**Anna University Examinations, Nov/Dec 2018**

**B.E-CSE- Final Year – VII Semester**

**CS6703- Grid and Cloud Computing**

**Solution**

**Part A**

1. **Outline any two advantages of distributed computing.**

1. Reliability, high fault tolerance

A system crash on one server does not affect other servers.

2. Scalability

In distributed computing systems you can add more machines as needed.

3. Flexibility

It makes it easy to install, implement and debug new services.

4. Fast calculation speed

A distributed computer system can have the computing power of multiple computers, making it faster than other systems.

5. Openness

Since it is an open system, it can be accessed both locally and remotely.

6. High performance

Compared to centralized computer network clusters, it can provide higher performance and better cost performance.

1. **What is service oriented architecture?**

A service-oriented architecture is intended to define loosely coupled and interoperable services/applications, and to define a process for integrating these interoperable components.

1. **Define the term web service.**

A Web service is a software service used to communicate between two devices on a network.

More specifically, a Web service is a software application with a standardized way of providing interoperability between disparate applications. It does so over HTTP using technologies such as XML, SOAP, WSDL, and UDDI.

1. **What is data grid?**

It is one family of grid services that integrate multiple distributed, heterogeneous and independently managed data sources to provide data catching and/or replication mechanisms to minimize network traffic on data discovery mechanisms, which allow the user to find data based on characteristics of the data.

Examples are TeraGrid (US), EGEE (EU), DataGrid (EU), Grid 5000 (france), ChinaGrid (China), NAS (NASA), LCG (Cern), e-Science (UK), D-Grid (Nordic) and   
FutureGrid (US).

1. **What is community cloud?**

Private clouds located at the premises of a third party; owned, managed, and operated by a vendor who is bound by custom SLAs and contractual clauses with security and compliance requirements.

1. **Define virtual cluster.**

Virtual clusters are defined as a cluster with distinct boundaries formed by a logical interconnection of physical clusters via virtual machines.

1. **What is distributed file system?**

Distributed files system is a local file system available in Virtual Machine which is used to perform memory migration from one physical cluster to another.

1. **How map reduce framework executes user jobs?**

Map Reduce is a processing technique and a program model for distributed computing based on java. The Map Reduce algorithm contains two important tasks, namely Map and Reduce. Map takes a set of data and converts it into another set of data, where individual elements are broken down into tuples (key/value pairs). Secondly, reduce task, which takes the output from a map as an input and combines those data tuples into a smaller set of tuples. As the sequence of the name Map Reduce implies, the reduce task is always performed after the map job.

1. **Write a short note on Kerberos.**

Kerberos is a network authentication protocol. It is designed to provide strong authentication for client/server applications by using secret-key cryptography.

1. **What is identity and access management in cloud environment?**

Identity and access management is shortly known as IAM and it is used for Authentication, Authorization, and Auditing (AAA) of users accessing cloud services.

Traditionally, organizations invest in IAM practices to improve operational efficiency and to comply with regulatory, privacy, and data protection requirements.

**Part B**

**11a. i) Outline the architecture of a cluster of cooperative computers with a diagram.**

A computing cluster consists of interconnected stand-alone computers which work

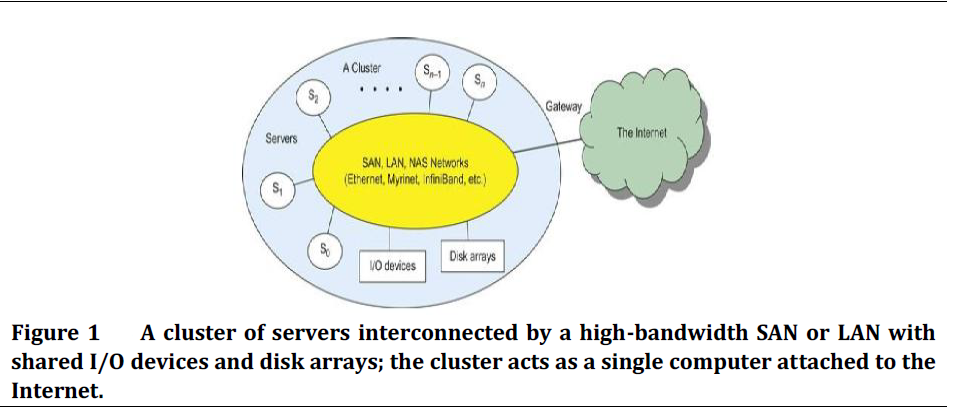
Cooperatively as a single integrated computing resource.

***Cluster Architecture***

Figure 1 shows the architecture of a typical server cluster built around a lowlatency,

high bandwidth interconnection network.

* This network can be as simple as a SAN (e.g., Myrinet) or a LAN (e.g., Ethernet).
* To build a larger cluster with more nodes, the interconnection network can be built with multiple levels of Gigabit Ethernet, Myrinet, or InfiniBand switches.
* Through hierarchical construction using a SAN, LAN, or WAN, one can build scalable clusters with an increasing number of nodes. The cluster is connected to the Internet via a virtual private network (VPN) gateway.
* The gateway IP address locates the cluster. The system image of a computer is decided by the way the OS manages the shared cluster resources.
* Most clusters have loosely coupled node computers.
* All resources of a server node are managed by their own OS. Thus, most clusters have multiple system images as a result of having many autonomous nodes under different OS control.



***Single-System Image***

* An ideal cluster should merge multiple system images into a single-system image (SSI).
* Cluster designers desire a cluster operating system or some middleware to support SSI at various levels, including the sharing of CPUs, memory, and I/O across all cluster nodes.
* An SSI is an illusion created by software or hardware that presents a collection of resources as one integrated, powerful resource.
* SSI makes the cluster appear like a single machine to the user.
* A cluster with multiple system images is nothing but a collection of independent computers.

**11a. ii) Outline the similarities and difference between distributed computing, grid computing and cloud computing.**

**Similarities between distributed computing, grid computing and cloud computing**

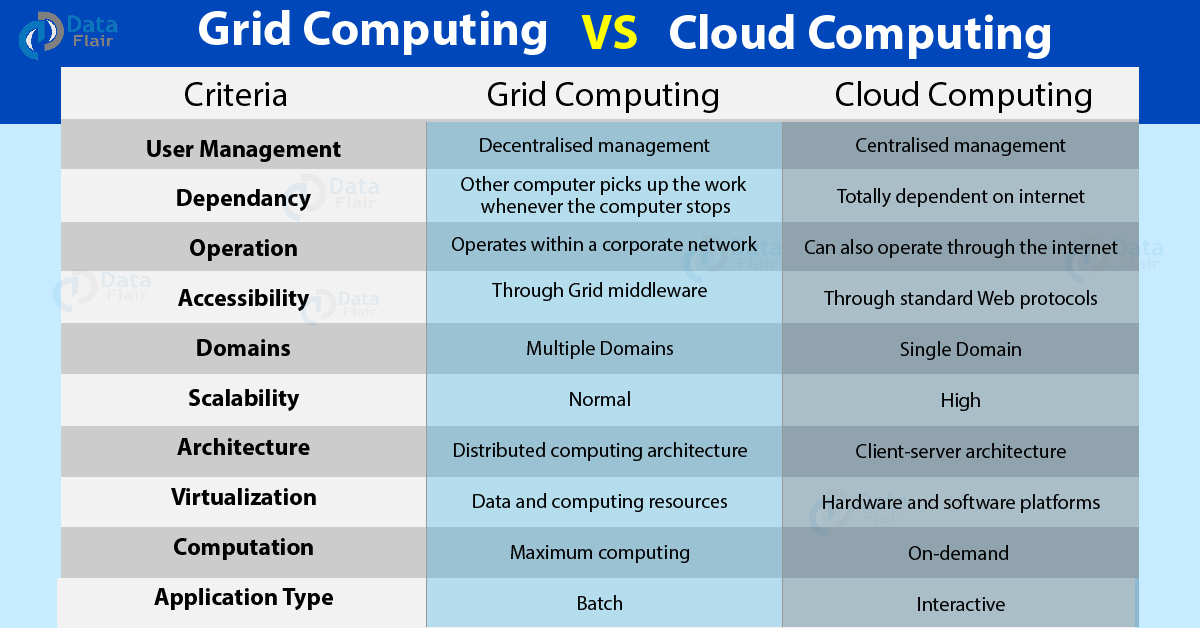
1. Scalability is the property of a system to handle a growing amount of work by adding resources to the system. It is supported by Distributed Computing (DC), Grid Computing (GC) and Cloud computing (CC).
2. Distributed computing is a base for all DC,GC and CC. DC will have loosely coupled and tightly coupled machines. GC will make use of all the loosely coupled machines to perform a larger task. CC will provide hire and use concept over internet.
3. SaaS, IaaS and PaaS. All these are there in both cloud and grid computing.
4. Grid Computing is used for high intensive computation which involves Number crunching and Cloud Computing is used for requirements where you require High CPU usage, High storage and High bandwidth
5. GC is evolved from DC. CC is evolved from GC.
6. DC covers small geographical area and no virtualization support. GC provides partial virtualization with SOA on Data Processing. CC provides complete virtualization on computing resources such as data, processor, space etc.

