

UNIT II CLOUD ENABLING TECHNOLOGIES

Service Oriented Architecture – REST and Systems of Systems – Web Services – Publish-Subscribe Model – Basics of Virtualization – Types of Virtualization – Implementation Levels of Virtualization – Virtualization Structures – Tools and Mechanisms – Virtualization of CPU – Memory – I/O Devices – Virtualization Support and Disaster Recovery.

Introduction

Web Service

⊠ Generic definition

- Any application accessible to other applications over the Web.

⊠ Definition of the UDDI consortium

- Web services are self-contained, modular business applications that have open, Internet-oriented, standards-based interfaces.

⊠ Definition of the World Wide Web Consortium (W3C)

- A Web service is a software system designed to support interoperable machine-to-machine interaction over a network.
- It has an interface described in a machine-processable format (specifically WSDL).
- Other systems interact with the Web service using SOAP messages.

What is a Web Service?

- Web Services are Classes/Methods, NOT Servlets
- Loosely-coupled
- Encapsulate functionality to logical entities
- Reuse of code and functionality
- Distributed Architecture
- Standardized interface & established Internet protocols

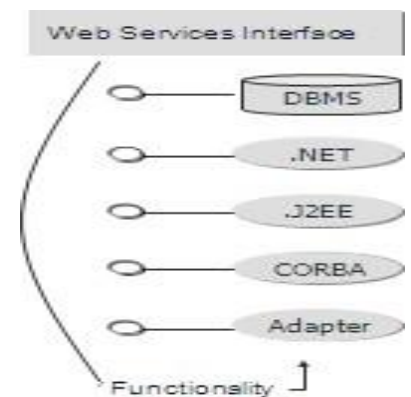


Figure 2.1 Service Interfaces

Characteristics of a Web Service

- ⊠ A web service interface generally consists of a collection of operations that can be used by a client over the Internet.
- ⊠ The operations in a web service may be provided by a variety of different resources, for example, programs, objects, or databases.
- ⊠ The key characteristic of (most) web services is that they can process XML-formatted SOAP messages. An alternative is the REST approach.
- ⊠ Each web service uses its own service description to deal with the service-specific characteristics of the messages it receives. Commercial examples include Amazon, Yahoo, Google and eBay.

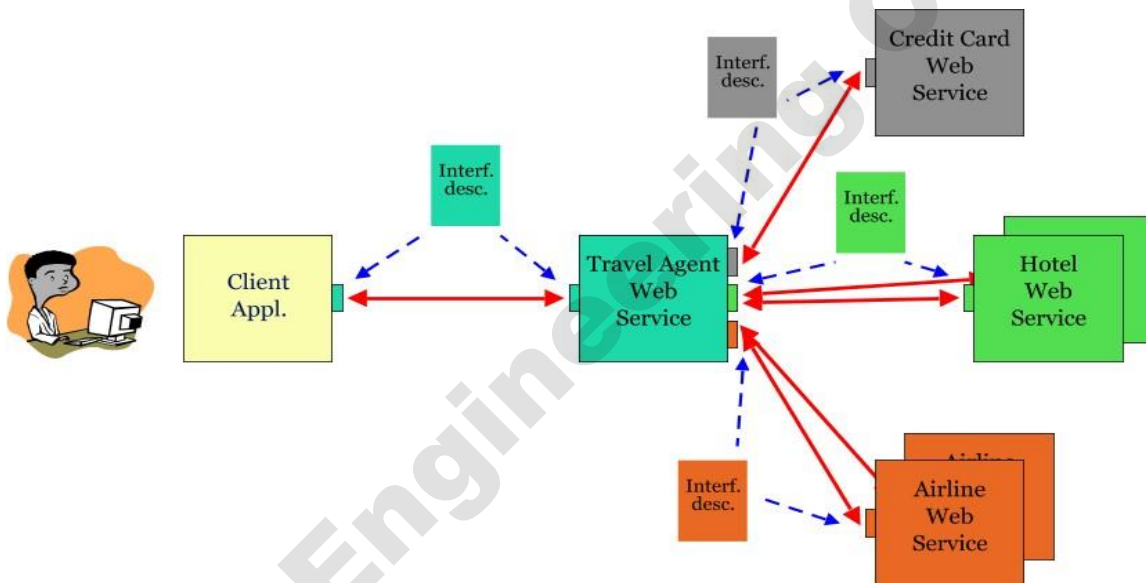


Figure 2.2 Example-Travel Agent Service

Remote Access

- ⊠ IP Address - Locate a Computer
- ⊠ URI - Uniform Resource Identifier
 - Locate a file in that Computer
- ⊠ Socket and Port- Binding to a Method

The Architecture of a Web Service

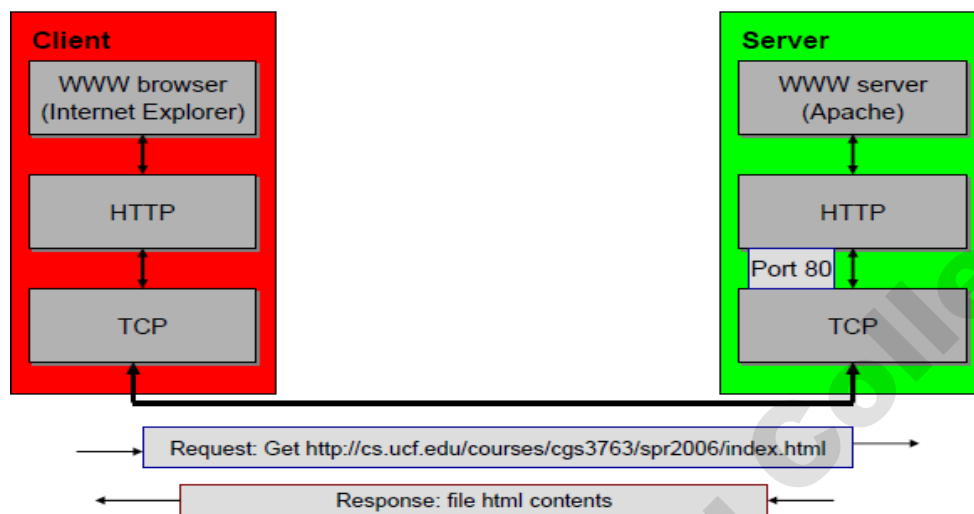
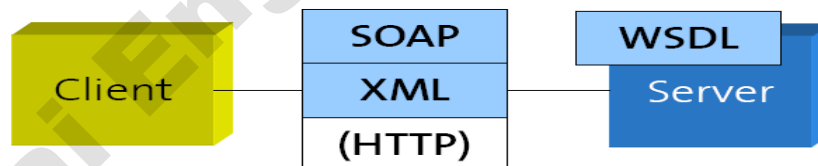


Figure 2.3 Web Service Architecture

Web Sites (1992)



WS-* Web Services (2000)



SOA – Service Oriented Architecture

- ☒ Service provider publishes service description (WSDL), e.g. on a service broker
- ☒ Service Requester finds service (on service broker) and dynamically binds to service
- ☒ Enables ad-hoc collaboration and Enterprise Application Integration (EAI) within web-based information systems
- ☒ SOA is about how to design a software system that makes use of services of new or legacy applications through their published or discoverable interfaces.

- ☒ These applications are often distributed over the networks.
- ☒ SOA also aims to make service interoperability extensible and effective.
- ☒ It prompts architecture styles such as loose coupling, published interfaces and a standard communication model in order to support this goal.

Properties of SOA

- ☒ Logical view
- ☒ Message orientation
- ☒ Description orientation

Logical view

- ☒ The SOA is an abstracted, logical view of actual programs, databases, business processes.
- ☒ Defined in terms of what it does, typically carrying out a business-level operation.
- ☒ The service is formally defined in terms of the messages exchanged between provider agents and requester agents.

Message Orientation

- ☒ The internal structure of providers and requesters include the implementation language, process structure, and even database structure.
- ☒ These features are deliberately abstracted away in the SOA
- ☒ Using the SOA discipline one does not and should not need to know how an agent implementing a service is constructed.
- ☒ The key benefit of this concerns legacy systems.
- ☒ By avoiding any knowledge of the internal structure of an agent, one can incorporate any software component or application to adhere to the formal service definition.

Description orientation

- ☒ A service is described by machine-executable metadata.
- ☒ The description supports the public nature of the SOA.
- ☒ Only those details that are exposed to the public and are important for the use of the service should be included in the description.

- ❑ The semantics of a service should be documented, either directly or indirectly, by its description.

SOA Realization (Two ways)

- ❑ XML - SOAP Based Web Services
- ❑ Extensible Markup Language (XML) is a markup language designed as a standard way to encode documents and data
- ❑ SOAP is an acronym for Simple Object Access Protocol. It is an XML-based messaging protocol for exchanging information among computers
- ❑ RESTful Web services

Web service is the terminology used everywhere

Service Oriented Architecture model implemented by XML Web Services

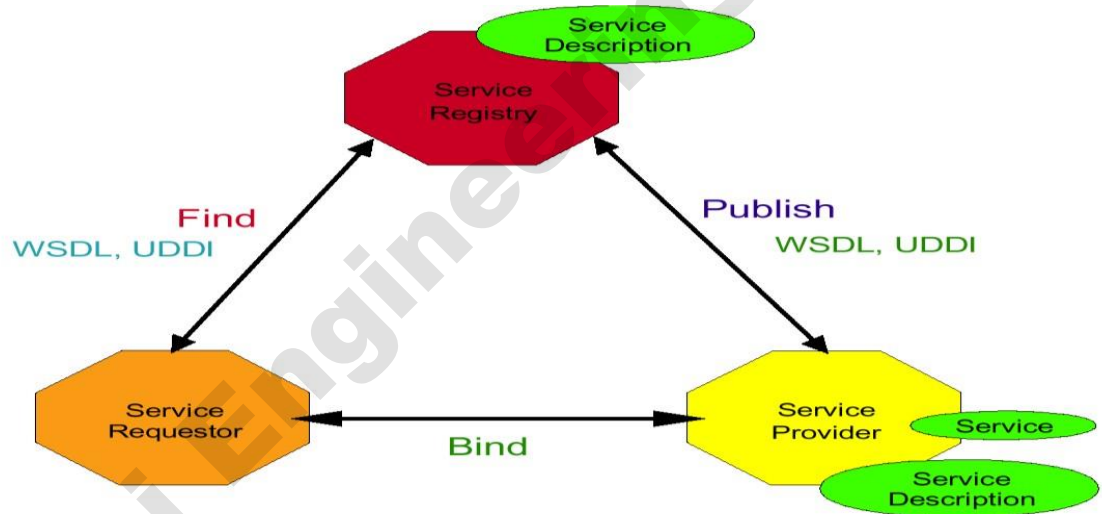


Figure 2.4 SOA Model

WSDL – Web Services Description Languages

- ❑ Provides a machine-readable description of how the service can be called, what parameters it expects, and what data structures it returns.
- ❑ Used in combination with SOAP and an XML Schema to provide Web services over the Internet.

