1

PARALLEL AND DISTRIBUTED COMPUTING

Unit Structure :

1.0 OBJECTIVES

After learning this chapter, you will be able to understand different cloud computing technologies available. It will help you in better understanding of parallel and distributed framework.

1.1 INTRODUCTION

Cloud computing is a revolutionary paradigm that has transformed the way businesses and individuals' access and utilize computing resources. It refers to the delivery of on-demand computing services over the internet, allowing users to store, process, and access data and applications without the need for local infrastructure.

At its core, cloud computing provides a flexible and scalable model for resource allocation. Instead of relying on local servers and data centers, users can leverage a vast network of remote servers maintained by cloud service providers. This eliminates the need for costly hardware investments and the burden of managing and maintaining infrastructure.

One of the key advantages of cloud computing is its elasticity. Users can easily scale their resources up or down based on demand, ensuring optimal performance and cost-efficiency. Whether it's a small startup requiring minimal resources or a large enterprise with heavy computational needs, cloud computing allows for seamless scalability.

Cloud computing offers several service models to cater to different requirements. Infrastructure as a Service (IaaS) provides virtualized computing resources such as virtual machines, storage, and networks. Platform as a Service (PaaS) offers a platform on which developers can build and deploy applications, while Software as a Service (SaaS) provides ready-to-use software applications accessible through the cloud.

Moreover, cloud computing facilitates collaboration and remote access. Users can access their data and applications from any device with an internet connection, enabling remote work and enhancing productivity. It also fosters data sharing and collaboration among teams by providing a centralized platform accessible to authorized users.

Security is a crucial aspect of cloud computing. Cloud service providers employ robust security measures to protect data from unauthorized access, ensuring data privacy and integrity. They also implement backup and disaster recovery mechanisms to safeguard against data loss.

The adoption of cloud computing has had a profound impact on various industries. It has enabled organizations to focus on their core competencies while offloading the responsibility of managing infrastructure. It has empowered startups with cost-effective solutions and leveled the playing field for businesses of all sizes.

In conclusion, cloud computing has revolutionized the IT landscape by providing scalable, on-demand computing resources. Its flexibility, accessibility, and cost-efficiency make it an attractive choice for individuals and businesses alike. As technology advances further, cloud computing is likely to continue its rapid growth, shaping the future of computing and transforming the way we live and work.

1.2 PARALLEL AND DISTRIBUTED COMPUTING

Parallel and distributed computing are two related concepts in the field of computer science that aim to increase computational power and efficiency by dividing tasks among multiple processors or machines. While they share some similarities, they differ in terms of architecture and resource management. Let's explore parallel and distributed computing in more detail.

Parallel computing involves the simultaneous execution of multiple computational tasks to solve a larger problem. It leverages the power of multiple processors or cores within a single machine, working together to complete a task faster. The processors communicate and collaborate with each other to divide the workload and share data. This allows for efficient utilization of computing resources and accelerates the overall computation.

In parallel computing, shared memory and shared address space are commonly used. Shared memory allows processors to access a common memory location, enabling efficient data sharing and communication. Each processor operates independently on a portion of the problem, and synchronization mechanisms ensure that processors coordinate their actions and avoid conflicts.

The diagram below illustrates a basic parallel computing architecture with multiple processors connected to a shared memory system:

```
 ______________________________
|        Processor 1        |
|______________________________|
|        Processor 2        |
|______________________________|
|        Processor 3        |
|______________________________|
|        Shared Memory      |
|______________________________|
```

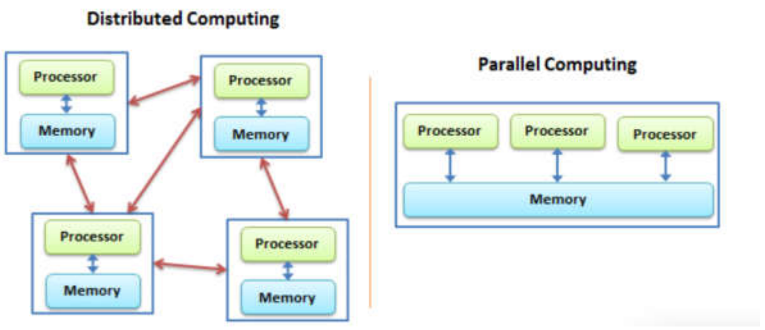
Distributed computing, on the other hand, involves the execution of a computational task across multiple interconnected machines or nodes. Each node in a distributed system has its own memory and processing unit, and the nodes communicate and coordinate their actions to solve a problem collectively. Distributed computing is often used to solve problems that require a vast amount of computational power, storage, or data processing capabilities.

In distributed computing, message passing is a common communication mechanism. Nodes exchange messages to share data and coordinate their activities. Distributed systems employ various algorithms and protocols to manage tasks distribution, load balancing, fault tolerance, and data consistency across multiple nodes.

The diagram below illustrates a basic distributed computing architecture with interconnected nodes:

```
 ________    ________    ________
| Node 1 |  | Node 2 |  | Node 3 |
|________|  |________|  |________|
   |____________|____________|
          Network
```

In a distributed computing environment, each node operates independently, working on its portion of the task. The nodes communicate and share intermediate results to collectively solve the problem. Distributed systems can be geographically distributed across multiple locations, enabling efficient resource utilization and fault tolerance.



Parallel and distributed computing techniques are often combined to achieve even higher levels of performance and scalability. For example, a large-scale problem can be divided into smaller subproblems, where each subproblem is solved in parallel on a cluster of distributed machines. This approach, known as parallel-distributed computing, allows for efficient utilization of both parallel processing within each machine and distributed computing across multiple machines.

1.3 ELEMENTS OF PARALLEL COMPUTING, ELEMENTS OF DISTRIBUTED COMPUTING

Elements of Parallel Computing:

1. **Task Decomposition:** In parallel computing, a problem is divided into smaller tasks that can be executed concurrently. Task decomposition involves identifying the dependencies and breaking down the problem into smaller, independent subtasks that can be executed simultaneously on different processors or cores.

2. **Data Partitioning:** Data partitioning involves dividing the data associated with a problem among different processors or cores. Each processor operates on its assigned data subset, minimizing data movement and maximizing locality, which can improve performance.

3. **Synchronization:** Synchronization mechanisms are essential for coordinating the activities of parallel processes. They ensure that multiple processes access shared resources or data in a controlled manner to avoid conflicts and maintain data integrity. Common synchronization techniques include locks, barriers, semaphores, and atomic operations.

4. **Load Balancing:** Load balancing aims to distribute the workload evenly among processors or cores to maximize resource utilization and minimize idle time. Load balancing techniques dynamically allocate tasks or data chunks to processors based on their current workload or performance characteristics.

5. **Communication:** Communication refers to the exchange of data and messages between parallel processes. Efficient communication mechanisms are crucial for sharing data, coordinating activities, and synchronizing the execution of parallel tasks. Communication can occur through shared memory or message passing.

Elements of Distributed Computing:

1. **Resource Discovery:** In distributed computing, resource discovery involves locating and identifying available computing resources (e.g., nodes, servers, services) within a distributed system. Resource discovery mechanisms allow nodes to find and utilize resources efficiently.

2. **Communication Protocols:** Distributed computing relies on communication protocols for nodes to exchange information and coordinate their activities. Protocols define the rules and formats for data transfer, message passing, and synchronization across the distributed system.

3. **Fault Tolerance:** Distributed systems are prone to failures, such as node crashes or network disruptions. Fault tolerance mechanisms ensure system resilience and reliability by detecting and handling failures. Techniques like replication, redundancy, and error detection and recovery are employed to maintain system integrity.

4. **Consistency and Data Management:** Consistency refers to the agreement or coherence of data across distributed nodes. Distributed systems often deal with shared data, and ensuring data consistency is crucial. Consistency protocols and distributed data management techniques, like distributed databases or distributed file systems, are used to maintain data integrity and coherence.

5. **Scalability:** Distributed computing emphasizes scalability to accommodate the increasing demands of users and applications. Scalability involves the ability to add or remove nodes dynamically, handle growing workloads, and distribute resources efficiently to ensure optimal performance.

Both parallel and distributed computing share elements such as communication, synchronization, and scalability. However, they differ in terms of the underlying architecture and resource management strategies. Understanding these elements is essential for designing and developing efficient parallel and distributed computing systems.

1.4 TECHNOLOGIES FOR DISTRIBUTED COMPUTING

There are several technologies available for distributed computing that enable the efficient execution of tasks across multiple interconnected machines or nodes. These technologies provide frameworks, libraries, and platforms for developing distributed applications and managing the complexities of distributed systems. Here are some notable technologies for distributed computing:

1. **Apache Hadoop:** Hadoop is a popular open-source framework for distributed storage and processing of large-scale data sets. It consists of the Hadoop Distributed File System (HDFS) for distributed storage and the MapReduce programming model for distributed processing. Hadoop enables parallel and fault-tolerant processing of big data by distributing data and computation across a cluster of machines.

2. **Apache Spark:** Spark is an open-source distributed computing system that provides a high-level API for large-scale data processing. It offers in-memory computing, allowing for faster data processing compared to disk-based systems like Hadoop. Spark supports various programming languages, such as Scala, Java, Python, and R, and provides libraries for machine learning, graph processing, and stream processing.

3. **Apache Kafka:** Kafka is a distributed streaming platform that enables the processing of real-time data streams. It provides high-throughput, fault-tolerant, and scalable messaging capabilities, allowing data to be reliably published, subscribed, and processed in real-time. Kafka is commonly used in distributed architectures for building event-driven systems and stream processing applications.

4. **Apache ZooKeeper:** ZooKeeper is a distributed coordination service that helps manage and synchronize distributed applications. It provides primitives such as distributed locks, configuration management, and leader election, which are essential for maintaining consistency, reliability, and fault tolerance in distributed systems. ZooKeeper is widely used in distributed databases, messaging systems, and other distributed applications.

5. **Kubernetes:** Kubernetes is an open-source container orchestration platform that facilitates the management and scaling of containerized applications across distributed clusters of machines. It automates tasks such as deployment, scaling, and load balancing, ensuring efficient resource utilization and fault tolerance. Kubernetes is commonly used for deploying and managing microservices-based distributed applications.