**Differences between distributed computing, grid computing and cloud computing**

Distributed computing is a model in which components of a software system are shared among multiple computers to improve efficiency and performance. According to the narrowest of definitions, distributed computing is limited to programs with components shared among computers within a limited geographic area. Broader definitions include shared tasks as well as program components. In the broadest sense of the term, distributed computing just means that something is shared among multiple systems which may also be in different locations.

Grid computing is a processor architecture that combines computer resources from various domains to reach a main objective. In grid computing, the computers on the network can work on a task together, thus functioning as a supercomputer. Typically, a grid works on various tasks within a network, but it is also capable of working on specialized applications. It is designed to solve problems that are too big for a supercomputer while maintaining the flexibility to process numerous smaller problems. Computing grids deliver a multiuser infrastructure that accommodates the discontinuous demands of large information processing.

Cloud computing is a method for delivering information technology (IT) services in which resources are retrieved from the Internet through web-based tools and applications, as opposed to a direct connection to a server. Rather than keeping files on a proprietary hard drive or local storage device, cloud-based storage makes it possible to save them to a remote database. As long as an electronic device has access to the web, it has access to the data and the software programs to run it. With cloud computing, you eliminate those headaches that come with storing your own data, because you’re not managing hardware and software — that becomes the responsibility of an experienced vendor like Salesforce.com: The Customer Success Platform To Grow Your Business. The shared infrastructure means it works like a utility: you only pay for what you need, upgrades are automatic, and scaling up or down is easy.

****

**11b. What is grid computing? Draw a typical view of grid environment and outline the key elements of grid.**

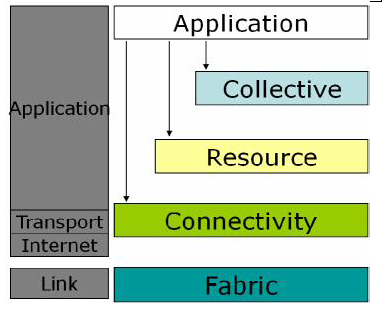
Grid computing has emerged as an important field synonymous to high throughput computing (HTC). The importance of grids is defined in terms of the amount of work they are able to deliver over a period of time.

**Definition of Grid**

[1] According to IBM’s definition “grid is a collection of distributed computing resources available over a local or wide area network that appear to an end user or application as one large virtual computing system. The vision is to create virtual dynamic organizations through secure, coordinated resource-sharing among individuals, institutions, and resources. Grid computing is an approach to distributed computing that spans not only locations but also organizations, machine architectures, and software boundaries to provide unlimited power, collaboration, and information access to everyone connected to a grid.”

[2] According to the Globus Alliance “The grid refers to an infrastructure that enables the integrated, collaborative use of high-end computers, networks, databases, and scientific instruments owned and managed by multiple organizations. Grid applications often involve large amounts of data and/or computing and often require secure resource sharing across organizational boundaries, and are thus not easily handled by today’s Internet and Web infrastructures.”

[3] Industry-formulated definition of grid computing is “A computational grid is a hardware and software infrastructure that provides dependable, consistent, pervasive, and inexpensive access to high-end computational capabilities. A grid is concerned with coordinated resource sharing and problem solving in dynamic, multiinstitutional virtual organizations. The key concept is the ability to negotiate resource-sharing arrangements among a set of participating parties (providers and consumers) and then to use the resulting resource pool for some purpose. The sharing that we are concerned with is not primarily file exchange but rather direct access to computers, software, data, and other resources, as is required by a range of collaborative problem-solving and resource-brokering strategies emerging in industry, science, and engineering. This sharing is, necessarily, highly controlled, with resource providers and consumers defining clearly and carefully just what is shared, who is allowed to share, and the conditions under which sharing occurs. A set of individuals and/or institutions defined by such sharing rules form what we call a virtual organization (VO).”

Grid architecture can be visualized as a layered architecture figure 2.

 The higher layers are focused on the user (user-centric)

 The lower layers are more focused on computers and networks (hardware-centric)

**Figure 2 Layered Grid Architecture.**

* At the base of everything, the **bottom layer** is the **network**, which assures the connectivity for the resources in the Grid.
* On top of it lies the **resource layer**, made up of the actual resources that are part of the Grid, such as computers, storage systems, electronic data catalogues, and even sensors such as telescopes or other instruments, which can be connected directly to the network.
* The **middleware layer** provides the tools that enable the various elements (servers, storage, networks, etc.) to participate in a unified Grid environment. The middleware layer can be thought of as the intelligence that brings the various elements together - the "brain" of the Grid.
* The highest layer of the structure is the **application layer**, which includes all different user applications (science, engineering, and business, financial), portals and development toolkits supporting the applications. This is the layer that users of the grid will see.

In most common Grid architectures, the application layer also provides the so-called **service ware**, the sort of general management functions such as

* measuring the amount a particular user employs the Grid
* billing for this use
* keeping accounts of who is providing resources and who is using them
* Keeps track of sharing the resources of a variety of institutions amongst large numbers of different users.
* The service ware is in the top layer, because it is something the user interacts with, whereas the middleware is a "hidden" layer that the user should not have to worry about.

**12a. What is open grid services architecture? Present a detailed view of open grid services architecture.**

The OGSA is an open source grid service standard jointly developed by academia and the IT industry under coordination of a working group in the Global Grid Forum (GGF).

* The standard was specifically developed for the emerging grid and cloud service communities. The OGSA is extended from web service concepts and technologies.
* The standard defines a common framework that allows businesses to build grid platforms across enterprises and business partners.
* The intent is to define the standards required for both open source and commercial software to support a global grid infrastructure.

The detailed view of OGSA is shown in figure3

There are two main logical components of OGSA:

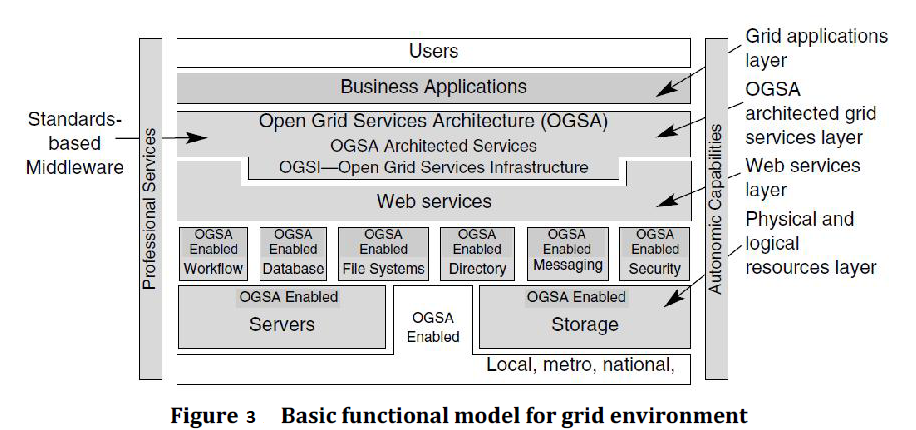
i. The Web-services-plus-OGSI layer, and

ii. The OGSA-architected services layer.

Four main layers comprise the OGSA architecture

**Grid applications layer**

* This layer is the user-visible layer.
* It supports user applications.
* Eventually, a “rich” set of grid-architected services is expected to be developed.

.

**OGSA-architected grid services layer**

* Services in this layer include:
  +  Discovery
  +  Lifecycle
  +  State management
  +  Service Groups
  +  Factory
  +  Notification, and
  +  Handle Map.

These services are based on the Web services layer.

* The GGF was working at press time to define many of these architected grid services in areas such as program execution, data services, and core services.

**Web Services layer, plus the OGSI extensions that define grid services**

The OGSI specification defines grid services and builds on standard Web services technology.

OGSI exploits the mechanisms of Web services such as XML and WSDL to specify standard interfaces, behaviours, and interaction for all grid resources.