UDDI – Universal Description, Discovery and Integration

- ❑ White Pages — address, contact, and known identifiers;

- ☒ Yellow Pages — industrial categorizations based on standard taxonomies;
- ☒ Green Pages — technical information about services exposed by the business.

Representational State Transfer (REST)

- ☒ APIs are for software components; a way for software to interact with other software.
- ☒ Web Services are a set of rules and technologies that enable two or more components on the web to talk to each other.
- ☒ Not every API is a web service.
- ☒ REST API is a web service.
- ☒ REST API is an API that follows the rules of REST specification.
- ☒ A web service is defined by rules:
 - a) How software components will talk?
 - b) What kind of messages they will send to each other?
 - c) How requests and responses will be handled?

HTTP and REST

- ☒ HTTP is an application layer protocol for sending and receiving messages over a network.
- ☒ REST is a specification that dictates how distributed systems on the web should communicate.
- ☒ REST is a way to implement and use the HTTP protocol.

REST and Systems of Systems

- ☒ SOA focuses on loosely coupled software applications running across different administrative domains, based on common protocols and technologies, such as HTTP and XML.
- ☒ SOA is related to early efforts on the architecture style of large scale distributed systems, particularly Representational State Transfer (REST).
- ☒ REST still provides an alternative to the complex standard-driven web services technology.
- ☒ Used in many Web 2.0 services.

- ❑ REST is a software architecture style for distributed systems, particularly distributed hypermedia systems, such as the World Wide Web.

Applications:

Google, Amazon, Yahoo, Facebook and Twitter

Advantage:

- ❑ Simplicity
- ❑ Ease of being published and consumed by clients.

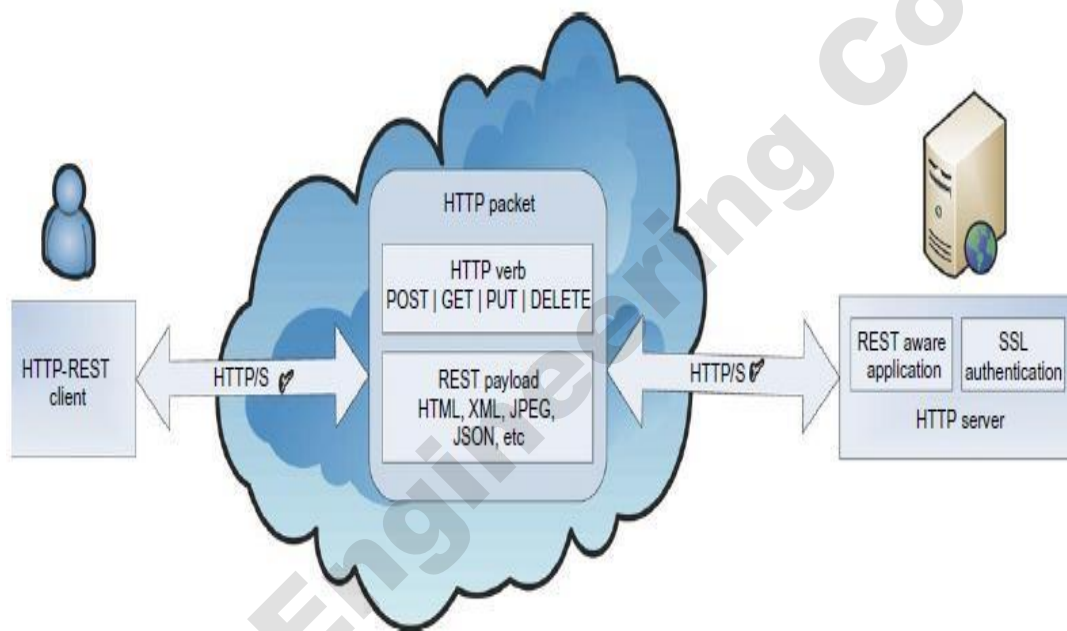


Figure 2.5 A simple REST interaction between user and server in HTTP specification

REST Principles

The REST architectural style is based on four principles:

- ❑ **Resource Identification through URIs**
- ❑ **Uniform, Constrained Interface**
- ❑ **Self-Descriptive Message**
- ❑ **Stateless Interactions**

Resource Identification through URIs

- ⊠ The RESTful web service exposes a set of resources which identify targets of interaction with its clients.
- ⊠ The key abstraction of information in REST is a resource.
- ⊠ Any information that can be named can be a resource, such as a document or image or a temporal service.
- ⊠ A resource is a conceptual mapping to a set of entities.
- ⊠ Each particular resource is identified by a unique name, or more precisely, a Uniform Resource Identifier (URI).
- ⊠ URI is of type URL, providing a global addressing space for resources involved in an interaction between components as well as facilitating service discovery.
- ⊠ The URIs can be bookmarked or exchanged via hyperlinks.
- ⊠ URIs provide more readability and the potential for advertisement.

Uniform, Constrained Interface

- ⊠ Interaction with RESTful web services is done via the HTTP standard, client/server cacheable protocol.
- ⊠ Resources are manipulated using a fixed set of four CRUD

(create, read, update, delete) verbs or operations:

- ⊠ PUT
- ⊠ GET
- ⊠ POST
- ⊠ DELETE

- ⊠ **PUT** creates a new resource.
- ⊠ The resource can then be destroyed by using **DELETE**.
- ⊠ **GET** retrieves the current state of a resource.
- ⊠ **POST** transfers a new state onto a resource.

Self-Descriptive Message

- ⊠ A REST message includes enough information to describe how to process the message.
- ⊠ This enables intermediaries to do more with the message without parsing the message contents.
- ⊠ In REST, resources are decoupled from their representation so that their content can be accessed in a variety of standard formats

Eg:- HTML, XML, MIME, plain text, PDF, JPEG, JSON, etc.

- ⊠ REST provides multiple/alternate representations of each resource.
- ⊠ Metadata about the resource is available and can be used for various purposes.
 - ❖ Cache control
 - ❖ Transmission error detection
 - ❖ Authentication or authorization
 - ❖ Access control.

Stateless Interactions

- ⊠ The REST interactions are “stateless”
- ⊠ Message does not depend on the state of the conversation.
- ⊠ Stateless communications improve visibility, reliability and increases scalability
- ⊠ Decrease network performance by increasing the repetitive data

REST - Advantages

- ⊠ RESTful web services can be considered an alternative to SOAP stack or “big web services
- ⊠ Simplicity
- ⊠ Lightweight nature
- ⊠ Integration with HTTP

REST Architectural Elements

REST Elements	Elements	Example
Data elements	Resource	The intended conceptual target of a hypertext reference
	Resource identifier	URL
	Representation	HTML document, JPEG image, XML, etc.
	Representation metadata	Media type, last-modified time
	Resource metadata	Source link, alternates, vary
	Control data	If-modified-since, cache-control
Connectors	Client	libwww, libwww-perl
	Server	libwww, Apache API, NSAPI
	Cache	Browser cache, Akamai cache network
	Resolver	Bind (DNS lookup library)
	Tunnel	SSL after HTTP CONNECT
Components	Origin server	Apache httpd, Microsoft IIS
	Gateway	Squid, CGI, Reverse Proxy
	Proxy	CERN Proxy, Netscape Proxy, Gauntlet
	User agent	Netscape Navigator, Lynx, MOMspider

Web Services

- ⊠ Web service is often referred to a self-contained, self-describing, modular application designed to be used and accessible by other software applications across the web.
- ⊠ This allows client software to dynamically determine what a service does, the data types that a service uses, how to invoke operations on the service, and the responses that the service may return.
- ⊠ Once a web service is deployed, other applications and other web services can discover and invoke the deployed service.
- ⊠ A web service is defined as a software system designed to support interoperable machine-to-machine interaction over a network
- ⊠ A web service has an interface described in a machine-executable format (specifically Web Services Description Language or WSDL).
- ⊠ Web services are remotely executed, they do not depend on resources residing on the client system that calls them.

- Other systems interact with the web service in a manner prescribed by its description using SOAP messages, typically conveyed using HTTP with an XML serialization

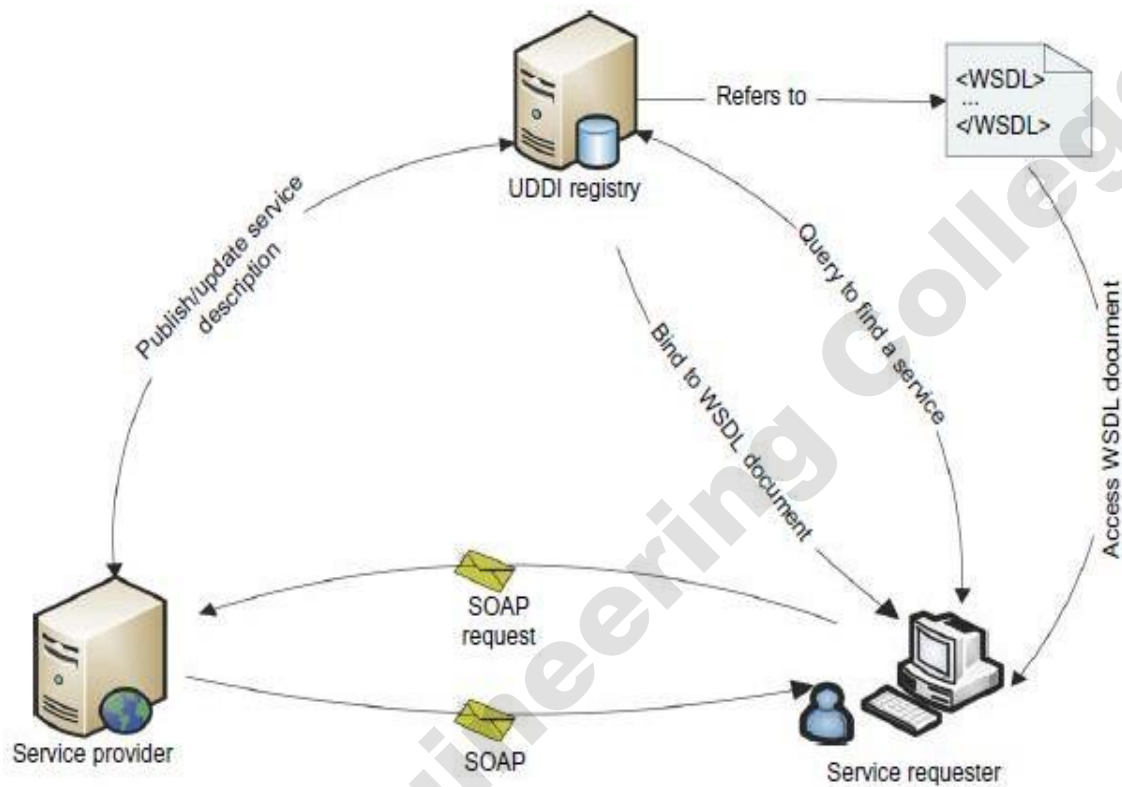


Figure 2.6 Web Services

Web Services- Technologies

- Simple Object Access Protocol (SOAP)
- Web Services Description Language (WSDL)
- Universal Description, Discovery and Integration (UDDI)

Simple Object Access Protocol (SOAP)

- SOAP provides a standard packaging structure for transmission of XML documents over various Internet protocols, such as SMTP, HTTP, and FTP.
- A SOAP message consists of a root element called envelope. Envelope contains a header: a container that can be extended by intermediaries.