6. **Apache Cassandra:** Cassandra is a distributed NoSQL database that offers high scalability and fault tolerance. It is designed to handle large amounts of data across multiple nodes and data centers, providing high availability and low-latency access to data. Cassandra's decentralized architecture and data replication mechanisms make it well-suited for distributed applications requiring high performance and fault tolerance.

7. **Amazon Web Services (AWS) and Microsoft Azure:** AWS and Azure are cloud computing platforms that offer a wide range of distributed computing services. These platforms provide scalable and

managed infrastructure for distributed applications, including virtual machines, container services, serverless computing, data storage, messaging services, and more. They enable developers to build and deploy distributed applications without the need for extensive infrastructure management.

These are just a few examples of the many technologies available for distributed computing. Other notable technologies include Apache Storm for stream processing, Apache Flink for batch and stream processing, RabbitMQ for distributed messaging, and Google Cloud Platform (GCP) services like BigQuery, Pub/Sub, and Dataflow. The choice of technology depends on the specific requirements of the distributed application, including data volume, processing speed, fault tolerance, and scalability needs.

1.5 RPC

RPC (Remote Procedure Call) is a communication protocol commonly used in cloud computing environments to enable interaction between different components or services distributed across a network. RPC allows a program or service to invoke a procedure or function on a remote system as if it were a local call, abstracting the complexities of network communication.

In a cloud computing context, RPC facilitates the seamless communication between cloud services, allowing them to interact and exchange data efficiently. Here's how RPC is utilized in the cloud:

Service Invocation: Cloud services can use RPC to invoke procedures or functions on other services deployed in the cloud. This enables them to utilize the functionalities and capabilities offered by different services. For example, a storage service might use RPC to invoke a data processing service to perform analytics on stored data.

1. **Inter-Service Communication:** RPC serves as a communication mechanism for cloud services to exchange data and messages. Services can use RPC to request and receive data from other services, enabling them to collaborate and share information. This facilitates the development of modular and scalable cloud architectures, where services can communicate and interact seamlessly.

2. **Service Orchestration:** In a cloud environment, RPC plays a vital role in service orchestration. Orchestrating services involves coordinating the execution of multiple services to achieve a specific task or workflow. RPC allows services to communicate and synchronize their activities, ensuring proper sequencing and coordination. This enables the composition of complex workflows and distributed systems in the cloud.

3. **API Invocation:** RPC is commonly used in cloud APIs (Application Programming Interfaces) to enable developers to interact with cloud services. Cloud providers expose APIs that allow developers to invoke

various operations and functionalities provided by the cloud platform. RPC facilitates the communication between client applications and the cloud API, allowing developers to integrate cloud services into their applications seamlessly.

4. **Client-Server Communication:** RPC is employed in client-server communication scenarios in the cloud. Clients can initiate RPC calls to remote server-side components or services to request data, perform operations, or trigger specific actions. The RPC protocol handles the serialization, transmission, and deserialization of data between the client and server, providing a transparent and efficient communication channel.

Overall, RPC plays a crucial role in enabling communication and interaction between distributed components and services in the cloud. It simplifies the development and integration of cloud services, allowing them to work together seamlessly and efficiently. By abstracting the complexities of network communication, RPC helps developers focus on building scalable and modular cloud applications that leverage the capabilities offered by different cloud services.

1.6 DISTRIBUTED OBJECT FRAMEWORKS

Distributed object frameworks are software frameworks that provide a programming model and infrastructure for developing distributed applications in cloud computing environments. These frameworks simplify the development of distributed systems by abstracting the complexities of network communication, data sharing, and resource management. They allow developers to work with distributed objects, which are objects that span multiple machines or nodes in a networked environment. Here are some notable distributed object frameworks used in the cloud:

1. **Java RMI (Remote Method Invocation):** Java RMI is a distributed object framework in the Java ecosystem. It enables Java objects to invoke methods on remote objects residing on different machines or JVMs. RMI provides a transparent mechanism for remote method invocation, where the developer can invoke methods on remote objects as if they were local objects. It supports serialization, object activation, and garbage collection across distributed systems.

2. **CORBA (Common Object Request Broker Architecture):** CORBA is a distributed object framework that provides a standardized model for building distributed systems. It supports multiple programming languages and platforms, allowing objects written in different languages to communicate and interact seamlessly. CORBA uses an Object Request Broker (ORB) to handle communication between objects, providing transparent object invocation and location transparency.

3. **Microsoft .NET Remoting:** .NET Remoting is a distributed object framework for developing distributed applications in the Microsoft .NET ecosystem. It allows objects written in .NET languages to be accessed and invoked remotely. .NET Remoting provides a flexible and extensible programming model for developing distributed systems, supporting various communication protocols and transport mechanisms.

4. **gRPC:** gRPC is a modern open-source framework developed by Google that facilitates the development of high-performance distributed systems. It uses the Protocol Buffers data format for efficient serialization and supports multiple programming languages. gRPC provides a flexible and extensible RPC-based communication model and supports features like bidirectional streaming, authentication, and load balancing. It is widely used in cloud-native applications and microservices architectures.

5. **Apache Thrift:** Apache Thrift is an open-source distributed object framework developed by Apache Software Foundation. It enables efficient cross-language service development and supports a wide range of programming languages. Thrift uses a binary protocol and provides an interface definition language (IDL) for defining services and data structures. It supports various transport protocols, including HTTP, TCP, and named pipes.

6. **Erlang OTP:** Erlang OTP (Open Telecom Platform) is a concurrent and distributed programming framework designed for building fault-tolerant and highly available systems. OTP provides abstractions for distributed computing, including distributed process management, message passing, and fault recovery. It is widely used in telecommunications and messaging systems, where fault tolerance and high availability are critical.

These distributed object frameworks help developers build scalable, modular, and distributed applications in cloud computing environments. They handle the complexities of communication, data sharing, and resource management, allowing developers to focus on application logic. These frameworks provide features like location transparency, object serialization, and remote method invocation, enabling developers to work with distributed objects seamlessly. By abstracting the intricacies of distributed systems, these frameworks promote code reuse, reduce development effort, and facilitate the creation of distributed applications in the cloud.

1.7 SERVICE ORIENTED COMPUTING

Service-Oriented Computing (SOC) is a computing paradigm that focuses on designing and building software systems as a collection of loosely coupled, interoperable services. In SOC, services are self-contained, modular units of functionality that can be independently deployed, invoked, and composed to fulfil specific business requirements. These services communicate with each other through well-defined interfaces, enabling flexibility, scalability, and reusability in software development. Here's an overview of Service-Oriented Computing:

1. **Service:** A service is a self-contained unit of functionality that encapsulates business logic and provides well-defined interfaces for communication. Services can be implemented using various technologies, such as web services, RESTful APIs, or microservices. Services have clear boundaries, are platform-independent, and can be developed using different programming languages and frameworks.

2. **Service Description:** A service is described using a service description language, such as Web Services Description Language (WSDL) or OpenAPI, which specifies the service's interface, operations, input/output parameters, and protocols for communication. The service description enables service discovery and allows clients to understand how to interact with the service.

3. **Service Discovery:** Service discovery mechanisms enable clients to locate and identify available services in a distributed environment. Service registries, such as UDDI (Universal Description, Discovery, and Integration), or service discovery frameworks like Consul or Eureka, provide a centralized repository or distributed mechanisms for registering and discovering services. Clients can query the registry to find services that match their requirements and obtain their endpoint information.

4. **Service Composition:** Service composition involves combining multiple services to create more complex and higher-level business functionalities. Services can be composed dynamically at runtime to fulfill specific requirements. Service composition can be achieved through choreography, where services collaborate based on predefined message exchanges, or orchestration, where a central controller coordinates the execution of services to achieve a specific business process.

5. **Service Orchestration:** Service orchestration involves designing and managing the flow of services to achieve a specific business goal. An orchestrator, typically a centralized component, coordinates the execution of services by invoking their operations in a predefined sequence. Orchestration allows for complex business processes to be modeled and managed effectively by coordinating the interactions between services.

6. **Service Choreography:** Service choreography refers to the collaboration and coordination of services without a central controller. Services interact based on predefined message exchanges and event-driven communication. Choreography enables more decentralized and loosely coupled interactions between services, allowing for flexibility and scalability in complex systems.

7. **Service Governance:** Service governance focuses on managing and controlling the lifecycle of services within an organization or ecosystem. It involves policies, standards, and guidelines for service development, deployment, versioning, security, and quality assurance. Service governance ensures consistency, compliance, and proper management of services throughout their lifecycle.

8. **Service Security:** SOC emphasizes security in service interactions. Service security mechanisms include authentication, authorization, encryption, and message integrity to ensure secure communication between services. Security policies and mechanisms are implemented at both service endpoints and in the underlying communication infrastructure to protect against potential threats and vulnerabilities.

Service-Oriented Computing promotes modularity, interoperability, and flexibility in software development. It enables organizations to build scalable and adaptable systems by leveraging existing services, promoting code reuse, and facilitating system integration. SOC provides a foundation for building distributed architectures, cloud-native applications, and microservices-based systems, supporting agility and the ability to respond to changing business needs.

1.8 VIRTUALIZATION

Virtualization is a technology that enables the creation of virtual instances or representations of computing resources, such as servers, operating systems, storage devices, or networks. It allows multiple virtual environments to run concurrently on a single physical infrastructure, providing improved efficiency, flexibility, and resource utilization. Here's an overview of virtualization:

1. **Server Virtualization:** Server virtualization is one of the most common forms of virtualization. It involves partitioning a physical server into multiple virtual machines (VMs), each running its own operating system and applications. Server virtualization allows for the consolidation of multiple servers onto a single physical machine, reducing hardware costs, power consumption, and physical space requirements.

2. **Hypervisor:** A hypervisor, also known as a virtual machine monitor (VMM), is the software layer that enables the creation and management of virtual machines. It abstracts the underlying hardware and provides a virtualization layer that allows multiple VMs to run on a single physical server. The hypervisor ensures isolation, resource allocation, and facilitates the communication between VMs and the physical hardware.