OGSI extends the definition of Web services to provide capabilities for dynamic, stateful, and manageable Web services that are required to model the resources of the grid.

**Physical and logical resources layer**

The concept of resources is central to OGSA and to grid computing in general.Resources comprises the capabilities of the grid. Physical resources include servers, storage, and network. Above the physical resources are logical resources. Logical resources provide additional function by virtualizing and aggregating the resources in the physical layer. General-purpose middleware such as file systems, database managers, directories, and workflow managers provide these abstract services on top of the physical grid. The GGF OGSA working group found it necessary to augment core Web services functionality to address grid services requirements.

**12b. What is open grid services infrastructure? Outline the open grid services infrastructure with a diagram.**

OGSI provides a mechanism, the Handle Resolver to support client resolution of a grid service handle into a grid service reference.

OGSA integrates key grid technologies including the Globus Toolkit with Web services mechanisms to create a distributed system framework based on the OGSI.

* A grid service instance is a (potentially transient) service that conforms to a set of conventions, expressed as WSDL interfaces, extensions, and behaviors, for such purposes as lifetime management, discovery of characteristics, and notification.
* OGSI introduces standard factory and registration interfaces for creating and discovering grid services.
* OGSI defines a component model that extends WSDL and XML schema definition to incorporate the concepts of
* Stateful Web services
* Extension of Web services interfaces
* Asynchronous notification of state change
* References to instances of services
* Collections of service instances
* Service state data that augment the constraint capabilities of XML schema definition
* The OGSI V1.0 specification proposes detailed specifications for the conventions that govern how clients create, discover, and interact with a grid service instance. That is, it specifies

(1) How grid service instances are named and referenced;

(2) The base, common interfaces (and associated behaviours) that all grid services implement;

(3) The additional (optional) interfaces and behaviours associated with factories and service groups.

OGSI extends Web services by introducing interfacesand conventions in two main areas

**1. “Interfaces.”**

The dynamic and potentially transient nature of services in a grid:

Particular service instances may come and go as work is dispatched, as resources are configured and provisioned, and as system state changes.

Therefore, grid services need interfaces to manage the creation, destruction, and lifecycle management of these dynamic services.

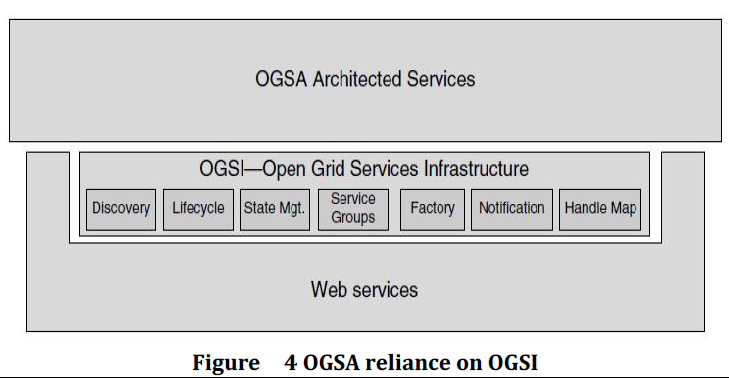
**2. “State.”**

Grid services typically have attributes and data associated with them. This is similar in concept to the traditional structure of objects in object-oriented programming:

* Objects have behaviour and data. Likewise, Web services were found to be in need of being extended to support state data associated with grid services.
* Basic Web services are stateless (e.g., add, subtract).
* Most real-world applications involve stateful transactions [e.g., query (sd2), getdata.
* State is linked to a “handle” or sessionID as a parameter.
* Protocols such as SOAP, SMTP, and FTP use state mechanisms (sessionID, packet headers, and TCP sockets, respectively).

Consistent with these two observations, OGSI introduces an interaction model for grid services. The interaction model provides a uniform way for software developers to model and interact with grid services by providing interfaces for discovery, life cycle, state management, creation and destruction, event notification, and reference management.

Figure 4 shows the services given by OGSI .



Below, we list interfaces and conventions that OGSI introduces

**Factory**

A mechanism (interface) that provides a way to create new grid services.

Factories may create temporary instances of limited function, such as a scheduler creating a service to represent the execution of a particular job; or they may create longer-lived services such as a local replica of a frequently used data set.

Not all grid services are created dynamically; for example, some services might be created as the result of an instance of a physical resource in the grid, such as a processor, storage, or network device.

**Life cycle**

A mechanism architected to prevent grid services from consuming resources indefinitely without requiring a large-scale distributed “garbage collection” scavenger.

Every grid service has a termination time set by the service creator or factory. Because grid services may be transient, grid service instances are created with a specified lifetime.

The lifetime of any particular service instance can be negotiated and extended, as required, by components that are dependent on or manage that service.

In turn, a client with appropriate authorization can use termination time information to check the availability (lease period) of the service; the client can also request to extend the current lease time by sending a keep-alive message to the service with a new termination time.

If the service accepts this request, the lease time can be extended to the new termination time requested by the client.

This soft-state life cycle is controlled by appropriate security and policy decisions of the service, and the service has the authority to control this behaviour for example, a service can arbitrarily terminate a service or can extend its termination time even while the client holds a service reference

**State management**

OGSI specifies a framework for representing this state, called service data, and a mechanism for inspecting or modifying that state, named Find/SetServiceData.

OGSI requires a minimal amount of state in service data elements that every grid service must support, and requires that all services implement the Find/SetServiceData port Type.

**Service groups**

Service groups are collections of grid services that are indexed (using service data described above) for some specific purpose. For example, they might be used to collect all the services that represent the resources in a particular cluster node within the grid.

**Notification**

* Services interact with one another by exchanging messages based on service invocation.
* The state information (the service data described above) that is modelled for grid services changes as the system runs.
* Many interactions between grid services require dynamic monitoring of changing state. Notification applies a traditional publish/subscribe paradigm to this monitoring.
* Grid services support an interface (Notification Source) to permit other grid services (Notification Sink) to subscribe to changes.
* The internal state of a grid service can keep track that this grid service has received one or zero messages.
* This reliable message delivery mechanism guaranteed by the internal state can build business-oriented transactions.
* In a transient stateful service, OGSA provides a mechanism to capture the state information associated with any operation that fails. If an operation fails, the keep alive messages cease if there is no service client for invoking this running service instance.
* Then the grid service instance automatically times out and frees the computing resources associated with this service instance

**Handle Map**

* This deals with service identity.
* When Factories are used to create a new instance of a Grid Service, the Factory returns the identity of the newly instantiated service.
* This identity is composed of two parts: a Grid Service Handle (GSH) and a Grid Service Reference (GSR).
* A GSH provides a reference the grid service indefinitely; GSR can change within the grid services lifetime.

The **Handle Map interface** provides a way to obtain a GSR given a GSH.

* The user application invokes create Grid Service requests on the Factory interface to create a new service instance.
* The newly created service instance associated with the grid service interface will be automatically allocated computing resources.
* Meanwhile, an initial lifetime of the instance can be specified before the service instance is created. The newly created service instance will keep the user credentials for performing further interactions with other systems over the Internet.
* The newly created grid service instance will be automatically assigned a globally unique called the GSH, which is used to distinguish this specific service instance from other grid service instances. These enhancements are specified in OGSI. As the OGSI specification was finalized and implementations began to appear, some standards organizations became interested in incorporating a portion of the functionality outlined in OGSI within appropriate Web services standards; hence, over time, it is expected that much of the OGSI functionality will be incorporated in Web services standards.

**13a. i) What are the pros and cons for public, private and hybrid cloud.**

A *public* cloud refers to a service that offers applications and IT functions to a general customer base with very few opportunities for individualization. However, higher education officials often define a public cloud as one that is hosted off premises by a third-party provider, but is protected in the sense that it is governed by a negotiated contract and includes customized pricing, service and security levels, as well as a strategy for data return should the arrangement end.

**PROS:** No capital investment; access to economies of scale and specialized knowledge; predictable, pay-as-you-go costs

**CONS:** Loss of control over institutional data; less ability to customize and prioritize how resources are managed

**POPULAR USES:** Student e-mail, storage, standard productivity software, non-core business applications

A *private* cloud is created for and used expressly by an individual college or university.

**PROS:** Control of data location, resource provisioning, security, service levels and compliance measures; improved flexibility and remote access

**CONS:** Resource-intensive and expensive; requires specialized IT expertise

**POPULAR USES:** Faculty and staff e-mail, enterprise resource planning (ERP), human resources and finance systems, learning management systems (LMS), classroom applications

A *hybrid* cloud within higher education typically comes in two flavors. The first is a cloud run by a consortium or a higher education–focused provider for use only by a group of academic institutions. The second is an arrangement in which the underlying infrastructure, or all or part of the application, is hosted offsite by a public cloud provider, but more sensitive components (and sometimes the application itself) are maintained within a private cloud by the higher education institution.