- Additional application-level elements such as routing information, authentication, transaction management, message parsing instructions and Quality of Service (QoS) configurations are also included.
- Body element that carries the payload of the message.
- The content of the payload will be marshaled by the sender's SOAP engine and unmarshaled at the receiver side, based on the XML schema that describes the structure of the SOAP message.

Web Services Description Language (WSDL)

WSDL describes the interface, a set of operations supported by a web service in a standard format.

- It standardizes the representation of input and output parameters of its operations as well as the service's protocol binding, the way in which the messages will be transferred on the wire.
- Using WSDL enables disparate clients to automatically understand how to interact with a web service.

Universal Description, Discovery, and Integration (UDDI)

- UDDI provides a global registry for advertising and discovery of web services.
- Performed by searching for names, identifiers, categories or the specification implemented by the web service

SOAP is an extension, and an evolved version of XML-RPC.

A simple and effective remote procedure call protocol which uses XML for encoding its calls and HTTP as a transport mechanism.

- ☒ A procedure executed on the server and the value it returns was formatted in XML.
- ☒ SOAP mainly describes the protocols between interacting parties
- ☒ Data format of exchanging messages is left to XML schema to be handled.



Figure 2.7 WS-I Protocol Stack

- ❑ **Business Process Execution Language for Web Services (BPEL4WS)**, a standard executable language for specifying interactions between web services.
 - ❑ Web services can be composed together to make more complex web services and workflows.
 - ❑ BPEL4WS is an XML-based language, built on top of web service specifications, which is used to define and manage long-lived service orchestrations or processes.
 - ❑ In BPEL, a business process is a large-grained stateful service, which executes steps to complete a business goal.
 - ❑ That goal can be the completion of a business transaction, or fulfillment of the job of a service.
- ❑ **Web Service WS-Notification** enables web services to use the publish and subscribe messaging pattern.
- ❑ **Web Services Security (WS-Security)** are set of protocols that ensure security for SOAP-based messages by implementing the principles of confidentiality, integrity and authentication.

- ❑ **Web Services Reliable Messaging (WS-Reliable Messaging)** describes a protocol that allows **messages** to be delivered reliably between distributed applications in the presence of software component, system, or network failures
- ❑ **WS-ResourceLifetime** specification standardizes the means by which a WS-Resource can be destroyed.
- ❑ **WS-Policy** is a specification that allows web services to use XML to advertise their **policies** (on security, quality of service, etc.) and for web service consumers to specify their **policy** requirements.
- ❑ **WS-ResourceProperties** defines a standard set of message exchanges that allow a requestor to query or update the property values of the WS-Resource.
- ❑ **WS-Addressing** is a specification of transport-neutral mechanism that allows web services to communicate addressing information.
- ❑ **WS-Transaction** **WS-Transaction** is a specification developed that indicates how transactions will be handled and controlled in Web Services.
- ❑ The transaction specification is divided into two parts - short atomic transactions (AT) and long business activity (BA).
- ❑ **Web Services Coordination (WS-Coordination)** describes an extensible framework for providing protocols that coordinate the actions of distributed applications.
- ❑ **The Java Message Service (JMS) API** is a messaging standard that allows application components based on the Java Platform Enterprise Edition (Java EE) to create, send, receive, and read messages.
- ❑ **Internet Inter-ORB Protocol (IIOP)** is an object-oriented protocol
- ❑ Used to facilitate network interaction between distributed programs written in different programming languages.
- ❑ IIOP is used to enhance Internet and intranet communication for applications and services.

SOAP Request	SOAP Response
<pre> <soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap- envelope" soap:encodingStyle= "http://www.w3.org/2001/12/soap-encoding"> <soap:Body> <CreateBucket xmlns="http://doc.s3.amazonaws .com/2010-03-15"> <Bucket>SampleBucket</Bucket> <AWSAccessKeyId> 1B9FVRAYCP1VJEXAMPLE= </AWSAccessKeyId> <Timestamp>2010-03-15T14:40:00.165Z </Timestamp> <Signature>luyz3d3P0aTou39dzbqaEXAMPLE =</Signature> </CreateBucket> </soap:Body> </soap:Envelope> </pre>	<pre> <soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap- envelope" soap:encodingStyle= "http://www.w3.org/2001/12/soap-encoding"> <soap:Body> <CreateBucket xmlns="http://doc.s3.amazonaws .com/2010-03-15"> <Bucket>SampleBucket</Bucket> <AWSAccessKeyId>1B9FVRAYCP1VJEXAMPLE= </AWSAccessKeyId> <Timestamp>2010-03-15T14:40:00.165Z </Timestamp> <Signature>luyz3d3P0aTou39dzbqaEXAMPLE =</Signature> </CreateBucket> </soap:Body> </soap:Envelope> </pre>

- ❑ A SOAP message consists of an envelope used by the applications to enclose information that need to be sent.
- ❑ An envelope contains a header and a body block.
- ❑ The EncodingStyle element refers to the URI address of an XML schema for encoding elements of the message.
- ❑ Each element of a SOAP message may have a different encoding, but unless specified, the encoding of the whole message is as defined in the XML schema of the root element.
- ❑ The header is an optional part of a SOAP message that may contain auxiliary information.
- ❑ The body of a SOAP request-response message contains the main information of the conversation, formatted in one or more XML blocks.
- ❑ In example, the client is calling CreateBucket of the Amazon S3 web service interface.
- ❑ In case of an error in service invocation, a SOAP message including a Fault element in the body.

Table 5.4 The 10 Areas Covered by the Core WS-* Specifications

WS-* Specification Area	Examples
1. Core Service Model	XML, WSDL, SOAP
2. Service Internet	WS-Addressing, WS-MessageDelivery, Reliable WSRM, Efficient MOTM
3. Notification	WS-Notification, WS-Eventing (Publish-Subscribe)
4. Workflow and Transactions	BPEL, WS-Choreography, WS-Coordination
5. Security	WS-Security, WS-Trust, WS-Federation, SAML, WS-SecureConversation
6. Service Discovery	UDDI, WS-Discovery
7. System Metadata and State	WSRF, WS-MetadataExchange, WS-Context
8. Management	WSDM, WS-Management, WS-Transfer
9. Policy and Agreements	WS-Policy, WS-Agreement
10. Portals and User Interfaces	WSRP (Remote Portlets)

PUBLISH SUBSCRIBE MODEL

- ☒ Publish/Subscribe systems are nowadays considered a key technology for information diffusion.
- ☒ Each participant in a publish/subscribe communication system can play the role of a publisher or a subscriber of information.
- ☒ Publishers produce information in form of events, which are then consumed by subscribers.
- ☒ Subscribers can declare their interest on a subset of the whole information issuing subscriptions.
- ☒ There are two major roles:
 - ☒ Publisher
 - ☒ Subscriber
- ☒ The former provides facilities for the later to register its interest in a specific topic or event.

- ❑ Specific conditions holding true on the publisher side can trigger the creation of messages that are attached to a specific event.
- ❑ Message will be available to all the subscribers that registered for the corresponding event.
- ❑ There are two major strategies for dispatching the event to the subscribers.

Push strategy:

- ❑ It is the responsibility of the publisher to notify all the subscribers. Eg: Method invocation.

Pull strategy :

- ❑ The publisher simply makes available the message for a specific event.
- ❑ It is the responsibility of the subscribers to check whether there are messages on the events that are registered.
- ❑ Subscriptions are used to filter out part of the events produced by publishers.
- ❑ In Software Architecture, Publish/Subscribe pattern is a message pattern and a network oriented architectural pattern
- ❑ It describes how two different parts of a message passing system connect and communicate with each other.
- ❑ There are three main components to the Publish Subscribe Model:
 - ❑ Publishers
 - ❑ Eventbus/broker
 - ❑ Subscribers

Publishers:

- ❑ Broadcast messages, with no knowledge of the subscribers.

Subscribers:

- ❑ They 'listen' out for messages regarding topic/categories that they are interested in without any knowledge of who the publishers are.

Event Bus:

- ❑ Transfers the messages from the publishers to the subscribers.

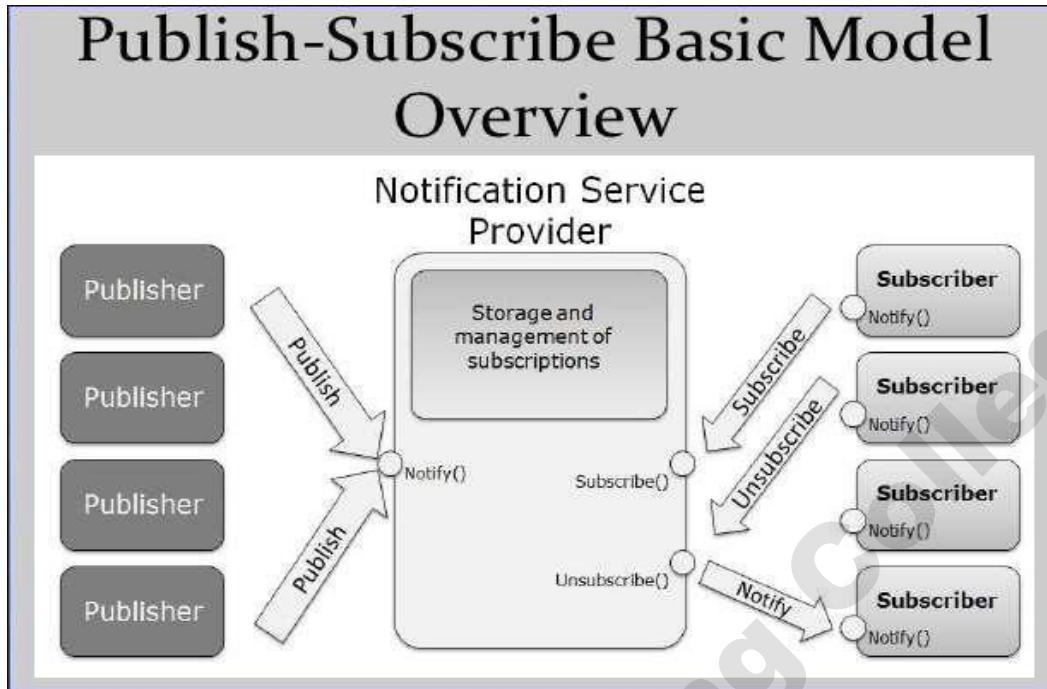


Figure 2.8 Publish Subscribe Model

- ❑ Each subscriber only receives a subset of the messages that have been sent by the Publisher.
- ❑ Receive the message topics or categories they have subscribed to.
- ❑ There are two methods of filtering out unrequired messages:
 - Topic based filter
 - Content based filter