3. **Operating System Virtualization:** Operating system (OS) virtualization, also known as containerization, allows multiple isolated user spaces or containers to run on a single operating system kernel. Containers share the host operating system, libraries, and resources,

but provide separate and isolated execution environments. OS virtualization offers lightweight and fast deployment of applications, enabling efficient resource utilization and scalability.

4. **Desktop Virtualization:** Desktop virtualization enables the creation of virtual desktop infrastructure (VDI) by hosting multiple desktop environments on a centralized server or data center. Users can access their virtual desktops remotely, allowing for flexibility and mobility. Desktop virtualization simplifies desktop management, enhances security, and enables remote collaboration.

5. **Storage Virtualization:** Storage virtualization abstracts physical storage resources, such as hard drives or storage arrays, and presents them as logical storage pools. It allows for centralized management, improved scalability, and simplified storage provisioning. Storage virtualization enables features like data migration, snapshots, replication, and backup, enhancing data availability and disaster recovery.

6. **Network Virtualization:** Network virtualization abstracts the physical network infrastructure, creating virtual networks that operate independently and are isolated from each other. Virtual networks can be provisioned and managed programmatically, allowing for efficient allocation of network resources and simplified network management. Network virtualization enables the creation of virtual LANs, virtual switches, and virtual routers, facilitating multi-tenancy, segmentation, and network flexibility.

Benefits of Virtualization:

Virtualization offers several benefits, including:

- **Increased Efficiency:** Virtualization enables better utilization of hardware resources, leading to cost savings, reduced power consumption, and improved efficiency
- **Flexibility and Scalability:** Virtualization provides the ability to dynamically allocate and adjust resources based on workload demands, allowing for scalability and flexibility.
- **Consolidation and Resource Optimization:** Virtualization allows for the consolidation of multiple systems onto a single physical infrastructure, reducing hardware and management costs.
- **Improved Disaster Recovery and High Availability:** Virtualization enables the creation of snapshots, backups, and replicas, facilitating faster disaster recovery and enhancing system availability.
- **Simplified Management:** Virtualization centralizes management and provisioning, making it easier to deploy, monitor, and maintain virtualized environments.

Virtualization has become a fundamental technology in modern data centers and cloud computing environments, enabling efficient resource utilization, agility, and cost savings. It forms the foundation for technologies like cloud computing, containerization, and software-defined infrastructure, revolutionizing the way computing resources are provisioned, managed, and utilized.

1.9 SUMMARY

Cloud computing is a paradigm that involves the delivery of on-demand computing resources over the internet. It provides organizations and individuals with access to a shared pool of configurable computing resources, including networks, servers, storage, applications, and services, that can be rapidly provisioned and released with minimal management effort.

Cloud computing offers several advantages, such as:

1. **Scalability:** Cloud resources can be easily scaled up or down based on demand. Organizations can quickly allocate additional resources during peak periods and scale them back when demand decreases, enabling cost optimization and efficient resource utilization.

2. **Cost Efficiency:** Cloud computing eliminates the need for upfront infrastructure investments. Organizations can pay for cloud services on a pay-as-you-go basis, allowing for cost savings by reducing capital expenditure and shifting to operational expenditure.

3. **Flexibility and Agility:** Cloud services provide flexibility in terms of access and usage. Users can access cloud resources from anywhere with an internet connection and can choose the type and level of services based on their specific requirements. Cloud platforms also offer rapid deployment of applications and services, enabling faster time-to-market for businesses.

4. **Reliability and High Availability:** Cloud service providers typically have robust infrastructure and redundant systems to ensure high availability and reliability. They often provide service level agreements (SLAs) that guarantee a certain level of uptime and performance, minimizing downtime and ensuring business continuity.

5. **Security:** Cloud providers invest heavily in security measures to protect customer data. They implement advanced security controls, encryption, access management, and regular audits to ensure data privacy and protection.

6. **Collaboration and Scalable Services:** Cloud computing enables seamless collaboration by providing shared access to resources, allowing teams to work on the same documents and applications simultaneously. It also offers a wide range of pre-built services and APIs that can be easily integrated into applications, enabling the development of scalable and feature-rich solutions.

7. **Disaster Recovery:** Cloud computing facilitates efficient data backup, replication, and disaster recovery solutions. Organizations can replicate their data and systems across multiple geographical locations, ensuring data redundancy and minimizing the risk of data loss.

Cloud computing is categorized into different service models:

1. **Infrastructure as a Service (IaaS):** Provides virtualized computing resources, such as virtual machines, storage, and networks, allowing users to deploy and manage their applications and systems.

2. **Platform as a Service (PaaS):** Offers a platform and runtime environment for developing, testing, and deploying applications without the need to manage the underlying infrastructure. PaaS provides a higher level of abstraction, enabling developers to focus on application development rather than infrastructure management.

3. **Software as a Service (SaaS):** Delivers ready-to-use software applications over the internet on a subscription basis. Users can access and use these applications through a web browser or thin client without the need for local installation or maintenance.

Cloud computing has revolutionized the IT industry, enabling organizations to focus on their core business activities while leveraging scalable and cost-effective computing resources. It has become an essential component of digital transformation strategies, empowering businesses to innovate, collaborate, and scale their operations efficiently.

1.10 REFERENCES

Enterprise Cloud Computing Technology, Architecture, Applications, Gautam Shroff, Cambridge University Press, 2010

- Mastering In Cloud Computing, Rajkumar Buyya, Christian Vecchiola And ThamariSelvi S, Tata Mcgraw-Hill Education, 2013

- Cloud Computing: A Practical Approach, Anthony T Velte, Tata Mcgraw Hill, 2009

References:

- Architecting the Cloud: Design Decisions for Cloud Computing Service Models

- (SaaS, PaaS, and IaaS), Michael J. Kavis, Wiley CIO, 2014

- Cloud Computing: SaaS, PaaS, IaaS, Virtualization, Business Models, Mobile,

Security and More, Kris Jamsa, Jones & Bartlett Learning, 2013

1.11 UNIT END QUESTION

1. Explain Parallel computing.
2. Explain Distributed Computing.
3. Difference between Parallel and Distributed system.
4. Explain elements of parallel and distributed computing.
5. Explain Virtualization in detail
6. Explain different Technologies for distributed computing.

❖❖❖❖

2

COMPUTING PLATFORMS-I

Unit Structure

2.0 OBJECTIVE

- To understand the concept of cloud computing technologies.
- To learn different types of virtualization in cloud computing.
- To understand the different characteristics of cloud computing.
- To understand the how and why the internet also evolved into a platform for enterprise applications

2.1 INTRODUCTION

Cloud computing technologies are:

- Virtualization
- Service-Oriented Architecture (SOA)
- Grid Computing
- Utility Computing

Virtualization

"Virtualization means the process of creating a virtual environment to run multiple applications and operating systems on the same server".

Types of Virtualization

1. Hardware virtualization
2. Server virtualization
3. Storage virtualization
4. Operating system virtualization
5. Data Virtualization

Service-Oriented Architecture (SOA)

Service-Oriented Architecture enables organizations to access on-demand cloud-based computing solutions on the report of the change of business requirement.

Applications of Service-Oriented Architecture

1. It is used in the healthcare industry.
2. It is used to create different mobile applications and games using SOA.
3. In the air force, SOA infrastructure is used to establish situational awareness systems.

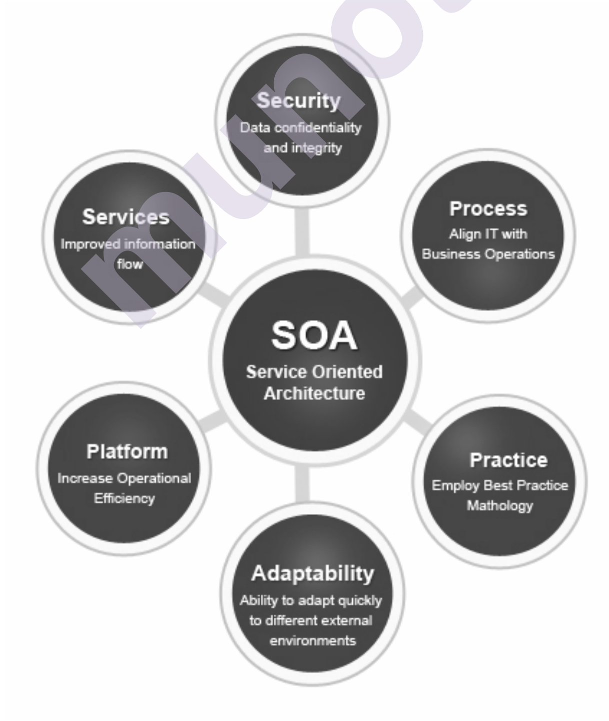

Security
Data confidentiality and integrity
Services
Improved information flow
Process
Align IT with Business Operations
SOA
Service Oriented Architecture
Platform
Increase Operational Efficiency
Practice
Employ Best Practice Mathology
Adaptability
Ability to adapt quickly to different external environments

Fig. 1. SOA

Grid computing

Grid computing is also known as distributed computing. It is a type of processor architecture that merges various different computing resources from multiple locations to achieve a common goal.

Types of machines used in computing:

1. Control Node: It is a group of servers which administers the whole network.

2. Provider: It is a computer which contributes its resources in the network resource pool.

3. User: It uses the resources on the network.

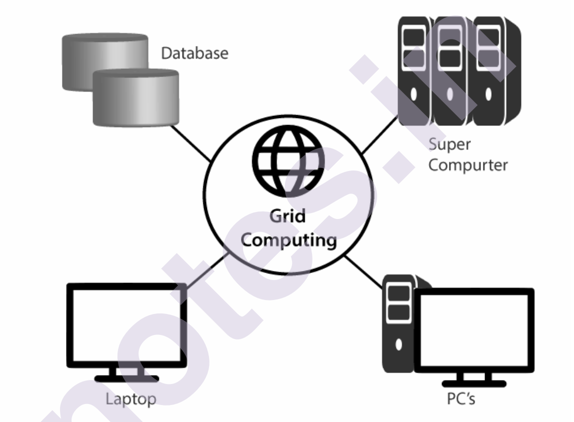


Fig.2 Grid Computing

Utility Computing

Utility computing is the bulk trending IT service model. It supply on-demand computing resources (i.e. computation, storage, and programming services via API) and infrastructure based on the pay per use method.