**PROS:** Speed to market; access to specialized application knowledge and high-capacity IT resources

**CONS:** Loss of control

**POPULAR USES:** ERP, LMS, high-performance computing, massive data storage and management

**13a. ii) Explain virtualization of I/O devices with an example.**

Virtualization is a computer architecture technology by which multiple virtual machines (VMs) are **multiplexed** in the same hardware machine. The purpose of a VM is to **enhance resource sharing** by many users and improve computer performance in terms of resource utilization and application flexibility. **Hardware resources** (CPU, memory, I/O devices, etc.) or **software resources** (operating system and software libraries) can be virtualized in various functional layers. The idea is to separate the hardware from the software to yield **better system efficiency**. Virtualization techniques can be applied to **enhance the use of compute engines, networks, and storage**.

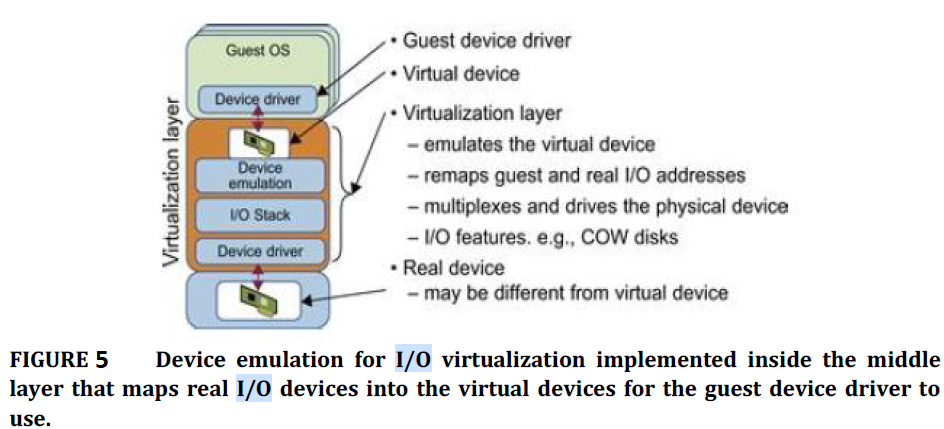
I/O virtualization involves managing the routing of I/O requests between virtual devices and the shared physical hardware.

There are three ways to implement I/O virtualization:

* full device emulation,
* para-virtualization, and
* direct I/O

**Full device emulation**

* This approach emulates well-known, real-world devices.
* All the functions of a device or bus infrastructure, such as device enumeration, identification, interrupts, and DMA, are replicated in software.
* This software is located in the VMM and acts as a virtual device. The I/O access requests of the guest OS are trapped in the VMM which interacts with the I/O devices.
* The full device emulation approach is shown in Figure 5.
* A single hardware device can be shared by multiple VMs that run concurrently.



**The para-virtualization method or split driver model**

* Consisting of a frontend driver and a backend driver. The frontend driver is running in Domain U and the backend driver is running in Domain 0.
* They interact with each other via a block of shared memory.
* The frontend driver manages the I/O requests of the guest OSes and the backend driver is responsible for managing the real I/O devices and multiplexing the I/O data of different VMs.
* Although para-I/O virtualization achieves better device performance than full device emulation, it comes with a higher CPU overhead.

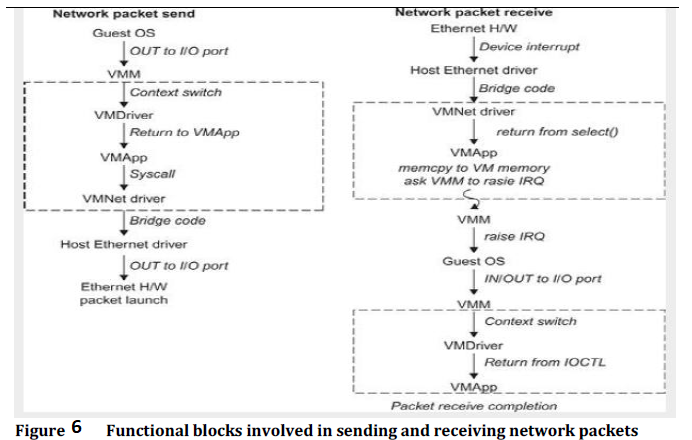
**Direct I/O virtualization**

* The VM access devices directly.
* It can achieve close-to-native performance without high CPU costs.
* There are a lot of challenges for commodity hardware devices.
* For example, when a physical device is reclaimed (required by workload migration) for later reassignment, it may have been set to an arbitrary state (e.g., DMA to some arbitrary memory locations) that can function incorrectly or even crash the whole system.
* Hardware-assisted I/O virtualization is critical.
* Intel VT-d supports the remapping of I/O DMA transfers and device-generated interrupts.
* The architecture of VT-d provides the flexibility to support multiple usage models that may run unmodified, special-purpose, or “virtualization-aware” guest OSes.

**Self-virtualized I/O** (SV-IO)

* The key idea of SVIO is to harness the rich resources of a multicore processor. All tasks associated with virtualizing an I/O device are encapsulated in SV-IO.
* It provides virtual devices and an associated access API to VMs and a management API to the VMM. SV-IO defines one virtual interface (VIF) for every kind of virtualized I/O device, such as virtual network interfaces, virtual block devices (disk), virtual camera devices, and others.
* The guest OS interacts with the VIFs via VIF device drivers.
* Each VIF consists of two message queues. One is for outgoing messages to the devices and the other is for incoming messages from the devices.
* Each VIF has a unique ID for identifying it in SV-IO.

**Example: VMware Workstation for I/O Virtualization**

The VMware Workstation runs as an application. It leverages the I/O device support in guest OSes, host OSes, and VMM to implement I/O virtualization. The application portion (VMApp) uses a driver loaded into the host operating system (VMDriver) to establish the privileged VMM, which runs directly on the hardware. A given physical processor is executed in either the host world or the VMM world, with the VMDriver facilitating the transfer of control between the two worlds. The VMware Workstation employs full device emulation to implement I/O virtualization. Figure 6 shows the functional blocks used in sending and receiving packets via the emulated virtual NIC.

**13b. What is data center? Outline the issues to be addressed with respect to virtualization for data center automation.**

A data center (or datacenter) is a facility composed of networked computers and storage that businesses or other organizations use to organize, process, store and disseminate large amounts of data. A business typically relies heavily upon the applications, services and data contained within a data center, making it a focal point and critical asset for everyday operations.

Data centers have grown rapidly in recent years, and all major IT companies are pouring their resources into building new data centers. In addition, Google, Yahoo!, Amazon, Microsoft, HP, Apple, and IBM are all in the game. All these companies have invested billions of dollars in data center construction and automation. Data-center automation means that huge volumes of hardware, software, and database resources in these data centers can be allocated dynamically to millions of Internet users simultaneously, with guaranteed QoS and cost effectiveness. This automation process is triggered by the growth of virtualization products and cloud computing services. The latest virtualization development highlights high availability (HA), backup services, workload balancing, and further increases in client bases. IDC projected that automation, service orientation, policy-based, and variable costs in the virtualization market.

***Server Consolidation in Data Centers***

In data centers, a large number of heterogeneous workloads can run on servers at various times. These heterogeneous workloads can be roughly divided into two categories:

**Chatty workloads** - these may burst at some point and return to a silent state at some other point. A web video service is an example of this, whereby a lot of people use it at night and few people use it during the day.

**Noninteractive workloads**- these workloads do not require people’s efforts to make progress after they are submitted. High-performance computing is a typical example of this. At various stages, the requirements for resources of these workloads are dramatically different. However, to guarantee that a workload will always be able to cope with all demand levels, the workload is statically allocated enough resources so that peak demand is satisfied. Therefore, it is common that most servers in **data centers are underutilized**. A large amount of hardware, space, power, and management cost of these servers is wasted.

**Server consolidation is an approach** to improve the low utility ratio of hardware resources by reducing the number of physical servers. Among several server consolidation techniques such as centralized and physical consolidation, virtualization based server consolidation is the most powerful. Data centers need to optimize their resource management. Yet these techniques are performed with the granularity of a full server machine, which makes resource management far from well optimized.