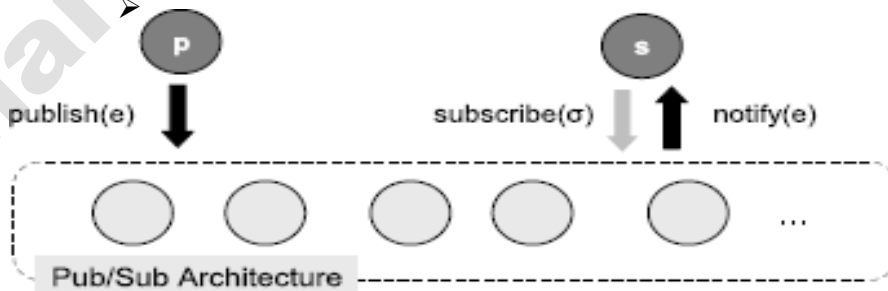


Figure 2.9 High Level View of A Publish/Subscribe System

- ❑ A generic pub/sub communication system is often referred as Event Service or Notification Service.

- ❑ System composed of a set of nodes distributed over a communication network.
- ❑ The clients of this system are divided according to their role into publishers and subscribers.
- ❑ Clients are not required to communicate directly among themselves.
- ❑ The interaction takes place through the nodes of the pub/sub system.

Elements of a Publish/Subscribe System

- ❑ A publisher submits a piece of information e (i.e., an event) to the pub/sub system by executing the `publish(e)` operation.
- ❑ An event is structured as a set of attribute-value pairs.
- ❑ Each attribute has a *name*, a *simple character* string, and a *type*.
- ❑ The type is generally one of the common primitive data types defined in programming languages or query languages (e.g. integer, real, string, etc.).
- ❑ On the subscriber's side, interest in specific events is expressed through subscriptions.
- ❑ A subscription is a filter over a portion of the event content (or the whole of it).
- ❑ Expressed through a set of constraints that depend on the subscription language.
- ❑ A subscriber installs and removes a subscription from the pub/sub system by executing the `subscribe()` and `unsubscribe()` operations respectively.
- ❑ An event e matches a subscription if it satisfies all the declared constraints on the corresponding attributes.
- ❑ The task of verifying whenever an event e matches a subscription is called matching.

Semantics of a Publish/subscribe System

- ❑ When a process issues a subscribe/unsubscribe operation, the pub/sub system is not immediately aware of the occurred event.
- ❑ The registration (resp. cancellation) of a subscription takes a certain amount of time, denoted as T_{sub} , to be stored into the system.

This time encompass the update of the internal data structures of the pub/sub system and the network delay due to the routing of the subscription.

Three properties:

- Safety (Legality): A subscriber cannot be notified for an information it is not interested in.
- Safety (Validity): A subscriber cannot be notified for an event that has not been previously published.
- Liveness: The delivery of a notification for an event is guaranteed only for those subscribers that subscribed at a time at least T_{sub} before the event was published.

Quality of Service in Publish/Subscribe Systems

- Reliable delivery
- Timeliness
- Security and trust

Reliable delivery

- ⊠ Reliable delivery of an event means determining the subscribers that have to receive a published event, as stated by the liveness property and delivering the event to all of them.

Timeliness

- ⊠ Real-time applications often require strict control over the time elapsed by a piece of information to reach all its consumers.
- ⊠ They are typically deployed over dedicated infrastructures or simply managed environments where synchronous message delivery can be safely assumed.

Security and trust

- ⊠ A subscriber wants to trust authenticity of the events it receives from the system.
- ⊠ Generated by a trusty publisher and the information they contains have not been corrupted.
- ⊠ Subscribers have to be trusted for what concerns the subscriptions they issue.
- ⊠ Since an event is in general delivered to several subscribers, the producer/consumer trust relationship that commonly occur in a point-to-point communication, in pub/sub system must involve multiple participants

Subscription Models

- Topic based Model
- Type based Model
- Concept based Model
- Content based Model

Topic-based Model

- ⊠ Events are grouped in topics.
- ⊠ A subscriber declares its interest for a particular topic to receive all events pertaining to that topic.
- ⊠ Each topic corresponds to a logical channel ideally connecting each possible publisher to all interested subscribers.
- ⊠ Requires the messages to be broadcasted into logical channels.
- ⊠ Subscribers only receive messages from logic channels they care about (and have subscribed to).

Type based Model

- ⊠ Pub/sub variant events are actually objects belonging to a specific type, which can thus encapsulate attributes as well as methods.
- ⊠ Types represent a more robust data model for application developer.
- ⊠ Enforce type-safety at the pub/sub system, rather than inside the application.
- ⊠ The declaration of a desired type is the main discriminating attribute.

Concept based Model

- ⊠ Allows to describe event schema at a higher level of abstraction by using ontologies.
- ⊠ Provide a knowledge base for an unambiguous interpretation of the event structure, by using metadata and mapping functions.

Content based Model

- ⊠ System allows subscribers to receive messages based on the content of the messages.

Subscribers themselves must sort out junk messages from the ones they want.

Benefits

Loose coupling

- ☒ The publisher is not aware of the number of subscribers, of the identities of the subscribers, or of the message types that the subscribers are subscribed to.

Improved security

- ☒ The communication infrastructure transports the published messages only to the applications that are subscribed to the corresponding topic.
- ☒ Specific applications can exchange messages directly, excluding other applications from the message exchange.

Improved testability.

- ☒ Topics usually reduce the number of messages that are required for testing.

Separation of concerns

- ☒ Due to the simplistic nature of the architecture, developers can exercise fine grained separation of concerns by dividing up message types to serve a single simple purpose each.
- ☒ Eg. data with a topic “/cats” should only contain information about cats.

Reduced cognitive load for subscribers

- ☒ Subscribers need not concern themselves with the inner workings of a publisher.
- ☒ They do not even have to access to the source code.
- ☒ Subscribers only interact with the publisher through the public API exposed by the publisher.

Drawbacks

Increased complexity.

Publish/Subscribe requires you to address the following:

- ☒ To design a message classification scheme for topic implementation.
- ☒ To implement the subscription mechanism.
- ☒ To modify the publisher and the subscribers.

Increased maintenance effort.

- ☒ Managing topics requires maintenance work.
- ☒ Organizations that maintain many topics usually have formal procedures for their use.

Decreased performance

- ☒ Subscription management adds overhead.
- ☒ This overhead increases the latency of message exchange, and this latency decreases performance.

Inflexibility of data sent by publisher

- ☒ The publish/subscribe model introduces high semantic coupling in the messages passed by the publishers to the subscribers.
- ☒ Once the structure of the data is established, it becomes difficult to change.
- ☒ In order to change the structure of the messages, all of the subscribers must be altered to accept the changed format

Instability of Delivery

- ☒ The publisher does not have perfect knowledge of the status of the systems listening to the messages.
- ☒ For instance, publish/subscribe is commonly used for logging systems.
- ☒ If a logger subscribing to the 'Critical' message type crashes or gets stuck in an error state, then the 'Critical' messages may be lost!
- ☒ Then any services depending on the error messages will be unaware of the problems with the publisher.

Applications

Used in a wide range of group communication applications including

- Software Distribution
- Internet TV
- Audio or Video-conferencing
- Virtual Classroom

- Multi-party Network Games
- Distributed Cache Update

It can also be used in even larger size group communication applications, such as broadcasting and content distribution.

- News and Sports Ticker Services
- Real-time Stock Quotes and Updates
- Market Tracker
- Popular Internet Radio Sites

VIRTUALIZATION

- Virtualization is a technique, which allows sharing single physical instance of an application or resource among multiple organizations or tenants (customers).
- Virtualization is a proved technology that makes it possible to run multiple operating system and applications on the same server at same time.
- Virtualization is the process of creating a logical(virtual) version of a server operating system, a storage device, or network services.
- The technology that work behind virtualization is known as a virtual machine monitor(VM), or virtual manager which separates compute environments from the actual physical infrastructure.
- Virtualization -- the abstraction of computer resources.
- Virtualization hides the physical characteristics of computing resources from their users, applications, or end users.
- This includes making a single physical resource (such as a server, an operating system, an application, or storage device) appear to function as multiple virtual resources.
- It can also include making multiple physical resources (such as storage devices or servers) appear as a single virtual resource.
- In computing, virtualization refers to the act of creating a virtual (rather than actual) version of something, like computer hardware platforms, operating systems, storage devices, and computer network resources

- Creation of a virtual machine over existing operating system and hardware.
- **Host machine:** The machine on which the virtual machine is created.
- **Guest machine:** virtual machines referred as a guest machine.
- **Hypervisor:** Hypervisor is a firmware or low-level program that acts as a Virtual Machine Manager.