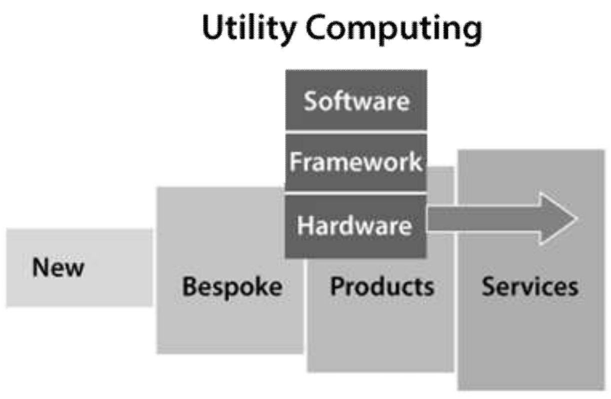


Fig.3. Utility Computing

- Cloud computing gives guarantee to transform computing into a utility delivered over the internet.
- Enterprise architecture is a function within IT departments that has developed over time, playing a high value role in managing transitions to new technologies, such as cloud computing.

2.2 CLOUD COMPUTING DEFINITION AND CHARACTERISTICS

- Cloud computing refers to different technologies, services, and concepts.
- It is associated with virtualized infrastructure or hardware on demand, utility computing, IT outsourcing, platform and software as a service, and many other things that now are the heart of the IT industry.
- Figure 4 shows too many different ideas included in current definitions of cloud computing systems.
- The term cloud has an abstraction of the network in system diagrams. This meaning is also put into cloud computing, which refers to an Internet-centric way of computing.
- The Internet plays a fundamental role in cloud computing, it shows the medium or the platform through which many cloud computing services are delivered and made accessible.

Definition of cloud computing:

"Cloud computing introduces both the applications delivered as services over the Internet and the hardware and system software in the data centers that provide those services."

- Cloud computing as aoccurance touching on the entire stack: from the underlying hardware to the high-level software services and its applications.
- Here is the concept of everything as a service, like XaaS, the different components of a system IT infrastructure, development platforms, databases, and so on can be delivered, measured, and consequently cost as a service.
- Cloud computing is a model for defining ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., like networks, servers, storage, applications, and services) that can be instantly provisioned and free with less management effort or service provider interaction.

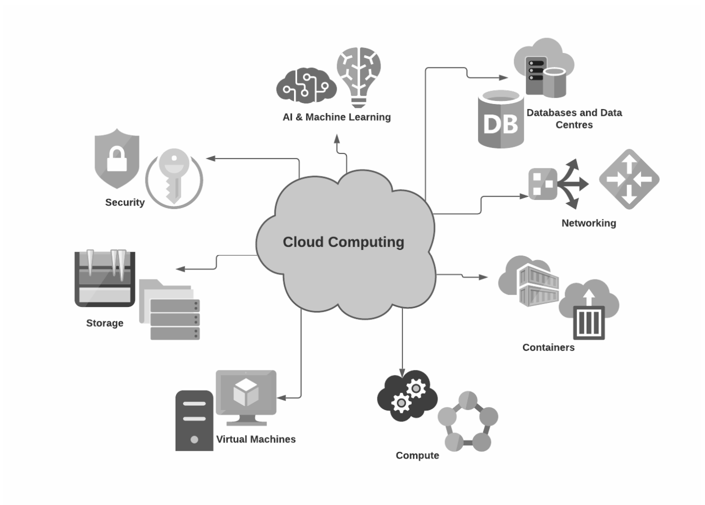


Fig. 4 Cloud computing environment

- Utility-oriented approach is an important aspect of cloud computing
- Cloud computing concentrates on delivering services with a given pricing model, in most cases a pay-per-use method.
- It is also possible to access online storage, rent the virtual hardware, or use development platforms and pay only for their effective usage, with no or minimal up-front costs.
- All these operations can be executed and billed simply by entering the credit card details and accessing the exposed services through a Web browser.
- The criteria to disfavor whether a service is delivered in the cloud computing style:

○ The service is accessible through a Web browser or a Web services application programming interface.

○ Zero capital expenditure

○ pay only for what you use

- **Characteristics**

○ **Resources Pooling**

Cloud providers pulled the computing resources to give services to multiple customers with the help of a multi-tenant model.

○ **On-Demand Self-Service**

It is one of the key and valuable features of Cloud Computing as the user can regularly monitor the server uptime, capabilities, and allotted network storage.

- **Easy Maintenance**

The servers are easy to maintain and the downtime is required very less and even in some situations, there is no downtime.

- **Large Network Access**

The user can access the data of the cloud or upload the data to the cloud from anywhere just with the help of a device and an internet connection.

- **Availability**

The potential of the Cloud can be altered as per the use and can be extended a lot. It studies storage usage and allows the user to purchase extra Cloud storage if needed for a very small amount.

- **Automatic System**

Cloud computing automatically analyzes the data needed and supports a metering capability at some level of services.

- **Economical**

It is a one-time investment as the company has to buy the storage and a small part of it can be provided to the many companies which save the host from monthly or yearly costs.

- **Security**

Security creates a snapshot of the data stored so that the data may not get lost even if one of the single servers gets damaged.

- **Pay as you used**

In cloud computing, the user has to pay only for the service they have utilized and used.

- **Measured Service**

Cloud Computing resources used to handle and the company uses it for recording.

2.3 ENTERPRISE COMPUTING,

- Enterprise computing defines the use of computers for data processing in large organizations, also called as 'information systems' (IS), or even 'information technology' (IT).
- The key elements of cloud computing, are:
  - Computing resources wrap up as a commodity and made available over the internet.
  - Ability for end-users to rapidly give the resources they need.
  - a costing that charges consumers only for those cloud resources they actually use.

MAINFRAME ARCHITECTURE

- The mainframe architecture is shown in Figure 5.
- A terminal-based user interface display screens controlled by the mainframe server using the "Virtual Telecommunications Access Method (VTAM)"

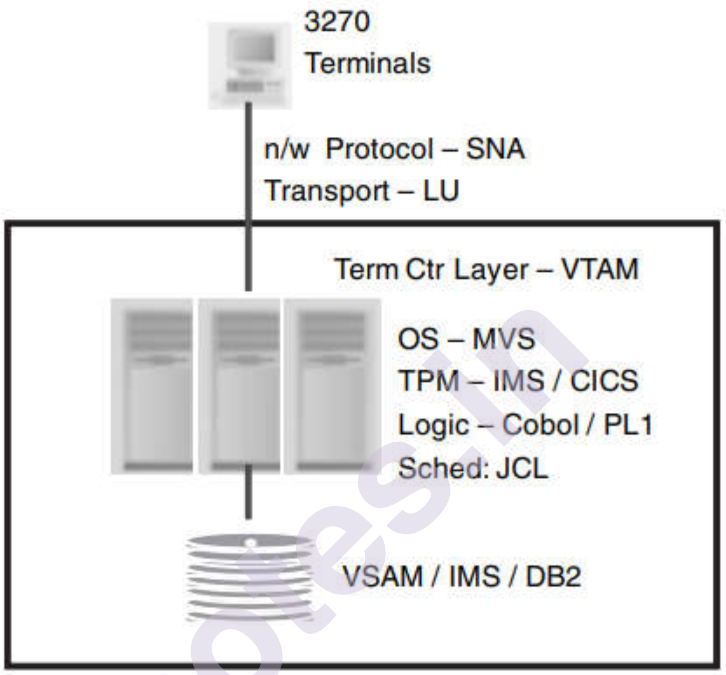


Fig. 5. Mainframe architecture

- for entering and viewing information. Terminals communicated with the mainframe using the 'systems network architecture' (SNA) protocol, instead of the ubiquitous TCP/IP protocol of today.
- These mainframe computers had limited CPU power by modern standards, their I/O bandwidth was extremely generous relative to their CPU power.
- In Outcome, mainframe applications were constructed using a batch architecture to minimize utilization of the CPU during data entry or retrieval.
- In the old time mainframe architectures, application data was stored either in structured files, or in database systems based on the hierarchical or networked data model.
- The storage subsystem in mainframes, called 'virtual storage access mechanism' (VSAM), built in support for a variety of file access and indexing mechanisms as well as sharing of data between concurrent users using record level locking mechanisms.

CLIENT-SERVER ARCHITECTURE

- The Client-Server Architecture system shown in Figure 6.
- First, the 'forms' architecture for minicomputer-based data processing became popular.
- This architecture took a part in the use of terminals to access server-side logic in C, mirroring the mainframe architecture, PC-based forms applications provided graphical 'GUIs' as opposed to the terminal-based character-oriented 'CUIs.'
- The GUI forms model represents the first client server architecture.
- The 'forms' architecture evolved into the more general client-server architecture, wherein significant processing logic executes in a client application, such as a desktop personal computer.

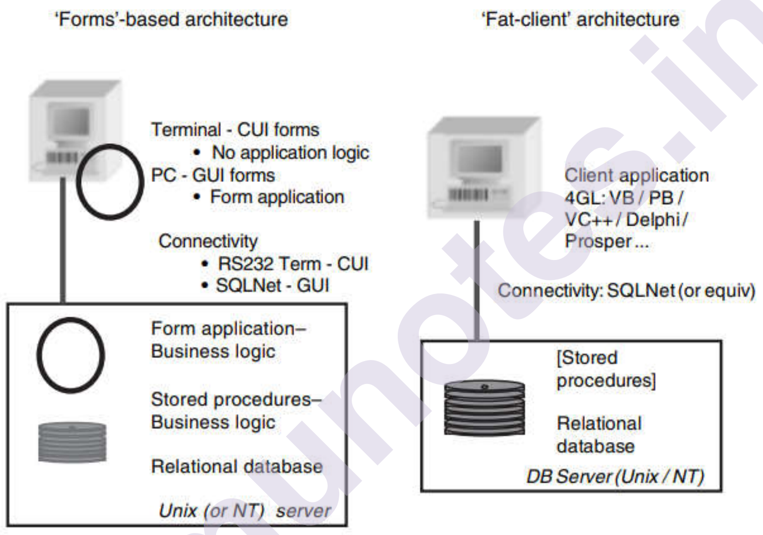


Fig.6. Client-server architectures

- The client-server architecture is also referred to as a 'fat-client' architecture, as shown in Figure 6.
- The client application makes calls using SQL command, to the relational database using networking protocols., running over a local area (or even wide area) network using TCP/IP. Business logic largely stays within the client application code, though some business ideas can also be implemented within the database for faster performance, using 'stored procedures.'
- client-server applications normally made many (client) requests to the server during the processing of a single screen. This type of request was large as compared to the terminal, where only the input and final result of a computation were transferred.