Server virtualization enables smaller resource allocation than a physical machine The use of VMs increases resource management complexity. This causes a challenge in terms of how to improve resource utilization as well as guarantee QoS in data centers. In detail, server virtualization has the following side effects:

Consolidation enhances hardware utilization. Many underutilized servers are consolidated into fewer servers to enhance resource utilization. Consolidation also facilitates backup services and disaster recovery.

This approach enables more agile provisioning and deployment of resources. In a virtual environment, the images of the guest OSes and their applications are readily cloned and reused.

The total cost of ownership is reduced. In this sense, server virtualization causes deferred purchases of new servers, a smaller data-center footprint, lower maintenance costs, and lower power, cooling, and cabling requirements.

This approach improves availability and business continuity. The crash of a guest OS has no effect on the host OS or any other guest OS. It becomes easier to transfer a VM from one server to another, because virtual servers are unaware of the underlying hardware.

To automate data-center operations, one must consider

* resource scheduling
* architectural support
* power management
* automatic or autonomic resource management
* performance of analytical models, and so on.

In **virtualized data centers**, an efficient, on-demand, fine-grained scheduler is one of the key

factors to improve resource utilization. Scheduling and reallocations can be done in a wide range of levels in a set of data centers.

The levels match at least at the VM level, server level, and data-center level. Ideally, scheduling and resource reallocations should be done at all levels. However, due to the complexity of this, current techniques only focus on a single level or, at most, two levels.

**Dynamic CPU allocation** is based on VM utilization and application-level QoS metrics.

One method considers both CPU and memory flowing as well as automatically adjusting resource overhead based on varying workloads in hosted services. Another scheme uses a two-level resource management system to handle the complexity involved. A local controller at the VM level and a global controller at the server level are designed. They implement autonomic resource allocation via the interaction of the local and global controllers. Multicore and virtualization are two cutting techniques that can enhance each other.

***Virtual Storage Management***

The term “storage virtualization” was widely used before the renaissance of system virtualization. Storage virtualization was largely used to describe the aggregation and repartitioning of disks at very coarse time scales for use by physical machines. Generally, the

data stored can be classified into two categories: VM images and application data. The VM images are special to the virtual environment, while application data includes all other data which is the same as the data in traditional OS environments.

**Encapsulation and isolation** - Traditional operating systems and applications running on them can be encapsulated in VMs. Only one operating system runs in a virtualization while many applications run in the operating system. System virtualization allows multiple VMs to run on a physical machine and the VMs are completely isolated. To achieve encapsulation and isolation, both the system software and the hardware platform, such as CPUs and chipsets, are rapidly updated.

**Storage is lagging -** The storage systems become the main bottleneck of VM deployment. In

virtualization environments, a virtualization layer is inserted between the hardware and traditional operating systems or a traditional operating system is modified to support virtualization. This procedure complicates storage operations.

**VMs are not nimble -** Hence, operations such as remapping volumes across hosts and check

pointing disks are frequently clumsy and esoteric, and sometimes simply unavailable. Since traditional storage management techniques do not consider the features of storage in virtualization environments, Parallax designs a novel architecture in which storage features that have traditionally been implemented directly on high-end storage arrays and switchers are relocated into a federation of storage VMs. These storage VMs share the same physical hosts as the VMs that they serve.

**14a. Explain main components and programming model for Globus Toolkit.**

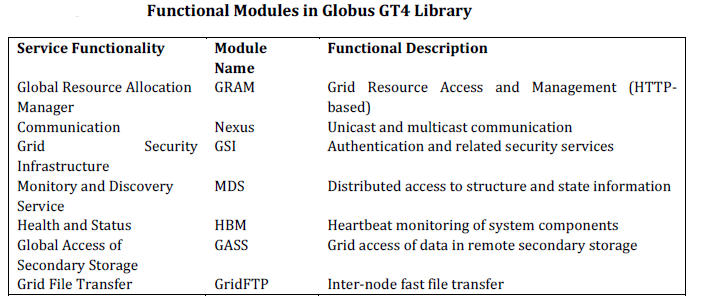
The Globus Toolkit, started in 1995 with funding from DARPA, is an open middleware library for the grid computing communities. These open source software libraries support many operational grids and their applications on an international basis. The toolkit addresses

common problems and issues related to grid resource discovery, management, communication, security, fault detection, and portability. The software itself provides a variety of components and capabilities. The library includes a rich set of service implementations.

The implemented software supports grid infrastructure management, provides tools for building new web services in Java, C, and Python, builds a powerful standard-based security infrastructure and client APIs (in different languages), and offers comprehensive command-line programs for accessing various grid services. The Globus Toolkit was initially motivated by a desire to remove obstacles that prevent seamless collaboration, and thus sharing of resources and services, in scientific and engineering applications. The shared resources can be computers, storage, data, services, networks, science instruments (e.g., sensors), and so on.

***The GT4 Library***

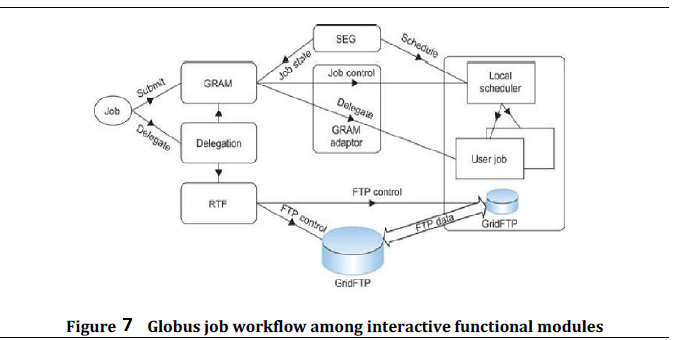
GT4 offers the middle-level core services in grid applications.

* The **high-level services and tools**, such as MPI, Condor-G, and Nirod/G, are developedby third parties for general-purpose distributed computing applications.
* The **local services**, such as LSF, TCP, Linux, and Condor, are at the bottom level and are fundamental tools supplied by other developers.
* Below given table summarizes GT4’s core grid services by module name.
* Nexus is used for collective communications and HBM for heartbeat monitoring of resource nodes. GridFTP is for speeding up inter node file transfers. The module GASSis used for global access of secondary storage.

**Globus Job Workflow**

Figure 7 shows the typical job workflow when using the Globus tools. A typical job execution sequence proceeds as follows:

* The user delegates his credentials to a delegation service.
* The user submits a job request to GRAM with the delegation identifier as a parameter.
* GRAM parses the request, retrieves the user proxy certificate from the delegation service, and then acts on behalf of the user.
* GRAM sends a transfer request to the RFT (Reliable File Transfer), which applies GridFTP to bring in the necessary files.
* GRAM invokes a local scheduler via a GRAM adapter and the SEG (Scheduler Event Generator) initiates a set of user jobs.
* The local scheduler reports the job state to the SEG. Once the job is complete, GRAM uses RFT and GridFTP to stage out the resultant files.
* The grid monitors the progress of these operations and sends the user a notification when they succeed, fail, or are delayed.

**14b. Explain Hadoop Distributed File System Architecture with Diagram.**

Hadoop comes with a distributed filesystem called HDFS, which stands for Hadoop Distributed Filesystem. HDFS is Hadoop’s flagship file system but Hadoop actually has a general purpose file system abstraction, so we’ll see along the way how Hadoop integrates with other storage systems (such as the local filesystem and Amazon S3).

***The Design of HDFS***

HDFS is a file system designed for storing very large files with streaming data access patterns, running on clusters of commodity hardware. Let’s examine this statement in more

detail:

**Very large files-**“Very large” in this context means files that are hundreds of megabytes, gigabytes, or terabytes in size. There are Hadoop clusters running today that store petabytes of data.

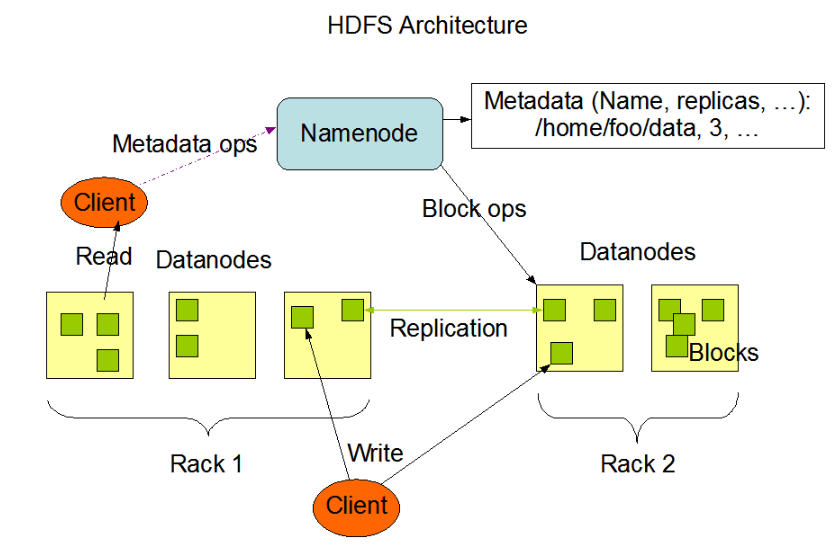
**Streaming data access-** HDFS is built around the idea that the most efficient data generated or copied from source, and then various analyses are performed on that processing pattern is a write-once, read-many-times pattern. A dataset is typically dataset over time. Each analysis will involve a large proportion, if not all, of the dataset, so the time to read the whole dataset is more important than the latency in reading the first record.