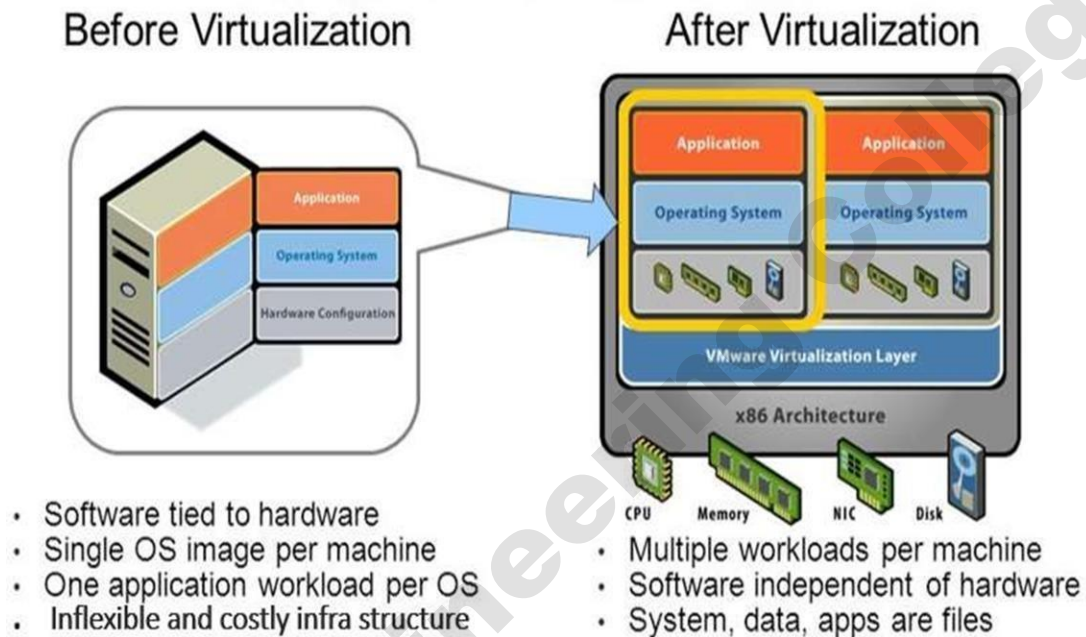


Figure 2.10 Virtualization Example

Advantages of Virtualization:

1. Reduced Costs.
2. Efficient hardware Utilization.
3. Virtualization leads to better resource Utilization and increase performance
4. Testing for software development.
5. Increase Availability
6. Save energy
7. Shifting all your Local Infrastructure to Cloud in a day
8. Possibility to Divide Services
9. Running application not supported by the host.

Disadvantages of Virtualization:

1. Extra Costs.
2. Software Licensing.

IMPLEMENTATION LEVELS OF VIRTUALIZATION

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. For example, computer users gained access to much enlarged memory space when the concept of virtual memory was introduced. Similarly, virtualization techniques can be applied to enhance the use of compute engines, networks and storage.

Levels of Virtualization:

A traditional computer runs with host operating system specially tailored for its hardware architecture, as shown in Figure 2.11 (a). After virtualization, different user applications managed by their own operating systems (guest OS) can run on the same hardware, independent of the host OS.

This is often done by adding additional software, called a virtualization layer as shown in Figure 2.11 (b). This virtualization layer is known as hypervisor or virtual machine monitor (VMM). The VMs are shown in the upper boxes, where applications run with their own guest OS over the virtualized CPU, memory, and I/O resources. The main function of the software layer for virtualization is to virtualize the physical hardware of a host machine into virtual resources to be used by the VMs, exclusively. The virtualization software creates the abstraction of VMs by interposing a virtualization layer at various levels of a computer system. Common virtualization layers include the instruction set architecture (ISA) level, hardware level, operating system level, library support level, and application level.

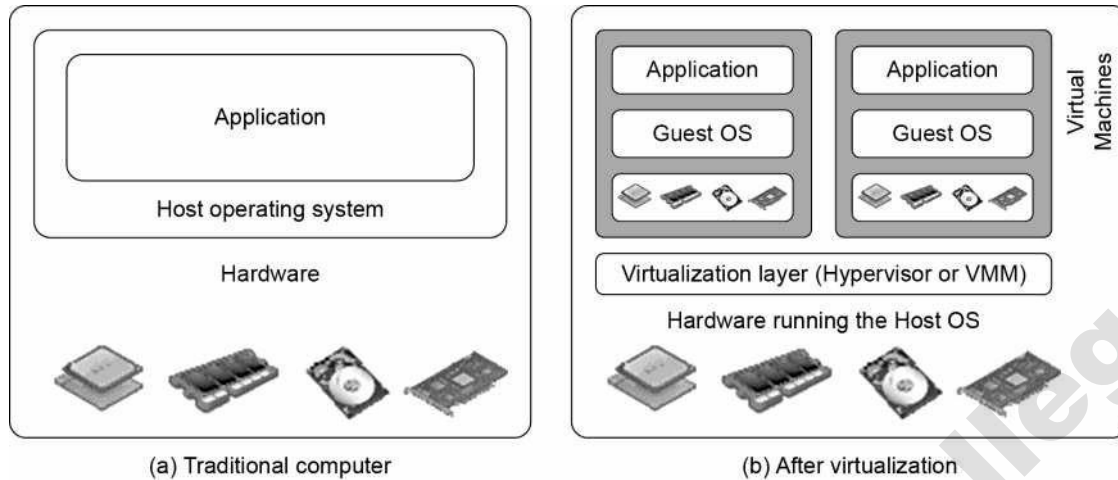


Figure 2.11 The architecture of a computer system before and after Virtualization

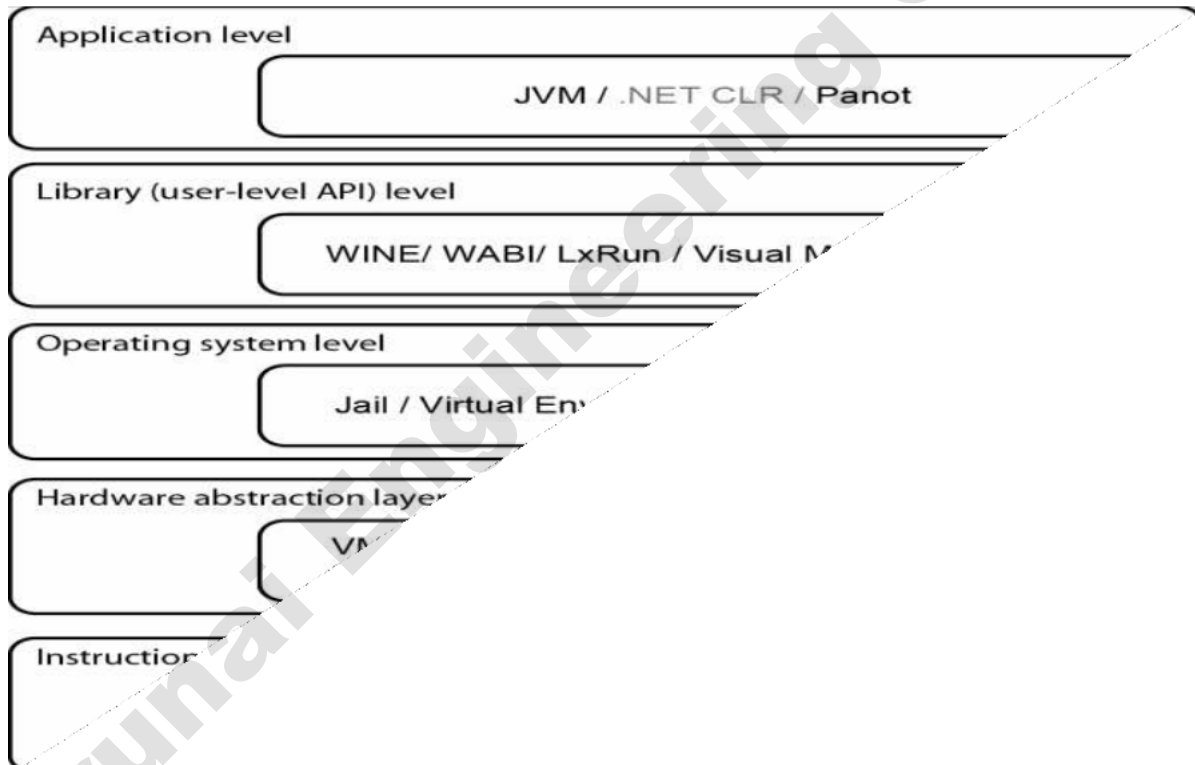


Figure 2.12 Virtualization ranging from hardware to applications in five abstraction levels.

Instruction Set Architecture Level:

At the ISA level, virtualization is performed by emulating a given ISA by the ISA of the host machine. For example, MIPS binary code can run on an x86-based host machine with the help of ISA emulation. With this approach, it is possible to run a large amount of legacy binary

code written for various processors on any given new hardware host machine. Instruction set emulation leads to virtual ISAs created on any hardware machine.

The basic emulation method is through code interpretation. An interpreter program interprets the source instructions to target instructions one by one. OneSource instruction may require tens or hundreds of native target instructions to perform its function. Obviously, this process is relatively slow. For better performance, dynamic binary translation is desired.

This approach translates basic blocks of dynamic source instructions to target instructions. The basic blocks can also be extended to program traces or super blocks to increase translation efficiency. Instruction set emulation requires binary translation and optimization. A virtual instruction set architecture (V-ISA) thus requires adding a processor-specific software translation layer to the compiler.

Hardware Abstraction Level:

Hardware-level virtualization is performed right on top of the bare hardware. The idea is to virtualize a computer's resources, such as its processors, memory, and I/O devices. The intention is to upgrade the hardware utilization rate by multiple users concurrently.

Operating System Level:

This refers to an abstraction layer between traditional OS and user applications. OS-level virtualization creates isolated containers on a single physical server and the OS instances to utilize the hardware and software in datacenters.

The containers behave like real servers. OS-level virtualization is commonly used in creating virtual hosting environments to allocate hardware resources among a large number of mutually distrusting users. It is also used, to a lesser extent, in consolidating server hardware by moving services on separate hosts into containers or VMs on one server.

Library Support Level:

Most applications use APIs exported by user level libraries rather than using lengthy system calls by the OS. Since most systems provide well documented APIs, such an interface becomes another candidate for virtualization.

Virtualization with library interfaces is possible by controlling the communication link between applications and the rest of a system through API hooks. The software tool WINE has implemented this approach to support Windows applications on top of UNIX hosts. Another

example is the vCUDA which allows applications executing within VMs to leverage GPU hardware acceleration.

User-Application Level:

Virtualization at the application level virtualizes an application as a VM. On a traditional OS, an application often runs as a process. Therefore, application-level virtualization is also known as process-level virtualization. The most popular approach is to deploy high level language (HLL)VMs.

VMM Design Requirements and Providers

Hardware-level virtualization inserts a layer between real hardware and traditional operating systems. This layer is commonly called the Virtual Machine Monitor (VMM) and it manages the hardware resources of a computing system. Each time programs access the hardware the VMM captures the process. VMM acts as a traditional OS.

One hardware component, such as the CPU, can be virtualized as several virtual copies. Therefore, several traditional operating systems which are the same or different can sit on the same set of hardware simultaneously.