3-TIER ARCHITECTURES WITH TP MONITORS

- Transaction-processing monitors were redeveloped to solve this problem for midrange database servers.
- These TP monitors were the first examples of 'middleware,' which reside between clients and a database server to manage access to scarce server resources, essentially by queuing client requests. Thus, as depicted in Figure 7, by limiting concurrent requests to a small number, say 50, the server could handle the large load while the clients only paid a small price in response time while their requests waited in the TP monitor queues.

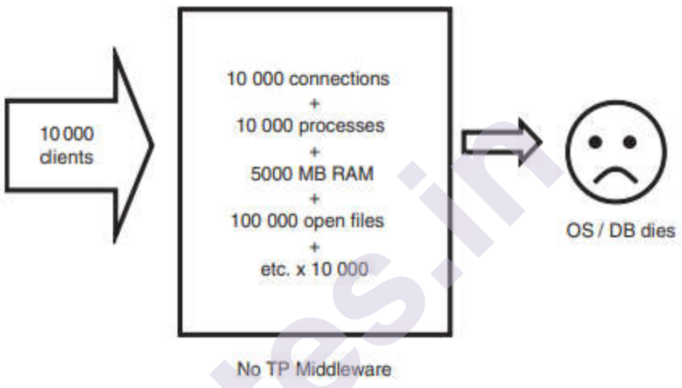


FIGURE 1.3. Client-server fails

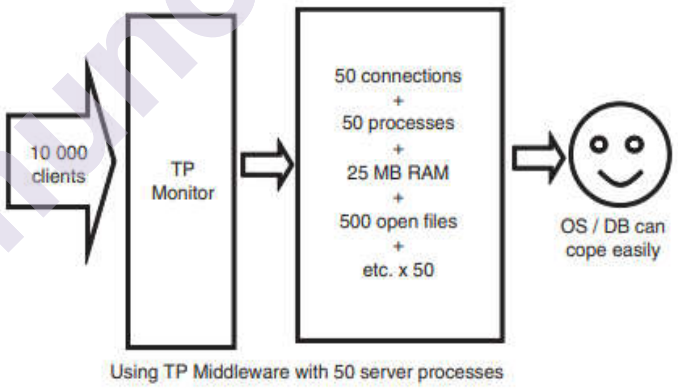


Fig.7 3-tier architecture scales

- In a TP monitor architecture, the requests being queued were 'services' implementing business logic and database operations.
- These were executed as a number of Unix processes, each one is publishing many such services, typically as remote procedure calls.
- The TP monitor model is also called the 3-tier architectural model, where client, business and data layers are clearly separated and often also stay on separate machines, as shown in diagram 8.

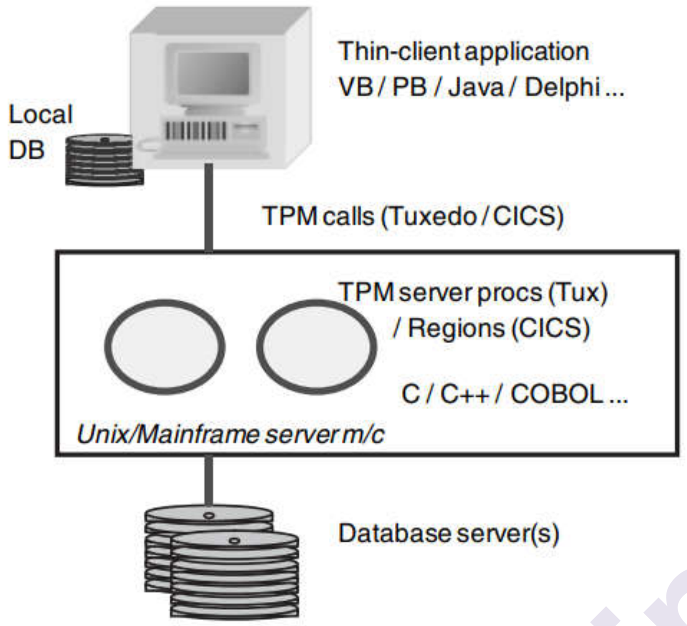

Thin-client application
VB / PB / Java / Delphi ...
Local DB
TPM calls (Tuxedo / CICS)
TPM server procs (Tux) / Regions (CICS)
C / C++ / COBOL ...
Unix/Mainframe server m/c
Database server(s)

Fig.8 3-tier TP monitor architecture

- The 3-tier model based on 'object-based' access to services, replacing flat remote procedure calls, with the introduction of object-oriented distributed communication systems such as CORBA.
- In CORBA (Common Object Request Broker Architecture), the client application can communicate with services on the server via methods on distributed objects in lieu of having to build in application specific message handling for passing parameters to services and receiving their responses accordingly.

2.4 THE INTERNET AS A PLATFORM

- The internet is used as a communication infrastructure for data sharing between large government research labs, and grew to include academic institutions across the world.
- The internet uses a platform for sharing data, documents, using the HTTP protocol and HTML.
- Using a browser, information published over the internet could be accessed unspecified by the public at large, giving rise to the world wide web (WWW).

INTERNET TECHNOLOGY AND WEB-ENABLED APPLICATIONS

- Internet-based applications depend on HTTP and HTML both are now standards defined by the world wide web consortium (W3C).

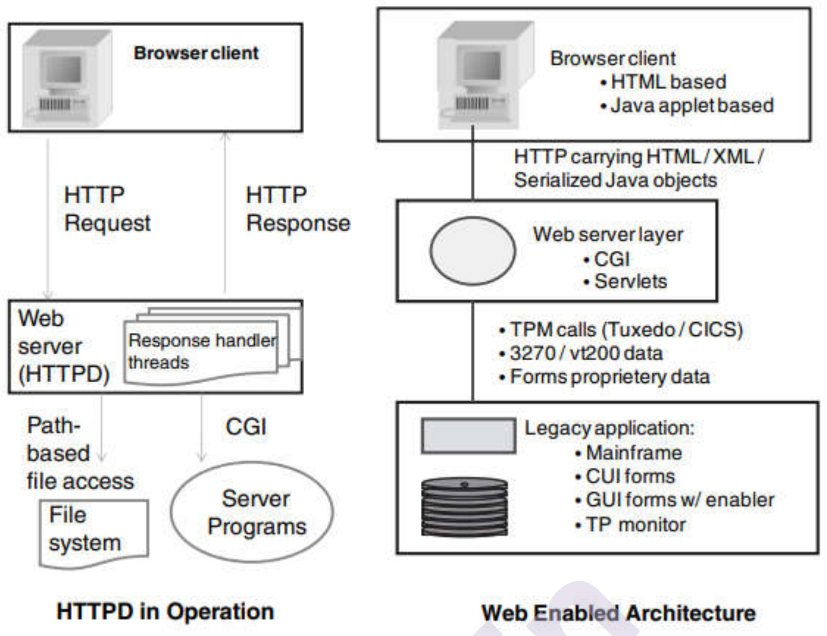


Fig.9. Internet technology and web-enabled applications

- As shown in Figure 9, a web server is a process, such as the Apache HTTPD daemon, that receives HTTP requests from clients, typically web browsers.
- Requests are lined up until assigned to a request handler thread within the web server process.
- The server sends an HTTP response which contains data, either recalled directly from a file system or as computed by a server program initiated to respond to the client request.
- The CGI (common gateway interface) protocol is used by the web server to launch server programs and communicate with them, i.e. pass parameters and accept their results, such as data retrieved from a database. The browser client purely interprets HTML returned by the server and displays it to the user.

WEB APPLICATION SERVERS

- In this application architecture, processing logic, including database access, took place outside the web server process via the programs invoked by it.
- To serve HTTP requests from files or CGI scripts, requests could also be processed from multithreaded execution environments, called containers carried within the web server.
- The servlet container, first introduced in the pure Java Apache Tomcat server, which allowed Java programs to execute in a multi-threaded manner within the server process as Servlet code.

- The container also handles load balancing across coming requests using these threads, as well as database connection pooling, in a mode similar to TP monitors.
- The application-server architecture is the same advantages of a 3-tier architecture, i.e, the ability to operate larger workloads as compared to the client-server model.

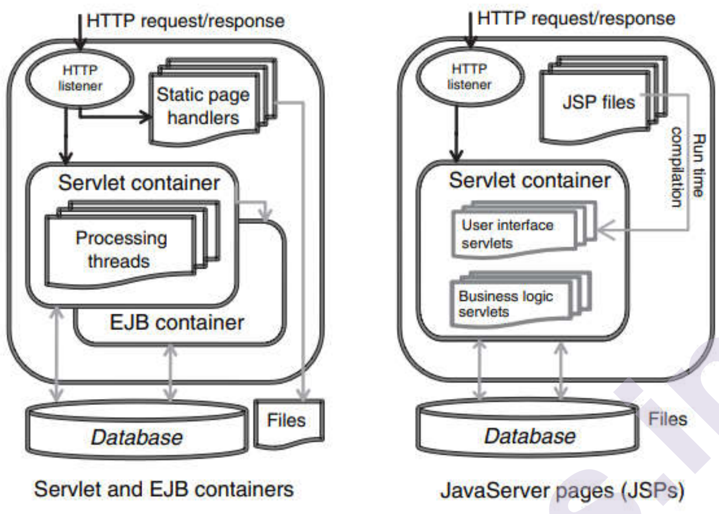


Fig.10. Web application server

- Servlet code used to respond to HTTP type requests.
- JSPs also introduced in Tomcat server, which allowed the user interface behavior to be encoded directly as Java code embedded within HTML code. Such 'JSP' files are dynamically compiled into servlets code, as shown on the right in Figure 10.