**Commodity hardware-** Hadoop doesn’t require expensive, highly reliable hardware. It’s designed to run on clusters of commodity hardware (commonly available hardware that can be obtained from multiple vendors)3 for which the chance of node failure across the cluster is high, at least for large clusters. HDFS is designed to carry on working without a noticeable interruption to the user in the face of such failure. It is also worth examining the applications for which using HDFS does not work so well. Although this may change in the future, these are areas where HDFS is not a good fit today:

**Low-latency data access-** Applications that require low-latency access to data, in the tens of milliseconds range, will not work well with HDFS. Remember, HDFS is optimized for delivering a high throughput of data.

**Lots of small files-** Because the name node holds file system metadata in memory, the limit to the number of files in a file system is governed by the amount of memory on the name node. As a rule of thumb, each file, directory, and block takes about 150 bytes. So, for example, if you had one million files, each taking one block, you would need at least 300 MB of memory. Although storing millions of files is feasible, billions is beyond the capability of current hardware.

**Multiple writers, arbitrary file modifications**- Files in HDFS may be written to by a single writer. Writes are always made at the end of the file, in append-only fashion. There is no support for multiple writers or for modifications at arbitrary offsets in the file. The HDFS is illustrated in figure8.

**Figure 8 HDFS Architecture.**

***HDFS Concepts***

**Blocks**

A disk has a block size, which is the minimum amount of data that it can read or write. HDFS has the concept of a block, but it is a much larger unit 128 MB by default. Like in a filesystem for a single disk, files in HDFS are broken into block-sized chunks, which are stored as independent units. Unlike a filesystem for a single disk, a file in HDFS that is smaller than a single block does not occupy a full block’s worth of underlying storage. (For example, a 1 MB

file stored with a block size of 128 MB uses 1 MB of disk space, not 128 MB.) When unqualified, the term “block” in this book refers to a block in HDFS.

**Large blocks in HDFS**

HDFS blocks are large compared to disk blocks, and the reason is to minimize the cost of seeks. If the block is large enough, the time it takes to transfer the data from the disk can be

significantly longer than the time to seek to the start of the block. Thus, transferring a large file made of multiple blocks operates at the disk transfer rate.

Having a block abstraction for a distributed file system brings several benefits.

**A file can be larger than any single disk in the network**. There’s nothing that requires the blocks from a file to be stored on the same disk, so they can take advantage of any of the disks in the cluster. In fact, it would be possible, if unusual, to store a single file on an HDFS cluster whose blocks filled all the disks in the cluster.

**Making the unit of abstraction a block rather than a file simplifies the storage subsystem**. Simplicity is something to strive for in all systems, but it is especially important for a distributed system in which the failure modes are so varied. The storage subsystem deals with blocks, simplifying storage management.

**Blocks fit well with replication for providing fault tolerance and availability**. To insure against corrupted blocks and disk and machine failure, each block is replicated to a small number of physically separate machines (typically three). If a block becomes unavailable, a copy can be read from another location in a way that is transparent to the client.

**Namenodes and Datanodes**

An HDFS cluster has two types of nodes operating in a master−worker pattern:

**A namenode (the master)**:

* The **namenode** manages the filesystem namespace.
* It maintains the filesystem tree and the metadata for all the files and directories in the tree.
* This information is stored persistently on the local disk in the form of two files: the namespace image and the edit log.

**A number of datanodes (workers):**

* The namenode also knows the **datanodes** on which all the blocks for a file are located
* it does not store block locations persistently, because this information is reconstructed from datanodes when the system starts.
* A **client** accesses the filesystem on behalf of the user by communicating with the namenode and datanodes.
* The client presents a filesystem interface similar to a Portable Operating System Interface (POSIX), so the user code does not need to know about the namenode and datanodes to function.
* Datanodes are the workhorses of the filesystem.
* They store and retrieve blocks when they are told to (by clients or the namenode), and they report back to the namenode periodically with lists of blocks that they are storing.
* Without the namenode, the filesystem cannot be used.
* if the machine running the namenode were obliterated, all the files on the filesystem would be lost since there would be no way of knowing how to reconstruct the files from the blocks on the datanodes.
* For this reason, it is important to make the namenode resilient to failure, and Hadoop provides two mechanisms for this.

**The first way is to back up the files** that make up the persistent state of the filesystem metadata. Hadoop can be configured so that the namenode writes its persistent state to multiple filesystems. These writes are synchronous and atomic. The usual configuration choice is to write to local disk as well as a remote NFS mount.

It is also possible to **run a secondary namenode**, which despite its name does not act as a namenode. Its main role is to periodically merge the namespace image with the edit log to prevent the edit log from becoming too large. The secondary namenode usually runs on a separate physical machine because it requires plenty of CPU and as much memory as the namenode to perform the merge. It keeps a copy of the merged namespace image, which can be used in the event of the namenode failing. However, the state of the secondary namenode lags that of the primary, so in the event of total failure of the primary, data loss is almost certain. The usual course of action in this case is to copy the namenodes metadata files that are on NFS to the secondary and run it as the new primary

**15a. Define authentication and authorization. Outline authentication and authorization in grids with relevant examples.**

**Authentication**

Authentication is about validating your credentials such as Username/User ID and password to verify your identity. The system then checks whether you are what you say you are using your credentials. Whether in public or private networks, the system authenticates the user identity through login passwords. Usually authentication is done by a username and password, although there are other various ways to be authenticated.

Authentication factors determine the many different elements the system uses to verify one’s identity before granting the individual access to anything. An individual’s identity can be determined by what the person knows, and when it comes to security at least two or all the three authentication factors must be verified in order to grant someone permission to the system. Based on the security level, authentication factors can vary from one of the following:

* **Single- Factor Authentication:**This is the simplest form of authentication method which requires a password to grant user access to a particular system such as a website or a network. The person can request access to the system using only one of the credentials to verify one’s identity. For example, only requiring a password against a username would be a way to verify a login credential using single- factor authentication.
* **Two- Factor Authentication:**This authentication requires a two- step verification process which not only requires a username and password, but also a piece of information only the user knows. Using a username and password along with a confidential information makes it that much harder for hackers to steal valuable and personal data.
* **Multi- Factor Authentication:**This is the most advanced method of authentication which requires two or more levels of security from independent categories of authentication to grant user access to the system. This form of authentication utilizes factors that are independent of each other in order to eliminate any data exposure. It is common for financial organizations, banks, and law enforcement agencies to use multiple- factor authentication.

**Authorization**

Authorization occurs after your identity is successfully authenticated by the system, which therefore gives you full access to resources such as information, files, databases, funds, etc. However authorization verifies your rights to grant you access to resources only after determining your ability to access the system and up to what extent. In other words, authorization is the process to determine whether the authenticated user has access to the particular resources. A good example of this is, once verifying and confirming employee ID and passwords through authentication, the next step would be determining which employee has access to which floor and that is done through authorization.

Access to a system is protected by authentication and authorization, and they are frequently used in conjunction with each other. Although both have different concepts behind then, they are critical to the web service infrastructure, especially when it comes to being granted access to a system. Understanding each term is very important and a key aspect of security.

The grid is increasingly deployed as a common approach to constructing dynamic, inter domain, distributed computing and data collaborations, “**lack of security/trust between different services**” is still an important challenge of the grid.

The grid requires a security infrastructure with the following properties:

 easy to use

 conforms with the VO’s security needs while working well with site policies of each resource provider site

 and provides appropriate authentication and encryption of all interactions

**Grid Security Infrastructure**

 The GSI is an important step toward satisfying these requirements.

 As a well-known security solution in the grid environment, GSI is a portion of the Globus Toolkit and provides fundamental security services needed to support grids, including supporting for

 message protection

 authentication and delegation

 and authorization

 GSI enables secure authentication and communication over an open network, and permits mutual authentication across and among distributed sites with single sign-on capability.