Three requirements for a VMM

- First, a VMM should provide an environment for programs which is essentially identical to the original machine.
- Second, programs run in this environment should show, at worst, only minor decreases in speed.
- Third, a VMM should be in complete control of the system resources

Provider and References	Host CPU	Host OS	Guest OS	Architecture
VMware Workstation [71]	x86, x86-64	Windows, Linux	Windows, Linux, Solaris, FreeBSD, Netware, OS/2, SCO, BeOS, Darwin	Full Virtualization
VMware ESX Server [71]	x86, x86-64	No host OS	The same as VMware Workstation	Para-Virtualization
Xen [7,13,42]	x86, x86-64, IA-64	NetBSD, Linux, Solaris	FreeBSD, NetBSD, Linux, Solaris, Windows XP and 2003 Server	Hypervisor
KVM [31]	x86, x86-64, IA-64, S390, PowerPC	Linux	Linux, Windows, FreeBSD, Solaris	Para-Virtualization

Virtualization Support at the OS Level

With the help of VM technology, a new computing mode known as cloud computing is emerging. Cloud computing is transforming the computing landscape by shifting the hardware and staffing costs of managing a computational center to third parties, just like banks. However, cloud computing has at least two challenges.

- The first is the ability to use a variable number of physical machines and VM instances depending on the needs of a problem.
- The second challenge concerns the slow operation of instantiating new VMs.

Currently, new VMs originate either as fresh boots or as replicates of a template VM, unaware of the current application state. Therefore, to better support cloud computing, a large amount of research and development should be done.

Why OS-Level Virtualization?

To reduce the performance overhead of hardware-level virtualization, even hardware modification is needed. OS-level virtualization provides a feasible solution for these hardware-level virtualization issues. Operating system virtualization inserts a virtualization layer inside an operating system to partition a machine's physical resources. It enables multiple isolated VMs within a single operating system kernel. This kind of VM is often called a virtual execution environment (VE), Virtual Private System (VPS), or simply container. From the user's point of view, VEs look like real servers. This means a VE has its own set of processes, file system, user accounts, network interfaces with IP addresses, routing tables, firewall rules, and other personal settings. Although VEs can be customized for different people, they share the same operating system kernel.

Advantages of OS Extensions

(1) VMs at the operating system level have minimal startup/shutdown costs, low resource requirements, and high scalability.

(2) For an OS-level VM, it is possible for a VM and its host environment to synchronize state changes when necessary.

These benefits can be achieved via two mechanisms of OS-level virtualization:

(1) All OS-level VMs on the same physical machine share a single operating system kernel

(2) The virtualization layer can be designed in a way that allows processes in VMs to access as many resources of the host machine as possible, but never to modify them.

Virtualization on Linux or Windows Platforms

Virtualization support on the Windows-based platform is still in the research stage. The Linux kernel offers an abstraction layer to allow software processes to work with and operate on resources without knowing the hardware details. New hardware may need a new Linux kernel to support. Therefore, different Linux platforms use patched kernels to provide special support for extended functionality.

Virtualization Support and Source of Information	Brief Introduction on Functionality and Application Platforms
Linux vServer for Linux platforms (http://linux-vserver.org/)	Extends Linux kernels to implement a security mechanism to help build VMs by setting resource limits and file attributes and changing the root environment for VM isolation
OpenVZ for Linux platforms [65]; http://ftp.openvz.org/doc/OpenVZ-Users-Guide.pdf	Supports virtualization by creating <i>virtual private servers (VPSes)</i> ; the VPS has its own files, users, process tree, and virtual devices, which can be isolated from other VPSes, and checkpointing and live migration are supported
FVM (Feather-Weight Virtual Machines) for virtualizing the Windows NT platforms [78]	Uses system call interfaces to create VMs at the NT kernel space; multiple VMs are supported by virtualized namespace and copy-on-write

Middleware Support for Virtualization

Library-level virtualization is also known as user-level Application Binary Interface (ABI) or API emulation. This type of virtualization can create execution environments for running alien programs on a platform rather than creating a VM to run the entire operating system. API call interception and remapping are the key functions performed. This provides an overview of several library-level virtualization systems: namely the Windows Application Binary Interface (WABI), Ixrun, WINE, Visual MainWin, and Vcuda.

Table 3.4 Middleware and Library Support for Virtualization

Middleware or Runtime Library and References or Web Link	Brief Introduction and Application
WABI (http://docs.sun.com/app/docs/doc/802-6306)	Middleware that converts applications running on x86 PCs to run on SPARC.
Lxrun (Linux Run) (http://www.ugcs.caltech.edu/~steven/lxrun/)	A system for running Linux on SPARC.
WINE (http://www.winehq.org/)	Windows compatibility layer for Linux.
Visual MainWin (http://www.riverbankcomputing.com/)	Windows compatibility layer for Linux.
vCUDA (Example)	Virtualized CUDA support.

Virtualization Structures/Tools and Mechanisms

There are three typical classes of VM architecture. Before virtualization, the operating system manages the hardware. After virtualization, a virtualization layer is inserted between the hardware and the operating system. In such a case, the virtualization layer is responsible for converting portions of the real hardware into virtual hardware. Therefore, different operating systems such as Linux and Windows can run on the same physical machine, simultaneously.

Depending on the position of the virtualization layer, there are several classes of VM architectures, namely the hypervisor architecture, para-virtualization, and host based virtualization. The hypervisor is also known as the VMM (Virtual Machine Monitor). They both perform the same virtualization operations.

Hypervisor and Xen Architecture:

The hypervisor supports hardware-level virtualization on bare metal devices like CPU, memory, disk and network interfaces. The hypervisor software sits directly between the physical hardware and its OS. This virtualization layer is referred to as either the VMM or the hypervisor. The hypervisor provides hypercalls for the guest OSes and applications. Depending on the functionality, a hypervisor can assume a micro-kernel architecture like the Microsoft Hyper-V. Or it can assume monolithic hypervisor architecture like the VMware ESX for server virtualization.

A micro-kernel hypervisor includes only the basic and unchanging functions (such as physical memory management and processor scheduling). The device drivers and other changeable components are outside the hypervisor. A monolithic hypervisor implements all the aforementioned functions, including those of the device drivers.

Therefore, the size of the hypervisor code of a micro-kernel hypervisor is smaller than that of a monolithic hypervisor. Essentially, a hypervisor must be able to convert physical devices into virtual resources dedicated for the deployed VM to use.

The Xen Architecture:

The core components of a Xen system are the hypervisor, kernel, and applications. The organization of the three components is important. Like other virtualization systems, many guest OSes can run on top of the hypervisor. However, not all guest OSes are created equal, and one in particular controls the others.

The guest OS, which has control ability, is called Domain 0, and the others are called Domain U. Domain 0 is a privileged guest OS of Xen. It is first loaded when Xen boots without any file system drivers being available. Domain 0 is designed to access hardware directly and manage devices. Therefore, one of the responsibilities of Domain 0 is to allocate and map hardware resources for the guest domains (the Domain U domains).

Binary Translation with Full Virtualization:

Depending on implementation technologies, hardware virtualization can be classified into two categories: full virtualization and host-based virtualization. Full virtualization does not need to modify the host OS. It relies on binary translation to trap and to virtualize the execution of certain sensitive, non virtualizable instructions. The guest OSes and their applications consist of noncritical and critical instructions. In a host-based system, both a host OS and a guest OS are used. A virtualization software layer is built between the host OS and guest OS.

Full Virtualization:

With full virtualization, noncritical instructions run on the hardware directly while critical instructions are discovered and replaced with traps into the VMM to be emulated by software. Both the hypervisor and VMM approaches are considered full virtualization.

Binary Translation of Guest OS Requests Using a VMM :

VMware puts the VMM at Ring 0 and the guest OS at Ring 1. The VMM scans the instruction stream and identifies the privileged, control- and behavior-sensitive instructions. When these instructions are identified, they are trapped into the VMM, which emulates the behavior of these instructions.



Figure 2.13 Indirect execution of complex instructions via binary translation of guest OS requests using the VMM plus direct execution of simple instructions on the same host.

The method used in this emulation is called binary translation. Therefore, full virtualization combines binary translation and direct execution. The guest OS is completely decoupled from the underlying hardware. Consequently, the guest OS is unaware that it is being virtualized. Binary translation employs a code cache to store translated hot instructions to improve performance, but it increases the cost of memory usage.

Host-Based Virtualization:

An alternative VM architecture is to install a virtualization layer on top of the host OS. This host OS is still responsible for managing the hardware. The guest OSes are installed and run on top of the virtualization layer. Dedicated applications may run on the VMs. Certainly, some other applications can also run with the host OS directly. This host-based architecture has some distinct advantages, as enumerated next. First, the user can install this VM architecture without modifying the host OS. The virtualizing software can rely on the host OS to provide device drivers and other low level services. This will simplify the VM design and ease its deployment. Second, the host-based approach appeals to many host machine configurations.

Compared to the hypervisor/VMM architecture, the performance of the host based architecture may also be low. When an application requests hardware access, it involves four layers of mapping which downgrades performance significantly.

Para-Virtualization with Compiler Support:

Para-virtualization needs to modify the guest operating systems. A para-virtualized VM provides special APIs requiring substantial OS modifications in user applications. Performance degradation is a critical issue of a virtualized system. No one wants to use a VM if it is much slower than using a physical machine.

The virtualization layer can be inserted at different positions in a machine software stack. However, para-virtualization attempts to reduce the virtualization overhead, and thus improve performance by modifying only the guest OS kernel. The guest operating systems are para-virtualized. The traditional x86 processor offers four instruction execution rings: Rings 0,1, 2, and 3. The lower the ring number, the higher the privilege of instruction being executed. The OS is responsible for managing the hardware and the privileged instructions to execute at Ring 0, while user-level applications run at Ring 3.