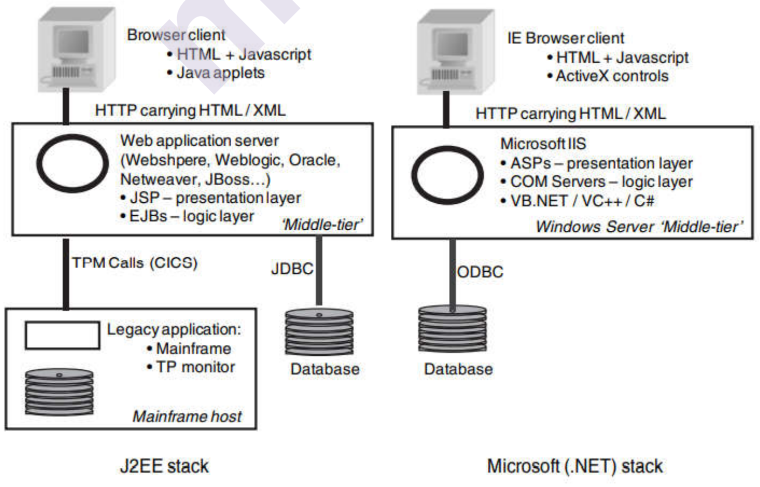


Fig.11. Web application server technology stacks

- The Microsoft web or application server, IIS (Internet Information Server), runs only on the Windows operating system.
- The J2EE stack, multiple language support provided, including C, C++, and Microsoft specific languages such as C# and VB.
- The application container at this point was simply Microsoft's COM environment on the Windows operating system that allowed multiple processes to execute and communicate with each other. Recent versions of this stack are called the .NET framework.

INTERNET OF SERVICES

- The web services and architectures for integration over the internet
- IOS provides a rich client-side interface such as a Google Map, can be implemented in Javascript that accesses the server over HTTP using asynchronous requests via XMLHTTPRequest.
- In conventional server-to-server web services, application server code accesses published services from another server via SOAP over HTTP.

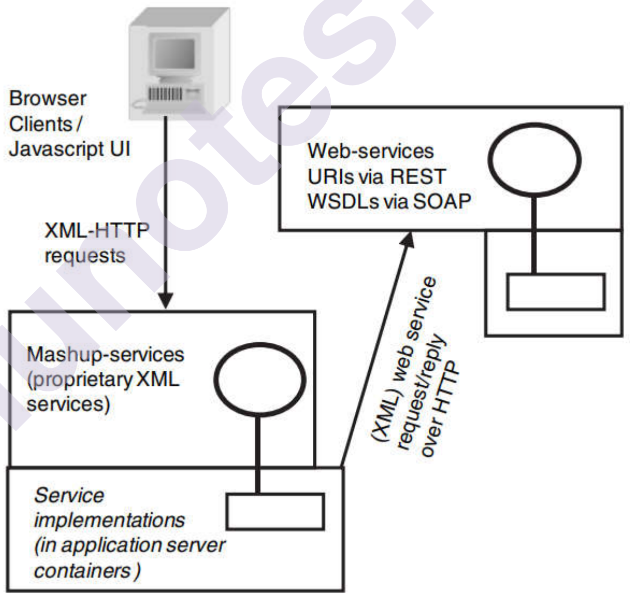


Fig.12 Internet of services

- The software as a service and cloud computing paradigms bring in this contractual aspect formally, while also re-spotlight the human element.

2.5 SUMMARY

- Virtualization is the process of creating a virtual environment to run multiple applications and operating systems on the same server.
- Cloud computing refers to both the applications delivered as services over the Internet and the hardware and system software in the data centers that provide those services.

- The internet is used as a communication infrastructure for data sharing between large government research labs, and grew to include academic institutions across the world.
- In a web-enabled application architecture, processing logic, including database access, took place outside the web server process via scripts or programs invoked by it, using CGI for interprocess communication.
- An Internet of Services (IoS) continues, it is of utmost importance for the telecoms industry to understand what the IoS is and upon what foundations and methodologies the IoS is based and built on.

2.6 REFERENCE FOR FURTHER READING

- Enterprise Cloud Computing Technology, Architecture, Applications, GautamShroff, Cambridge University Press, 2010
- Mastering In Cloud Computing, RajkumarBuyya, Christian Vecchiola And ThamariSelvi S, Tata Mcgraw-Hill Education, 2013
- Cloud Computing: A Practical Approach, Anthony T Velte, Tata Mcgraw Hill, 2009

2.7 UNIT END EXERCISES

1. What is cloud computing? Explain the characteristics of cloud computing.
2. Write a short note on:
   a. Client-Server Architecture.
   b. Mainframe Architecture.
3. Explain the internet as a platform in cloud computing.

❖❖❖❖

3

COMPUTING PLATFORMS-II

Unit Structure :

3.0 OBJECTIVE

- To understand the different types of cloud computing servcies.
- To understand the enterprise architecture used in cloud computing.
- To study the different types of clouds.

3.1 INTRODUCTION

- Cloud Computing can be defined as the exercise of using a network of remote servers hosted on the Internet to store, manage, and process data, alternatively a local server or a may be a personal computer.
- Organizations offering such types of cloud computing services are called cloud providers and charge for cloud computing services based on their usage.
- Grids and clusters are the base for cloud computing.

3.2 CLOUD COMPUTING SERVICES: SAAS, PAAS, IAAS, SOFTWARE AS A SERVICE (SAAS)

- Software-as-a-Service (SaaS) is a software delivery model that give access to applications through the Internet as a Webbased service.
- It provides a free users from complex hardware and software management by offloading such tasks to third parties, which build applications accessible to multiple users through a Web browser.

- Example:customers neither need to install anything on their premises nor have to pay costs to buy the software and the need licenses. They directly access the application website, enter their username and password and other billing details, and can immediately use the application.
- On the source side, they keep maintaining specific details and features of each customer's infrastructure and make it available on user request.
- SaaS as a one-to-many software delivery model, whereby an application is shared across multiple users.
- Example includes CRM3 and ERP4 applications that add up common needs for almost all enterprises, from small to medium-sized and large businesses.
- This structure relives the development of software platforms that provide a general set of features and support specialization and ease of integration of new components.
- SaaS applications are naturally multitenant.
- The acronym SaaS was then invented in 2001 by the Software Information & Industry Association (SIIA).
- The analysis done by SIIA was mainly aligned to cover application service providers (ASPs) and all their variations, which imprison the concept of software applications consumed as a service in a wide sense.
- ASPs Core characteristics of SaaS:

○ The product sold to customers is an application approach.

○ The application is centrally managed.

○ The service delivered is one-to-many.

○ The service provides is an integrated solution delivered on the contract, which means provided as promised

Platform as a service

- Platform-as-a-Service (PaaS) which provides a development and deployment platform for running applications in the cloud.
- They compose the middleware on top of which applications are built.
- Following figure shows a general overview of the features characterizing the PaaS.

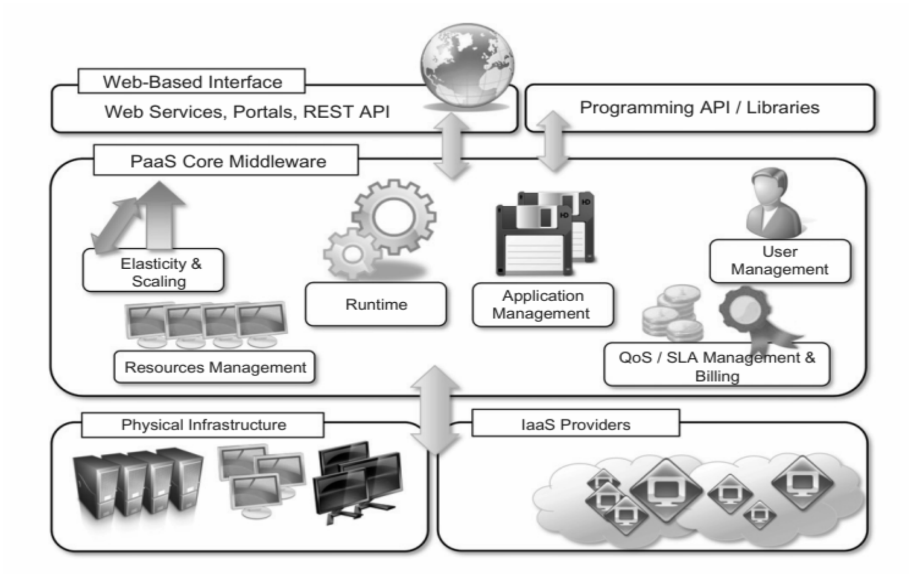


Fig.1. The Platform-as-a-Service reference model

- Application management is the key functionality of the middleware systems.
- PaaS implementations provide applications with a runtime environment and do not expose any service for managing the underlying infrastructure.
- They automating the process of deploying applications to the infrastructure, configuring application components, provisioning and configuring supporting technologies such as load balancers and databases, and managing system change based on policies set by the user.
- The core middleware is responsible for managing the resources and scaling applications on demand, according to the adherence made with users.
- The core middleware exposes interfaces that enable programming and installing applications on the cloud.
- The PaaS model provides a complete object model for representing an application and provides a programming language-based approach.
- In this point the traditional development environments can be used to design and develop applications, which are then deployed on the cloud by using the APIs revealed by the PaaS provider.
- PaaS offers middleware for developing applications together with the infrastructure.

Infrastructure as a service or hardware as a service

- Infrastructure as a Service is the most popular model and developed market segment of cloud computing.
- They deliver customizable infrastructure on request.
- The IaaS offering umbrella ranges from single servers to entire infrastructures, including network devices, load balancers, and database and Web servers.
- The main aim of this technology used to deliver and implement these solutions is hardware virtualization:

○ one or more virtual machines configured and interconnected

○ Virtual machines also constitute the atomic components that are installed and charged according to the specific features of the virtual hardware:

■ memory

■ number of processors, and

■ disk storage

- IaaS shows all the benefits of hardware virtualization:

○ workload partitioning

■ application isolation

■ sandboxing, and

■ hardware tuning

- HaaS allows better utilization of the IT infrastructure and provides a more safe environment for executing third party applications.
- Figure 2 shows a total view of the components setup an Infrastructure-as-a-Service.
- It is possible to identified three principal layers:

○ the physical infrastructure

○ the software management infrastructure

○ the user interface

- At the top layer the user interface allow to access the services exposed by the software management infrastructure. This types of an interface is generally based on Web technologies:

- ○ Web services
- ○ RESTful APIs, and
- ○ mash-ups

- These automation allow applications or final users to access the services exposed by the underlying infrastructure.
- A central role of the scheduler, is in charge of allocating the execution of virtual machine instances. The scheduler communicates with the other components that perform a variety of tasks.

