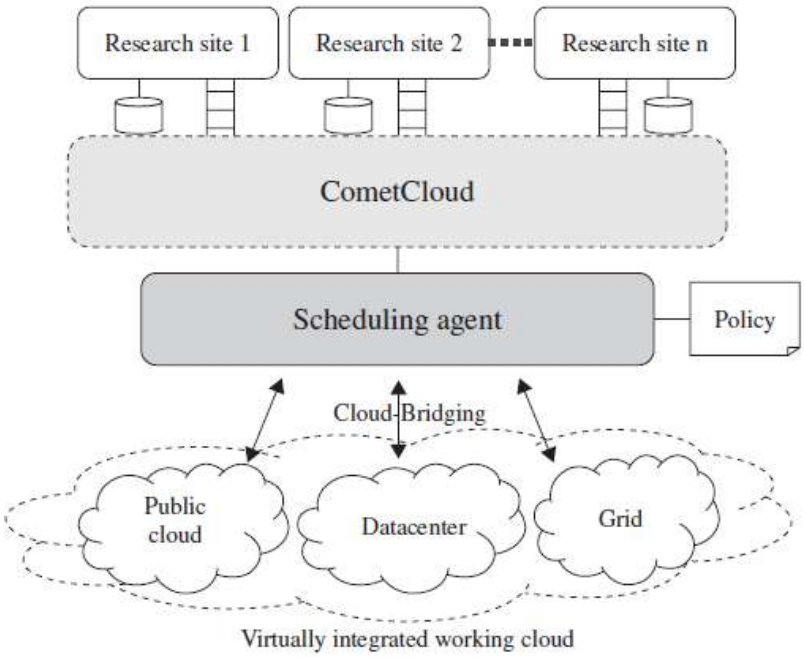


Figure shows an overview of the operation of the CometCloudbased autonomic cloudbridging. cloudbursts over the virtual cloud, and there can be one or more scheduling agents. A scheduling agent is located at a robust/secure master site. If multiple collaborating research groups work together and each group requires generating tasks with its own data and managing the virtual cloud by its own policy, then it can have a separate scheduling agent in its master site.

A scheduling agent manages autonomic cloudbridging and guarantees QoS within user policies. Autonomic cloudburst is represented by changing resource provisioning not to violate defined policy. We define three types of policies.

Deadline-Based. When an application needs to be completed as soon as possible, assuming an adequate budget, the maximum required workers are allocated for the job.

Budget-Based. When a budget is enforced on the application, the number of workers allocated must ensure that the budget is not violated.

Workload-Based. When the application workload changes, the number of workers explicitly defined by the application is allocated or release.

T-SYSTEMS

T-Systems approaches cloud computing from the viewpoint of an organization with an established portfolio of dynamic, scalable services delivered via networks. The service provider creates end-to-end offerings that integrate all elements, in collaboration with established hardware and software vendors.

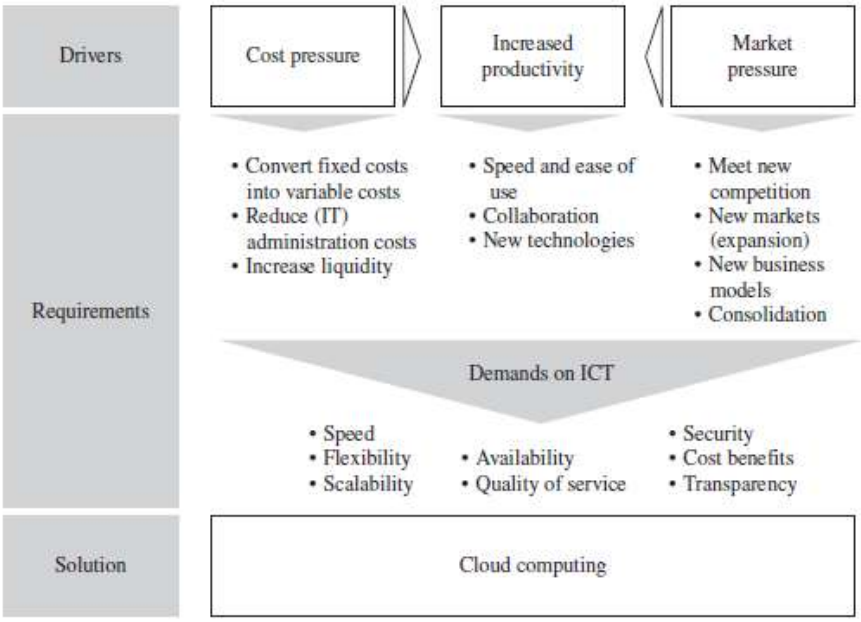
WHAT ENTERPRISES DEMAND OF CLOUD COMPUTING

Whether operated in-house or by an external provider, ICT is driven by two key factors:
cost pressure and market pressure.