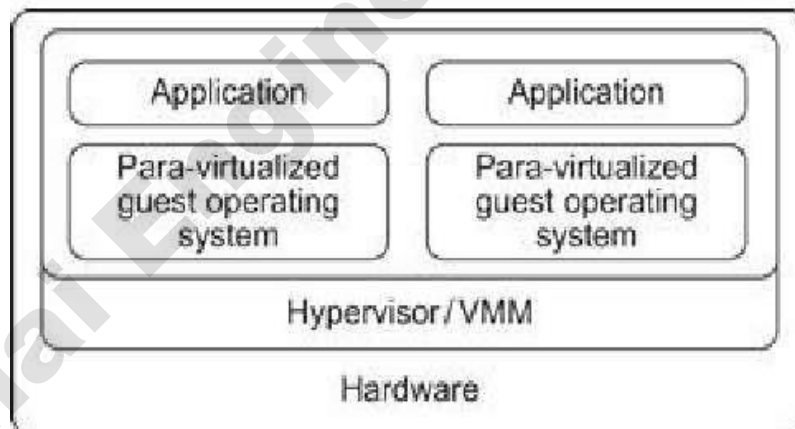


Figure 2.14 Para-virtualized VM architecture

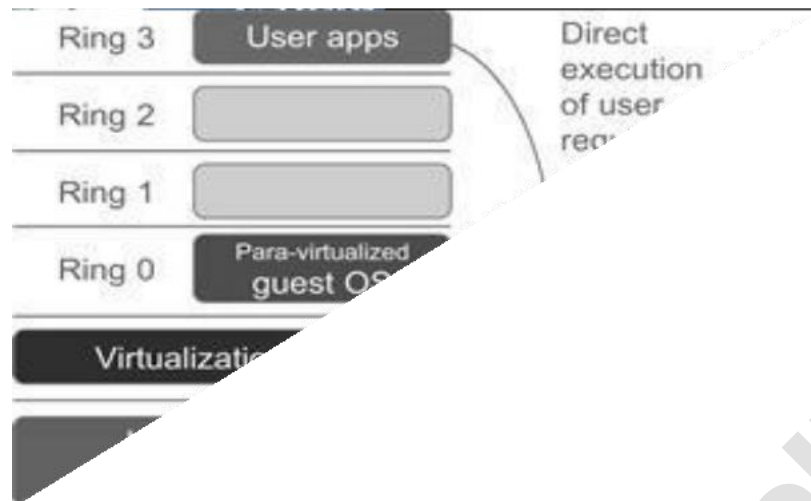


Figure 2.15 The use of a para-virtualized guest OS assisted by an intelligent compiler to replace non virtualizable OS instructions by hyper calls.

Para-Virtualization Architecture:

When the x86 processor is virtualized, a virtualization layer is inserted between the hardware and the OS. According to the x86 ring definitions, the virtualization layer should also be installed at Ring 0. The para-virtualization replaces non virtualizable instructions with hyper calls that communicate directly with the hypervisor or VMM. However, when the guest OS kernel is modified for virtualization, it can no longer run on the hardware directly.

Although para-virtualization reduces the overhead, it has incurred other problems. First, its compatibility and portability may be in doubt, because it must support the unmodified OS as well. Second, the cost of maintaining para-virtualized OSes is high, because they may require deep OS kernel modifications. Finally, the performance advantage of para virtualization varies greatly due to workload variations.

KVM (Kernel-Based VM):

This is a Linux para-virtualization system—a part of the Linux version 2.6.20 kernel. Memory management and scheduling activities are carried out by the existing Linux kernel. The KVM does the rest, which makes it simpler than the hypervisor that controls the entire machine. KVM is a hardware-assisted para-virtualization tool, which improves performance and supports unmodified guest OSes such as Windows, Linux, Solaris, and other UNIX variants. Unlike the

full virtualization architecture which intercepts and emulates privileged and sensitive instructions at runtime, para-virtualization handles these instructions at compile time.

The guest OS kernel is modified to replace the privileged and sensitive instructions with hyper calls to the hypervisor or VMM. Xen assumes such a para virtualization architecture. The guest OS running in a guest domain may run at Ring 1 instead of at Ring 0. This implies that the guest OS may not be able to execute some privileged and sensitive instructions. The privileged instructions are implemented by hypercalls to the hypervisor. After replacing the instructions with hyper calls, the modified guest OS emulates the behavior of the original guest OS.

VIRTUALIZATION OF CPU, MEMORY, AND I/O DEVICES

To support virtualization, processors such as the x86 employ a special running mode and instructions, known as hardware-assisted virtualization. In this way, the VMM and guest OS run in different modes and all sensitive instructions of the guest OS and its applications are trapped in the VMM. To save processor states, modes switching are completed by hardware. For the x86 architecture, Intel and AMD have proprietary technologies for hardware-assisted virtualization.

Hardware Support for Virtualization: Modern operating systems and processors permit multiple processes to run simultaneously. If there is no protection mechanism in a processor, all instructions from different processes will access the hardware directly and cause a system crash. Therefore, all processors have at least two modes, user mode and supervisor mode, to ensure controlled access of critical hardware. Instructions running in supervisor mode are called privileged instructions.

Other instructions are unprivileged instructions. In a virtualized environment, it is more difficult to make OSes and applications run correctly because there are more layers in the machine stack.

CPU Virtualization:

A VM is a duplicate of an existing computer system in which a majority of the VM instructions are executed on the host processor in native mode. Thus, unprivileged instructions of VMs run directly on the host machine for higher efficiency. Other critical instructions should be handled carefully for correctness and stability. The critical instructions are divided into three categories:

Privileged instructions - Privileged instructions execute in a privileged mode and will be trapped if executed outside this mode.

Control sensitive instructions - Control-sensitive instructions attempt to change the configuration of resources used.

Behavior-sensitive instructions - Behavior-sensitive instructions have different behaviors depending on the configuration of resources, including the load and store operations over the virtual memory.

A CPU architecture is virtualizable if it supports the ability to run the VM's privileged and privileged instructions in the CPU's user mode while the VMM runs in supervisor mode. When the privileged instructions including control- and behavior sensitive instructions of a VM are executed, they are trapped in the VMM. In this case, the VMM acts as a unified mediator for hardware access from different VMs to guarantee the correctness and stability of the whole system. RISC CPU architectures can be naturally virtualized because all control- and behavior-sensitive instructions are privileged instructions.

Hardware-Assisted CPU Virtualization:

This technique attempts to simplify virtualization because full or para virtualization is complicated. Intel and AMD add an additional mode called privilege mode level (some people call it Ring-1) to x86 processors. Therefore, operating systems can still run at Ring 0 and the hypervisor can run at Ring -1. All the privileged and sensitive instructions are trapped in the hypervisor automatically. This technique removes the difficulty of implementing binary translation of full virtualization. It also lets the operating system run in VMs without modification.

Memory Virtualization:

Virtual memory virtualization is similar to the virtual memory support provided by modern operating systems. In a traditional execution environment, the operating system maintains mappings of virtual memory to machine memory using page tables, which is a one-stage mapping from virtual memory to machine memory. All modern x86 CPUs include a memory management unit (MMU) and a translation lookaside buffer (TLB) to optimize virtual memory performance.

However, in a virtual execution environment, virtual memory virtualization involves sharing the physical system memory in RAM and dynamically allocating it to the physical

memory of the VMs. That means a two-stage mapping process should be maintained by the guest OS and the VMM, respectively: virtual memory to physical memory and physical memory to machine memory. Furthermore, MMU virtualization should be supported, which is transparent to the guest OS. The guest OS continues to control the mapping of virtual addresses to the physical memory addresses of VMs. But the guest OS cannot directly access the actual machine memory. The VMM is responsible for mapping the guest physical memory to the actual machine memory. Figure 2.16 shows the two-level memory mapping procedure.

I/O Virtualization:

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
- Direct I/O

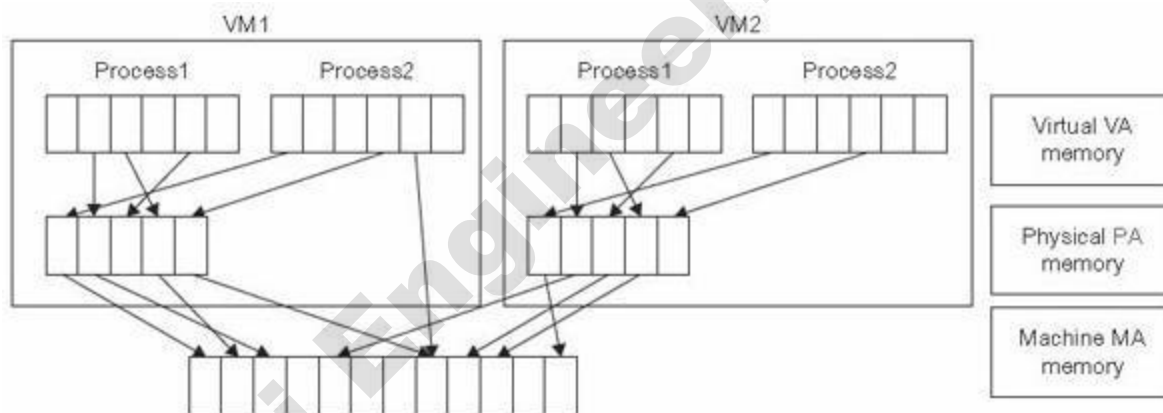


Figure 2.16 Two-level memory mapping procedure.

Full device emulation is the first approach for I/O virtualization. Generally, 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.

A single hardware device can be shared by multiple VMs that run concurrently. However, software emulation runs much slower than the hardware it emulates. The para

virtualization method of I/O virtualization is typically used in Xen. It is also known as the 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.

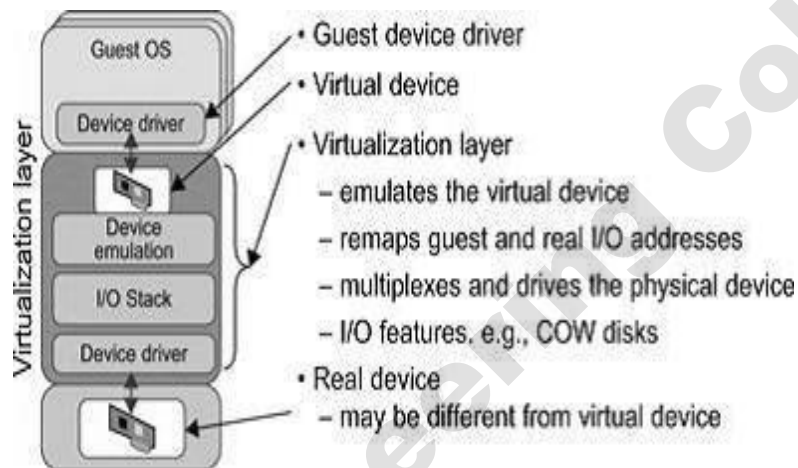


Figure 2.17 Device emulation for I/O virtualization implemented inside the middle layer that maps real I/O devices into the virtual devices for the guest device driver to use.

Virtualization in Multi-Core Processors:

Virtualizing a multi-core processor is relatively more complicated than virtualizing a uniprocessor. Though multicore processors are claimed to have higher performance by integrating multiple processor cores in a single chip, multi-core virtualization has raised some new challenges to computer architects, compiler constructors, system designers, and application programmers.

There are mainly two difficulties: Application programs must be parallelized to use all cores fully, and software must explicitly assign tasks to the cores, which is a very complex problem.

Virtualization Support and Disaster Recover:

One very distinguishing feature of cloud computing infrastructure is the use of system virtualization and the modification to provisioning tools. Virtualizations of servers on a shared cluster can consolidate web services. As the VMs are the containers of cloud services, the provisioning tools will first find the corresponding physical machines and deploy the VMs to those nodes before scheduling the service to run on the virtual nodes.