 No centrally managed security system is required, and the grid maintains the integrity

of its members’ local policies.

 GSI supports both message-level security, which supports the WS Security standard and the WS-Secure Conversation specification to provide message protection for SOAP messages, and transport-level security, which means authentication via TLS with support for X.509 proxy certificates.

**Example: Mutual Authentication between Two Parties**

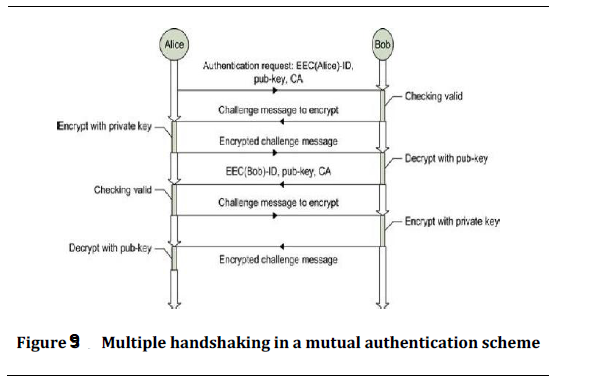
Mutual authentication is processes by which two parties with certificates signed by the CA prove to each other that they are who they say they are based on the certificate and the trust of the CAs that signed each other’s certificates.

GSI uses the **Secure Sockets Layer** (SSL) for its mutual authentication protocol, which is described in Figure 9. To mutually authenticate, the first person (Alice) establishes a connection to the second person (Bob) to start the authentication process.

Alice gives Bob her certificate. The certificate tells Bob who Alice is claiming to be (the identity), what Alice’s public key is, and what CA is being used to certify the certificate. Bob will first make sure the certificate is valid by checking the CA’s digital signature to make sure

the CA actually signed the certificate and the certificate hasn’t been tampered with. Once Bob has checked out Alice’s certificate, Bob must make sure Alice really is the person identified in the certificate.

Bob generates a random message and sends it to Alice, asking Alice to encrypt it. Alice encrypts the message using her private key, and sends it back to Bob. Bob decrypts the message using Alice’s public key. If this results in the original random message, Bob knows Alice is who she says she is. Now that Bob trusts Alice’s identity, the same operation must happen in reverse. Bob sends Alice his certificate, and Alice validates the certificate and sends a challenge message to be encrypted. Bob encrypts the message and sends it back to Alice, and Alice decrypts it and compares it with the original. If it matches, Alice knows Bob is who he says he is.



**15b. Describe Infrastructure as a Service (IaaS), Platform as a Service (Paas) and Software as a Service (SaaS) with an example.**

**IAAS: INFRASTRUCTURE AS A SERVICE**

Cloud infrastructure services, known as Infrastructure as a Service (IaaS), are self-service models for accessing, monitoring, and managing remote datacenter infrastructures, such as compute (virtualized or bare metal), storage, networking, and networking services (e.g. firewalls). Instead of having to purchase hardware outright, users can purchase IaaS based on consumption, similar to electricity or other utility billing.

Compared to SaaS and PaaS, IaaS users are responsible for managing applications, data, runtime, middleware, and OSes. Providers still manage virtualization, servers, hard drives, storage, and networking. Many IaaS providers now offer databases, messaging queues, and other services above the virtualization layer as well. Some tech analysts draw a distinction here and use the IaaS+ moniker for these other options. What users gain with IaaS is infrastructure on top of which they can install any required platform. Users are responsible for updating these if new versions are released.

IaaS Examples: Amazon Web Services (AWS), Cisco Metapod, Microsoft Azure, Google Compute Engine (GCE), Joyent

**PAAS: PLATFORM AS A SERVICE**

Cloud platform services, or Platform as a Service (PaaS), are used for applications, and other development, while providing cloud components to software. What developers gain with PaaS is a framework they can build upon to develop or customize applications. PaaS makes the development, testing, and deployment of applications quick, simple, and cost-effective. With this technology, enterprise operations, or a third-party provider, can manage OSes, virtualization, servers, storage, networking, and the PaaS software itself. Developers, however, manage the applications.

Enterprise PaaS provides line-of-business software developers a self-service portal for managing computing infrastructure from centralized IT operations and the platforms that are installed on top of the hardware. The enterprise PaaS can be delivered through a hybrid model that uses both public IaaS and on-premise infrastructure or as a pure private PaaS that only uses the latter.

Similar to the way in which you might create macros in Excel, PaaS allows you to create applications using software components that are built into the PaaS (middleware). Applications using PaaS inherit cloud characteristic such as scalability, high-availability, multi-tenancy, SaaS enablement, and more. Enterprises benefit from PaaS because it reduces the amount of coding necessary, automates business policy, and helps migrate apps to hybrid model. For the needs of enterprises and other organizations, Apprenda is one provider of a private cloud PaaS for .NET and Java.

Enterprise PaaS Examples: Apprenda

**SAAS: SOFTWARE AS A SERVICE**

Cloud application services, or Software as a Service (SaaS), represent the largest cloud market and are still growing quickly. SaaS uses the web to deliver applications that are managed by a third-party vendor and whose interface is accessed on the clients’ side. Most SaaS applications can be run directly from a web browser without any downloads or installations required, although some require plugins.

Because of the web delivery model, SaaS eliminates the need to install and run applications on individual computers. With SaaS, it’s easy for enterprises to streamline their maintenance and support, because everything can be managed by vendors: applications, runtime, data, middleware, OSes, virtualization, servers, storage and networking.

Popular SaaS offering types include email and collaboration, customer relationship management, and healthcare-related applications. Some large enterprises that are not traditionally thought of as software vendors have started building SaaS as an additional source of revenue in order to gain a competitive advantage.

SaaS Examples: Google Apps, Salesforce, Workday, Concur, Citrix GoToMeeting, Cisco WebEx

**Part C**

**16a. Elaborate hybrid trust model for grid security enforcement.**

There are two major trust models used in grid security enforcement. They are Reputation based and Fuzzy Trust Model.

***Reputation-Based Trust Model***

 In a reputation-based model, jobs are sent to a resource site only when the site is trustworthy to meet users’ demands. The site trustworthiness is usually calculated from the following information:

 the defense capability

 direct reputation

 recommendation trust

 The **defense capability** refers to the site’s ability to protect itself from danger. It is assessed according to such factors as intrusion detection, firewall, response capabilities, anti-virus capacity, and so on.

 **Direct reputation** is based on experiences of prior jobs previously submitted to the site.

 The **reputation** is measured by many factors such as prior job execution success rate, cumulative site utilization, job turnaround time, job slowdown ratio, and so on.

 A **positive experience** associated with a site will improve its reputation. On the contrary, a **negative experience** with a site will decrease its reputation.

***A Fuzzy-Trust Model***

 In this model the job security demand (SD) is supplied by the user programs.

 The trust index (TI) of a resource site is aggregated through the fuzzy-logic inference process over all related parameters.

 One can use a **two-level fuzzy logic** to estimate the aggregation of numerous trust parameters and security attributes into scalar quantities that are easy to use in the job scheduling and resource mapping process.

 The TI is normalized as a single real number with

o 0 representing the condition with the highest risk at a site

o 1 representing the condition which is totally risk-free or fully trusted.

 The fuzzy inference is accomplished through four steps:

o fuzzification,

o inference,

o aggregation, and

o defuzzification.

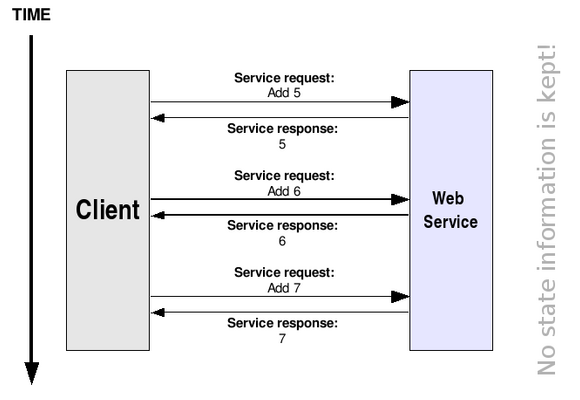
 The second salient feature of the trust model is that if a site’s trust index cannot match the job security demand (i.e., SD > TI), the trust model could deduce detailed security features to guide the site security upgrade as a result of tuning the fuzzy system.