Both of these call for increases in productivity.

Changing Markets

Today's markets are increasingly dynamic.



Workflow Engine:

A workflow models a process as consisting of a series of steps that simplifies the complexity of execution and management of applications. Processing and managing such large amounts of data require the use of a distributed collection of computation and storage facilities.

Architectural view of aWorkflow Management System (WfMS) utilizing cloud resources to drive the execution of a scientific as shown in below figure.

Scientific applications are typically modeled as workflows, consisting of tasks, data elements, control sequences and data dependencies. Workflow management systems are responsible for managing and executing these workflows.

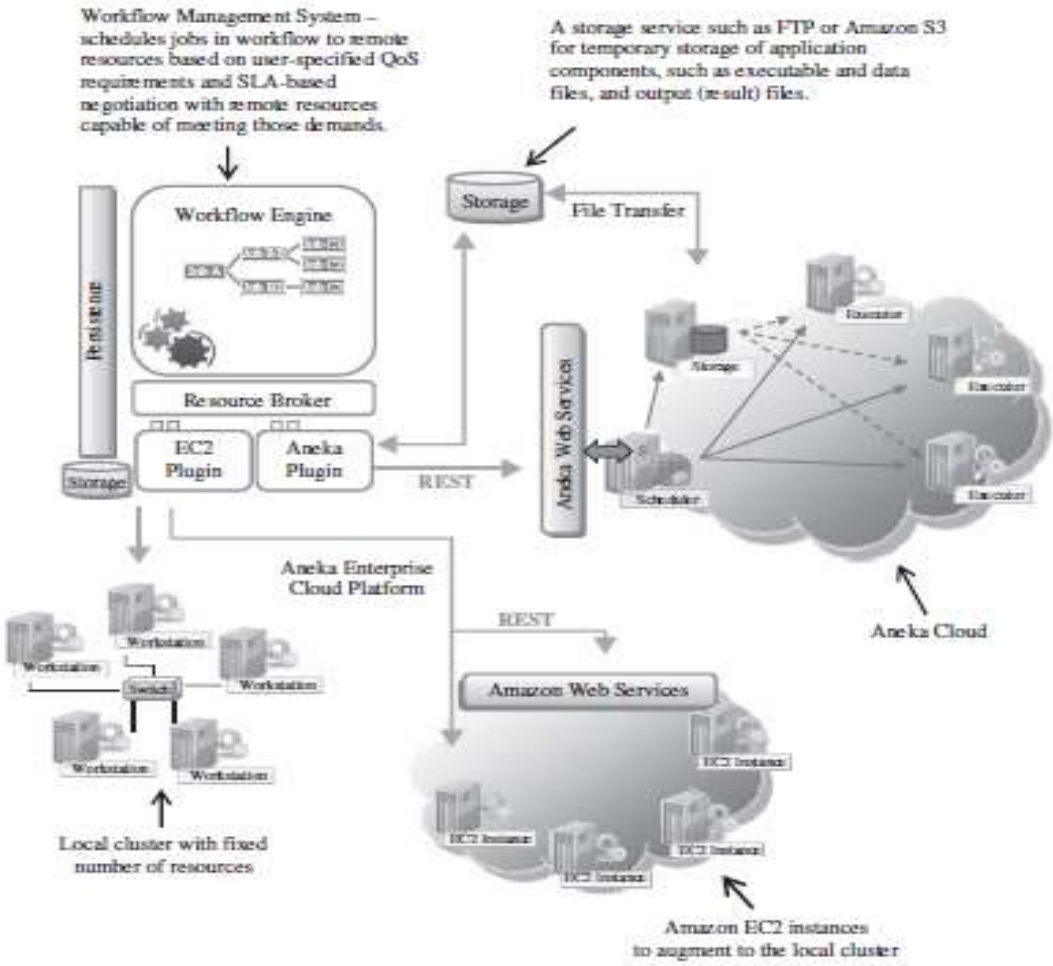
Scientific workflow management systems are engaged and applied to the following aspects of scientific computations:
(1) describing complex scientific procedures (using GUI tools, workflow specific languages),
(2) automating data derivation processes (data transfer components),
(3) high-performance computing (HPC) to improve throughput and performance (distributed resources and their coordination), and
(4) provenance management and query (persistence components).