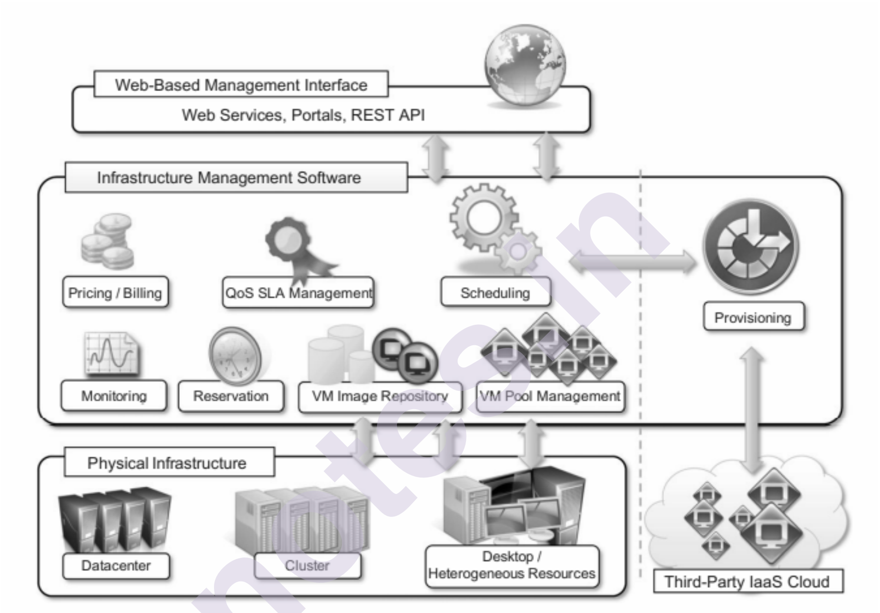


Fig.2. Infrastructure-as-a-Service reference implementation

3.3 ENTERPRISE ARCHITECTURE

- The 'enterprise architecture' function within enterprise IT has evolved to manage the complexities of an ever-changing technical environment.
- In the process enterprise architects found it useful to maintain a description of all of an enterprise's software applications, how they fulfill business needs, how they are implemented technically and how they communicate with each other.

ENTERPRISE DATA AND PROCESSES

- The first step is identifying and naming each process, and identifying business events that mark its start and end.
- Enterprise processes can often be classified as 'vertical' or 'horizontal'.

- ○ Vertical processes typically operate within a single organizational function, such as sales or accounting, manage a cohesive information set pertaining to that function, and are typically supported by software packages or systems dedicated to that department. 'Prospect to order', for example, is a vertical process limited to the sales function.
- ○ Horizontal processes, on the other hand, cut across functional units; 'order to cash' is a horizontal process since it spans sales, production and finance.

ENTERPRISE COMPONENTS

- The term 'component' has traditionally been taken to mean a 'software sub-system that has well-defined interfaces which can be used independently of its internal implementation' [50]. Structuring software into components drives modularity in software development and makes it easier to evolve a large system by incrementally replacing its components over time.

  Figure 4.1 shows this component view of enterprise architecture.

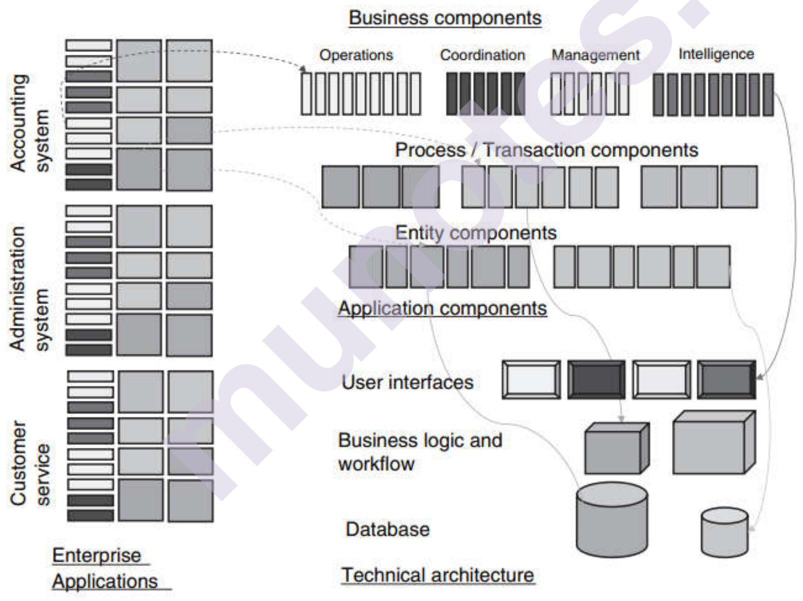


Fig 3. Enterprise components

APPLICATION INTEGRATION AND SOA

- Application integration is the often perceived need for a unified view of data residing in disparate application systems, say for end-to-end process monitoring, real-time decision support, or for data warehousing and business intelligence.

There are a number of mechanisms that applications can use to communicate with each other at different 'levels'

1. **Data level integration:** direct data transfer using batch programs or on-line exchange using database triggers
2. **API level integration:** applications publish API libraries that are used by other applications to access their data
3. **Service-method-level integration:** applications publish services using say, web service protocols, in an organized manner so that many different applications can use a particular service
4. **User interface level integration:** applications publish mashup APIs that are used to provide a common user interface to functionality from many applications
5. **Workflow level integration:** tasks performed in one application lead to work items being created in others, thereby driving the flow of work in a business process

ENTERPRISE TECHNICAL ARCHITECTURE

The definition and management of standards defining the technical architecture, tools and technical components used in an enterprise:

1. Unformity

- Cost and simplicity is the motivation for standardizing technical components such as application servers, databases and integration tools used in an enterprise.
- This 'uniformity' approach is clearly sound when most of the integration between disparate technologies is carried out by the enterprise itself.

2. Network and data security

- Security issues arising from technology choices are also part of the enterprise architecture function. While considering the option of cloud deployment the question of security of data that will reside outside the enterprise data center is a common concern
- Network security, or rather securing applications on the network, is a more serious concern when considering cloud deployment.

3. Implementation architectures and quick-wins

- One of the aspects enterprise architects pay attention to are the 'implementation architectures' required for adopting any new technology.
- These include the people skills required, along with development, testing and deployment tools needed, as well as the impact on business-continuity and disaster-recovery environments.

3.4 TYPES OF CLOUDS

- Clouds compose the primary outcome of cloud computing.
- They are a type of parallel and distributed system, physical and virtual computers conferred as a unified computing asset.
- Clouds build the infrastructure on top of which services are implemented and delivered to customers. Such infrastructures can be of different types and provide useful information about the nature and the services offered by the cloud.
- A more convenient classification is given according to the administrative domain of a cloud: It identifies the boundaries within which cloud computing services are implemented, provides hints on the underlying infrastructure take on to support such services, and qualifies them. It is then possible to evolve four different types of cloud:
- Public clouds.
  - The cloud is open to the wider public.
- Private clouds.
  - The cloud is executed within the private property of an institution and generally made accessible to the members of the institution
- Hybrid clouds.
  - The cloud is a combination of the two previous clouds and most likely identifies a private cloud that has been augmented with services hosted in a public cloud.
- Community clouds.
  - The cloud is distinguished by a multi administrative domain consisting of different deployment models (public, private, and hybrid).

Public clouds

- Public clouds account for the first expression of cloud computing.
- They are an awareness of the canonical view of cloud computing in which the services provided are made available to anyone, from anywhere, and at any time through the Network.
- From a structural point of view they are a distributed system, most likely composed of one or more data centers connected together, on top of which the specific services offered by the cloud are implemented.
- Any customer can easily agree with the cloud provider, enter her username and password and billing details.

- They offer results for minimizing IT infrastructure costs and serve as a viable option for handling peak loads on the local infrastructure.
- They are used for small enterprises, which are able to initiate their businesses without large up-front investments by completely relying on public infrastructure for their IT needs.
- A public cloud can recommend any type of service such as infrastructure, platform, or applications. For example, Amazon EC2 is a public cloud that provides infrastructure as a service; Google AppEngine is a public cloud that provides an application development platform as a service; and SalesForceservice.com is a public cloud that provides software as a service.

Private clouds

- private clouds, which are the same as public clouds, but their resource provisioning model is restricted within the boundaries of an organization.
- Private clouds have the benefit of keeping the core business operations in house by depending on the existing IT infrastructure and reducing the cost of maintaining it once the cloud has been set up.
- The private cloud can provide services to a different range of users.
- private clouds is the possibility of testing applications and systems at a comparatively less price rather than public clouds before implementing them on the public virtual infrastructure.
- The main advantages of a private cloud computing infrastructure:

1. Customer information protection.
2. Infrastructure ensuring SLAs.
3. Compliance with standard procedures and operations.

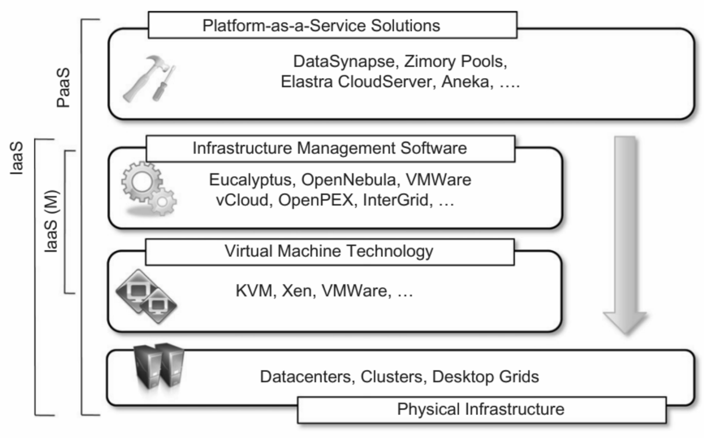

Platform-as-a-Service Solutions
DataSynapse, Zimory Pools, Elastra CloudServer, Aneka, ….
PaaS
IaaS
IaaS (M)
Infrastructure Management Software
Eucalyptus, OpenNebula, VMWare vCloud, OpenPEX, InterGrid, …
Virtual Machine Technology
KVM, Xen, VMWare, …
Datacenters, Clusters, Desktop Grids
Physical Infrastructure

Fig.4. Private clouds hardware and software stack.