In addition, in cloud computing, virtualization also means the resources and fundamental infrastructure are virtualized. The user will not care about the computing resources that are used for providing the services. Cloud users do not need to know and have no way to discover physical resources that are involved while processing a service request.

Also, application developers do not care about some infrastructure issues such as scalability and fault tolerance (i.e., they are virtualized). Application developers focus on service logic. Figure 2.18 shows the infrastructure needed to virtualize the servers in a data center for implementing specific cloud applications.

Hardware Virtualization

In many cloud computing systems, virtualization software is used to virtualize the hardware. System virtualization software is a special kind of software which simulates the execution of hardware and runs even unmodified operating systems. Cloud computing systems use virtualization software as the running environment for legacy software such as old operating systems and unusual applications. Virtualization software is also used as the platform for developing new cloud applications that enable developers to use any operating systems and programming environments they like.

The development environment and deployment environment can now be the same, which eliminates some runtime problems. Some cloud computing providers have used virtualization technology to provide this service for developers. As mentioned before, system virtualization software is considered the hardware analog mechanism to run an unmodified operating system, usually on bare hardware directly, on top of software. Table 4.4 lists some of the system virtualization software in wide use at the time of this writing. Currently, the VMs installed on a cloud computing platform are mainly used for hosting third-party programs. VMs provide flexible runtime services to free users from worrying about the system environment

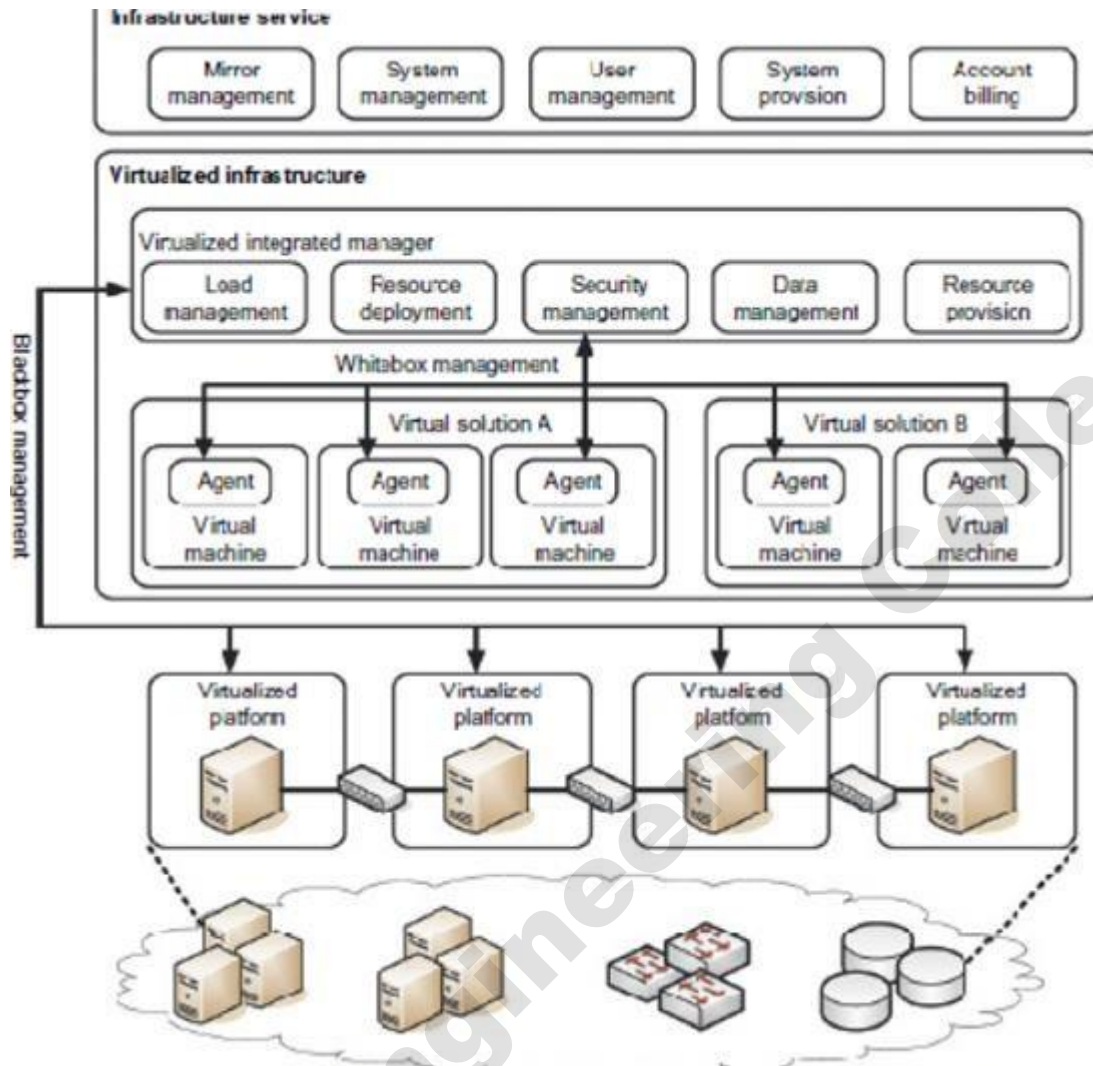
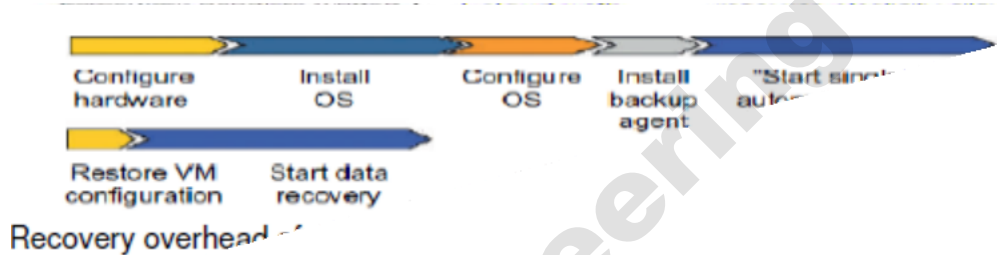


Figure 2.18 Virtualized Storage server and Networks

Using VMs in a cloud computing platform ensures extreme flexibility for users. As the computing resources are shared by many users, a method is required to maximize the users' privileges and still keep them separated safely. Traditional sharing of cluster resources depends on the user and group mechanism on a system. Such sharing is not flexible. Users cannot customize the system for their special purposes. Operating systems cannot be changed. The separation is not complete.

Table 4.4 Virtualized Resources in Compute, Storage, and Network Clouds [4]

Provider	AWS	Microsoft Azure	GAE
Compute cloud with virtual cluster of servers	x86 instruction set, Xen VMs, resource elasticity allows scalability through virtual cluster, or a third party such as RightScale must provide the cluster	Common language runtime VMs provisioned by declarative descriptions	Predefined from
Storage cloud with virtual storage	Models for block store (EBS) and augmented key/blob store (SimpleDB), automatic scaling varies from EP to fully automatic (S3)	SQL	
Network cloud services	Declarative placement		



An environment that meets one user’s requirements often cannot satisfy another user. Virtualization allows users to have full privileges while keeping them separate. Users have full access to their own VMs, which are completely separate from other users’ VMs. Multiple VMs can be mounted on the same physical server. Different VMs may run with different Oses. We also need to establish the virtual disk storage and virtual networks needed by the VMs. The virtualized resources form a resource pool.

The virtualization is carried out by special servers dedicated to generating the virtualized resource pool. The virtualized infrastructure (black box in the middle) is built with many virtualizing integration managers. These managers handle loads, resources, security, data, and provisioning functions.

Virtualization Support in Public Clouds

AWS provides extreme flexibility (VMs) for users to execute their own applications. GAE provides limited application-level virtualization for users to build applications only based on the services that are created by Google. Microsoft provides programming-level virtualization (.NET virtualization) for users to build their applications. The VMware tools apply to workstations, servers, and virtual infrastructure. The Microsoft tools are used on PCs and some special servers. The Xen Enterprise tool applies only to Xen-based servers.

Everyone is interested in the cloud; the entire IT industry is moving toward the vision of the cloud. Virtualization leads to HA, disaster recovery, dynamic load leveling, and rich provisioning support. Both cloud computing and utility computing leverage the benefits of virtualization to provide a scalable and autonomous computing environment.

Storage Virtualization for Green Data Centers

IT power consumption in the United States has more than doubled to 3 percent of the total energy consumed in the country. The large number of data centers in the country has contributed to this energy crisis to a great extent. More than half of the companies in the Fortune 500 are actively implementing new corporate energy policies. Recent surveys from both IDC and Gartner confirm the fact that virtualization had a great impact on cost reduction from reduced power consumption in physical computing systems. This alarming situation has made the IT industry become more energy aware.

With little evolution of alternate energy resources, there is an imminent need to conserve power in all computers. Virtualization and server consolidation have already proven handy in this aspect. Green data centers and benefits of storage virtualization are considered to further strengthen the synergy of green computing.

Virtualization for IaaS

VM technology has increased in ubiquity. This has enabled users to create customized environments atop physical infrastructure for cloud computing.

Use of VMs in clouds has the following distinct benefits:

- (1) System administrators consolidate workloads of underutilized servers in fewer servers;
- (2) VMs have the ability to run legacy code without interfering with other APIs;

- (3) VMs can be used to improve security through creation of sandboxes for running applications with questionable reliability;
- (4) Virtualized cloud platforms can apply performance isolation, letting providers offer some guarantees and better QoS to customer applications.

2.9.5.5 VM Cloning for Disaster Recovery

VM technology requires an advanced disaster recovery scheme. One scheme is to recover one physical machine by another physical machine. The second scheme is to recover one VM by another VM. Traditional disaster recovery from one physical machine to another is rather slow, complex, and expensive. Total recovery time is attributed to the hardware configuration, installing and configuring the OS, installing the backup agents, and the long time to restart the physical machine. To recover a VM platform, the installation and configuration times for the OS and backup agents are eliminated.

Therefore, we end up with a much shorter disaster recovery time, about 40 percent of that to recover the physical machines. Virtualization aids in fast disaster recovery by VM encapsulation. The cloning of VMs offers an effective solution. The idea is to make a clone VM on a remote server for every running VM on a local server. Among all the clone VMs, only one needs to be active. The remote VM should be in a suspended mode.

A cloud control center should be able to activate this clone VM in case of failure of the original VM, taking a snapshot of the VM to enable live migration in a minimal amount of time. The migrated VM can run on a shared Internet connection.

Only updated data and modified states are sent to the suspended VM to update its state. The Recovery Property Objective (RPO) and Recovery Time Objective (RTO) are affected by the number of snapshots taken. Security of the VMs should be enforced during live migration of VMs.