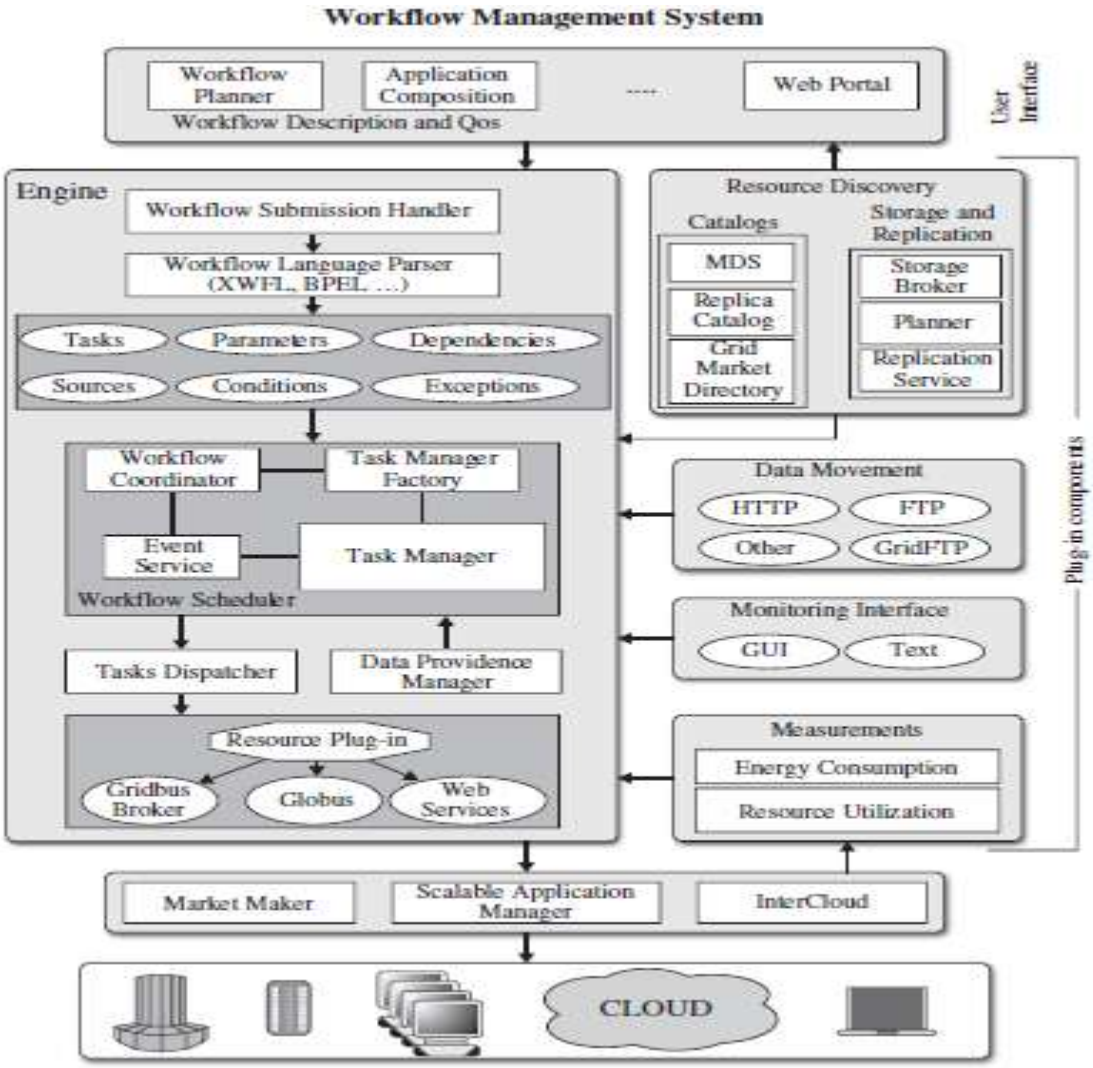
The Cloudbus Workflow Management System consists of components that are responsible for handling tasks, data and resources taking into account users' QoS requirements.

Its architecture is shown in Figure 12.2. The architecture consists of three major parts: (a) the user interface, (b) the core, and (c) plug-ins. The user interface allows end users to work with workflow composition, workflow execution planning, submission, and monitoring. These

features are delivered through a Web portal or through a stand-alone application that is installed at the user's end. Workflow composition is done using an XML-based Workflow Language (xWFL). Users define task properties and link them based on their data dependencies. Multiple tasks can be constructed using copy-paste functions present in most GUIs.