***Fuzzy Reputation Model as a Hybrid Model.***

P2P grid computing combines and integrates grid and P2P technologies to implement peer-to-peer communications with greater distribution and scalability. Trust management is a complicated and difficult task in such an environment, because resources are geographically distributed and belong to distinct organizations . In a P2P environment, some peers may provide services with low quality and may not promise to satisfy user requirements. An unfavourable situation arises when providers offer incorrect information about their resources/services to exaggerate the quality of their services [3]. To inspire resource sharing among nodes and protect against malicious node behaviours, a reputation system for trust management is necessary. Such system allows nodes to estimate the trustworthiness of others and to selectively interact with the more respectable ones and avoid egocentric, dishonest, and malevolent node behaviours. We combine reputation-based trust, fuzzy theory, and a web of trust and recursively propagating trust to propose a model for trust management in semantic P2P grids. Our approach is usable in both P2P Grid systems and semantic P2P grid systems. In fact the usable of semantic is just to create Semantic Overlay Network (SON) which has lots of advantages for clustering the network. SON improves query performance and maintains a high degree of node autonomy. In another word, semantic is just used for clustering the grid environment based on the similarity between resources while fuzzy theory is used for trust management

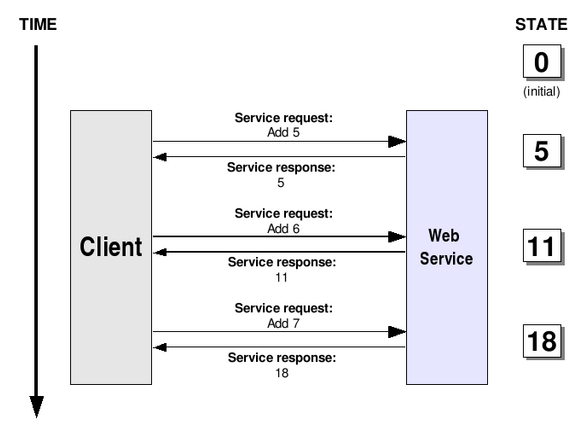
**16b. Describe a framework for building a web service with Globus Toolkit.**

## *WSRF:* The Web Services Resource Framework

Plain Web services are usually stateless (even though, in theory, there is nothing in the Web Services Architecture that says they can't be stateful). This means that the Web service can't "remember" information, or keep state, from one invocation to another. For example, imagine we want to program a very simple Web service which simply acts as an integer accumulator. This accumulator is initialized to zero, and we want to be able to add (accumulate) values in it. Suppose we have an **add** operation which receives the value to add and returns the current value of the accumulator. As shown in the following figure, our first invocation of this operation might seem to work (we request that 5 be added, and we receive 5 in return). However, since a Web service is stateless, the following invocations have no idea of what was done in the previous invocations. So, in the second call to **add** we get back 6, instead of 11 (which would be the expected value if the Web service was able to keep state).

The fact that Web services don't keep state information is not necessarily a bad thing. There are plenty of applications which have no need whatsoever for statefulness. For example, the Weather Web service we saw in the previous section is a real, working Web service which has no need to know what happened in the previous invocations.

However, Grid applications do generally require statefulness. So, we would ideally like our Web service to somehow keep state information:

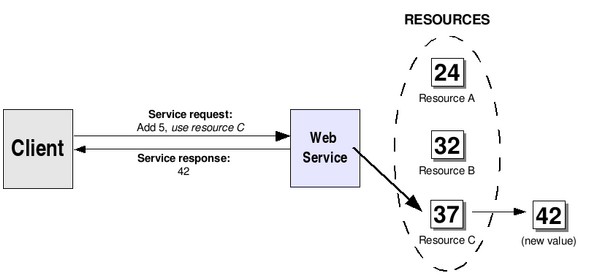
However, this is a pretty peculiar dilemma since, as mentioned above, a Web service is usually a stateless entity. In fact, some people might argue that a "stateful Web service" is a bit of a contradiction in terms! So, how do we get out of this jam?

### The resource approach to statefulness

Giving Web services the ability to keep state information while still keeping them stateless seems like a complex problem. Fortunately, it's a problem with a very simple solution: simply keep the Web service and the state information completely separate.

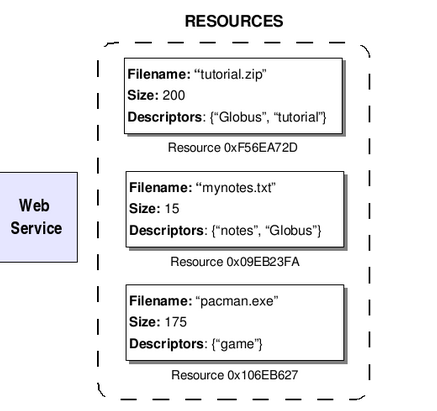
Instead of putting the state in the Web service (thus making it stateful, which is generally regarded as a bad thing) we will keep it in a separate entity called a resource, which will store all the state information. Each resource will have a unique key, so whenever we want a stateful interaction with a Web service we simply have to instruct the Web service to use a particular resource.

For example, take the accumulator example. As shown in the next figure, our Web service could have three different resources (A, B, C) to choose from. If we want the integer value to be 'remembered' from invocation to invocation, the client simply has to specify that he wants a method invoked with a certain resource.



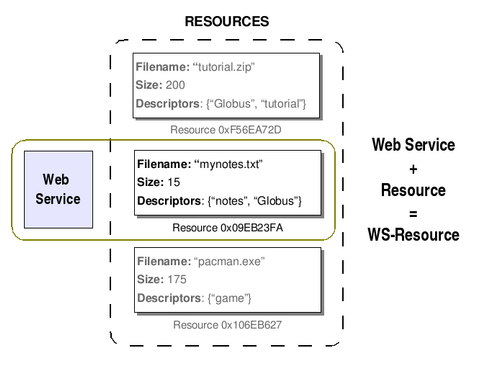
In the figure we can see that the client wants the **add** operation invoked with resource C. When the Web service receives the **add** request, it will make sure to retrieve resource C so that **add** is actually performed on that resource. The resource themselves can be stored in memory, on secondary storage, or even in a database. Also, notice how a Web service can have access to more than one resource.

Of course, resources can come in all different shapes and sizes. A resource can keep multiple values (not just a simple integer value, as shown in the previous figure). For example, our resources could represent files:



A URI might be enough to address the Web service, but how do we specify the resource on top of that? There are actually several different ways of doing this. As we'll see later on, the preferred way of doing it is to use a relatively new specification called WS-Addressing which provides a more versatile way of addressing Web Services (when compared to plain URIs).

Finally, a bit of terminology before we continue. A pairing of a Web service with a resource is called a WS-Resource. The address of a particular WS-Resource is called an endpoint reference (this is WS-Addressing lingo).



The Web Services Resources Framework is a collection of five different specifications. Of course, they all relate (in some way or another) to the management of WS-Resources.

#### WS-ResourceProperties

A resource is composed of zero or more resource properties. For example, in the figure shown above each resource has three resource properties: Filename, Size, and Descriptors. WS-ResourceProperties specifies how resource properties are defined and accessed. As we'll see later on when we start programming, the resource properties are defined in the Web service's WSDL interface description.

#### WS-ResourceLifetime

Resources have non-trivial lifecycles. In other words, they're not a static entity that is created when our server starts and destroyed when our server stops. Resources can be created and destroyed at any time. The WS-ResourceLifetime supplies some basic mechanisms to manage the lifecycle of our resources.

#### WS-ServiceGroup

We will often be interested in managing groups of Web Services or groups of WS-Resources, and performing operations such as 'add new service to group', 'remove this service from group', and (more importantly) 'find a service in the group that meets condition FOOBAR'. The WS-ServiceGroup specifies how exactly we should go about grouping services or WS-Resources together. Although the functionality provided by this specification is very basic, it is nonetheless the base of more powerful discovery services (such as GT4's IndexService) which allow us to group different services together and access them through a single point of entry (the service group).

#### WS-BaseFaults

Finally, this specification aims to provide a standard way of reporting faults when something goes wrong during a WS-Service invocation.

#### WS-Notification

WS-Notification is another collection of specifications that, although not a part of WSRF, is closely related to it. This specification allows a Web service to be configured as a notification producer, and certain clients to be notification consumers (or subscribers). This means that if a change occurs in the Web service (or, more specifically, in one of the WS-Resources), that change is notified to all the subscribers (not all changes are notified, only the ones the Web services programmer wants to).

#### WS-Addressing

As mentioned before, the WS-Addressing specification provides us a mechanism to address Web services which is much more versatile than plain URIs. In particular, we can use WS-Addressing to address a Web service + resource pair (a WS-Resource).