The components within the core are responsible for managing the execution of workflows. They facilitate in the translation of high-level workflow descriptions (defined at the user interface using XML) to task and data objects. These objects are then used by the execution subsystem. The scheduling component applies user-selected scheduling policies and plans to the workflows at various stages in their execution. The tasks and data dispatchers interact with the resource interface plug-ins to continuously submit and monitor tasks in the workflow. These components form the core part of the workflow engine. The plug-ins support workflow executions on different environments and platforms. Our system has plug-ins for querying task and data characteristics





The plug-ins support workflow executions on different environments and platforms. Our system has plug-ins for querying task and data characteristics. transferring data to and from resources (e.g., transfer protocol implementations, and storage and replication services), monitoring the execution status of tasks and applications (e.g., real-time monitoring GUIs, logs of execution, and the scheduled retrieval of task status), and measuring energy consumption. The resources are at the bottom layer of the architecture and include clusters, global grids, and clouds. The WfMS has plug-in components for interacting with various resource management systems present at the front end of distributed resources. Currently, the Cloudbus WfMS supports Aneka, Pbs, Globus, and fork-based middleware. The resource managers may communicate with the market maker, scalable application manager, and InterCloud services for global resource management.

MAPREDUCE PROGRAMMING MODEL

MapReduce is a software framework for solving many large-scale computing problems. The MapReduce abstraction is inspired by the Map and Reduce functions, which are commonly used in functional languages such as Lisp.

The map function, written by the user, processes a key/value pair to generate a set of intermediate key/value pairs:
map (key1, value1) - list (key2, value2)

The reduce function, also written by the user, merges all intermediate values associated with the same intermediate key:
reduce (key2, list (value2)) - list (value2)

Main Features

Data-Aware. When the MapReduce-Master node is scheduling the Map tasks for a newly submitted job, it takes in consideration the data location information retrieved from the GFS-Master node.

Simplicity. As the MapReduce runtime is responsible for parallelization and concurrency control, this allows programmers to easily design parallel and distributed applications.

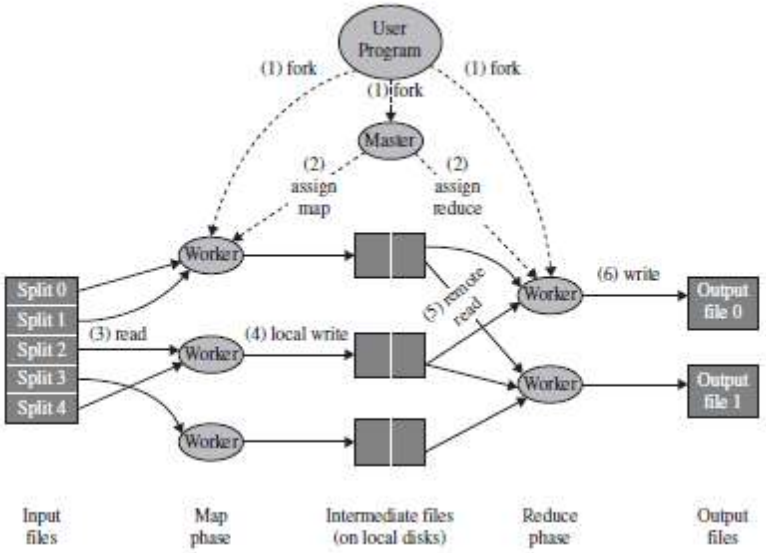
Manageability. In traditional data-intensive applications, where data are stored separately from the computation unit, we need two levels of management: (i) to manage the input data and then move these data and prepare them to be executed; (ii) to manage the output data. In contrast, in the Google MapReduce model, data and computation
are allocated, taking advantage of the GFS, and thus it is easier to manage the input and output data.

Scalability. Increasing the number of nodes (data nodes) in the system will increase the performance of the jobs with potentially only minor losses.

fault Tolerance and Reliability. The data in the GFS are distributed on clusters with thousands of nodes. Thus any nodes with hardware failures can be handled by simply removing them and installing a new node in their place. Moreover, MapReduce, taking advantage of the replication in GFS, can achieve high reliability by (1) rerunning all the tasks (completed or in progress) when a host node is going off-line, (2) rerunning failed tasks on another node, and (3) launching backup tasks when these tasks are slowing down and causing a bottleneck to the entire job.

Execution Overview

When the user program calls the MapReduce function, the following sequence of actions occurs.



The MapReduce library in the user program first splits the input files into M pieces of typically 16 to 64 megabytes (MB) per piece. It then starts many copies of the program on a cluster. One is the "master" and the rest are "workers." The master is responsible for scheduling and monitoring.

When map tasks arise, the master assigns the task to an idle worker, taking into account the data locality. A worker reads the content of the corresponding input split and emits a key/value pairs to the user-defined Map function. The intermediate key/value pairs produced by the Map function are first buffered in memory and then periodically written to a local disk, partitioned into R sets by the partitioning function.

The master passes the location of these stored pairs to the reduce worker, which reads the buffered data from the map worker using remote procedure calls (RPC). It then sorts the intermediate keys so that all occurrences of the same key are grouped together. For each key, the worker passes the corresponding intermediate value for its entire occurrence to the Reduce function.

Finally, the output is available in R output files.