Hybrid clouds

- A hybrid cloud could be an attractive opportunity for taking advantage of the best of the private and public clouds. This shows the development and diffusion of hybrid clouds.
- Hybrid clouds enable enterprises to utilize existing IT infrastructures, maintain sensitive information within the area, and naturally increase and reduce by provisioning external resources and releasing them when they're no longer needed.
- Figure 5 provides a general overview of a hybrid cloud:

○ It is a heterogeneous distributed system resulting from a private cloud that integrates additional services or resources from one or more public clouds.

○ For this reason they are also called heterogeneous clouds.

○ Hybrid clouds address scalability issues by leveraging external resources for exceeding

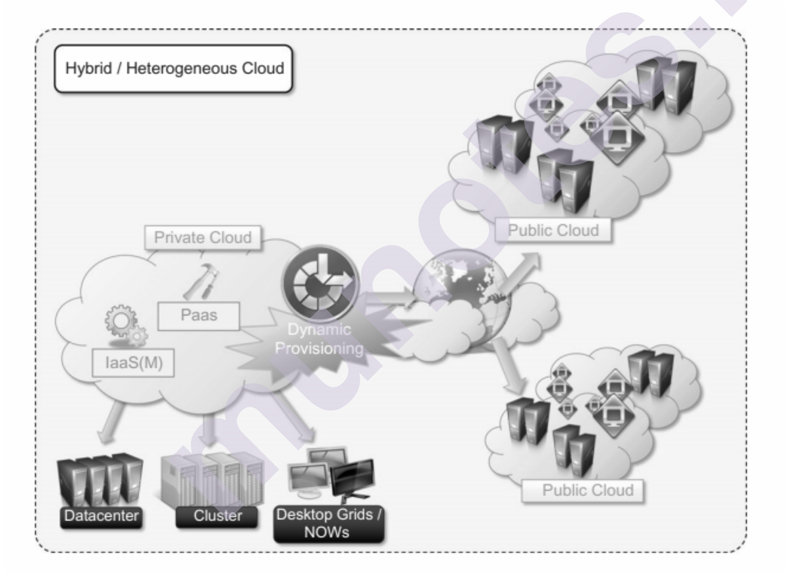


Fig.5. Hybrid/heterogeneous cloud overview.

Community clouds

- Community clouds are distributed systems created by integration of services of different clouds to handle the specific requirement of an industry, a community, or a business sector.
- The National Institute of Standards and Technologies (NIST) characterizes community clouds as follows:

○ The infrastructure is shared by different organizations and supports a certain community that has shared concerns.

- ○ It may be controlled by the organizations or a third party and may exist on premise or off premise.

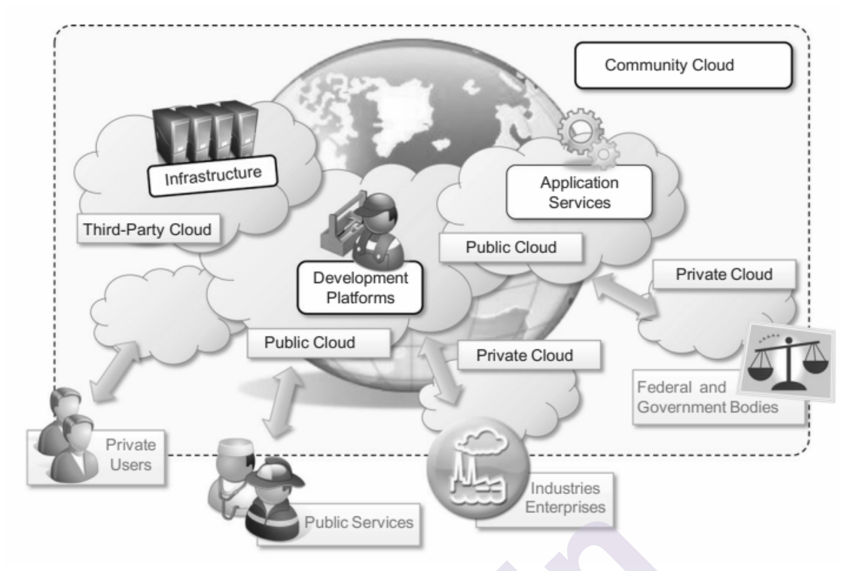


Fig. 6 general view community clouds

- ● Figure 6 shows a view of the usage scenario of community clouds, jointly with reference architecture.
- ● The users of a distinct community cloud fall into a well identified community, sharing the same concerns or needs such as government bodies, industries, or even simple users, but all of them concentrate on the same problem for their interaction with the cloud.
- ● Community clouds are the services that are generally delivered within the institution that owns the cloud.
- ● Candidate district for community clouds are as follows:
- ○ Media industry
- ■ Where Companies are finding low-cost, agile, and simple solutions to better the efficiency of content production.
- ■ Most media involve an expanded ecosystem of partners.
- ■ Community clouds can provide a shared environment where services can ease business to business participation and give the horsepower in terms of aggregate bandwidth, CPU, and storage required to efficiently support media production.
- ○ Healthcare industry.
- ■ In the healthcare industry, there are different storyline in which community clouds are used.

- ■ Community clouds provide a global platform on which to share information and knowledge without telling sensitive data maintained within the private infrastructure.
- ■ The naturally hybrid deployment model of community clouds supports the storing of patient data in a private cloud while using the shared infrastructure for noncritical services and automating processes within hospitals.

○ Energy and other core industries.

- ■ These industries concern different service providers, vendors, and organizations, a community cloud can give the right type of infrastructure to create an open and upright market.

○ Public sector.

- ■ The public sector can limit the adoption of public cloud offerings.
- ■ governmental processes involve several institutions and agencies
- ■ Aimed at providing strategic solutions at local, national, and international administrative levels.
- ■ involve business-to-administration, citizen-to-administration, and possibly business-to-business processes.
- ■ Examples, invoice approval, infrastructure planning, and public hearings.

○ Scientific research.

- ■ THis is an interesting example of community clouds.
- ■ In this point, the common interest in handling and using different organizations to split a large distributed infrastructure is scientific computing.

The Advantages of community clouds:

- ● Openness.

Clouds are open systems in which fair competition between different solutions can occur.

- ● Community.

Providing resources and services, the infrastructure turns out to be more scalable.

- ● Graceful failures.

There is no single provider & vendor in control of the infrastructure, there is no chance of a single point of failure.

- Convenience and control.

There is no dispute between convenience and control because the cloud is shared and owned by the community, which makes all the decisions through a collective representative process.

- Environmental sustainability.

These clouds tend to be more organic by increasing and shrinking in a symbiotic relationship to support the demand of the community.

3.5 SUMMARY

- Three service models. Software-as-a-Service (SaaS), Platform-as-a-Service (PaaS), and Infrastructure-as-a-Service (IaaS).
- Four deployment models. Public clouds, private clouds, community clouds, and hybrid clouds.
- Cloud computing has been rapidly adopted in industry, there are several open research challenges in areas such as management of cloud computing systems, their security, and social and organizational issues.

3.6 REFERENCE FOR FURTHER READING

- Enterprise Cloud Computing Technology, Architecture, Applications, GautamShroff, Cambridge University Press, 2010
- Mastering In Cloud Computing, RajkumarBuyya, Christian Vecchiola And ThamariSelvi S, Tata Mcgraw-Hill Education, 2013
- Cloud Computing: A Practical Approach, Anthony T Velte, Tata Mcgraw Hill, 2009

3.7 UNIT END EXERCISES

1. What does Infrastructure-as-a-Service refer to?
2. What are the main characteristics of a Platform-as-a-Service solution?
3. What does the acronym SaaS mean? How does it relate to cloud computing?
4. Classify the various types of clouds.
5. Give an example of the public cloud.

❖❖❖❖

4

CLOUD TECHNOLOGIES-I

Unit Structure

4.0 OBJECTIVES

The objectives of this chapter are:

- To understand the concept of cloud computing
- To appreciate the evolution of cloud from the existing technologies
- To have knowledge on the various issues in cloud computing
- To be familiar with the lead players in cloud
- To appreciate the emergence of cloud as the next generation computing paradigm

4.1 INTRODUCTION

The provision of computing resources as a service is known as cloud computing. Going to the cloud essentially means that a provider rather than the end user owns and manages the resources.Through their internet-connected PCs, smartphones, tablets, and wearables, users of cloud computing technologies can access storage, files, applications, and servers. Data is processed and stored by cloud computing services away from end users.

In its simplest form, cloud computing refers to the capability of storing and accessing data and software through the internet as opposed to a hard drive. This implies that organisations of any size may use robust software and IT infrastructure to grow, get leaner, and become more flexible while competing with much larger firms. Contrary to traditional hardware and software, cloud computing enables companies to stay on the cutting edge of innovation without having to make significant financial commitments in their own equipment purchases, upkeep, and repairs.

4.2 CLOUD COMPUTING PLATFORMS

We will discuss the main platforms from Amazon, Google, and Microsoft in this chapter, summarising the services they offer from the viewpoint of end users.We will also discuss more cloud service providers software& resources used for set up.

4.2.1 Infrastructure as a service: Amazon EC2

Through an automated web-based administration panel, they offer IaaS, which allows users to virtually provision computing infrastructure within fraction of time. Elastic Compute Cloud (EC2) is one of several IaaS services that make up the platform.

Facilities offered are depicted in Figure 1. The picture depicts a very vast network of servers that are used to implement these services as dashed boxes. Users of the Elastic Compute Cloud service can use dedicated virtual machines with the end-user being unaware of the physical server's specifics, like as its location, capacity, etc. Users create PKI1 key pairs through the administration dashboard, which they can use to securely access these virtual servers online.

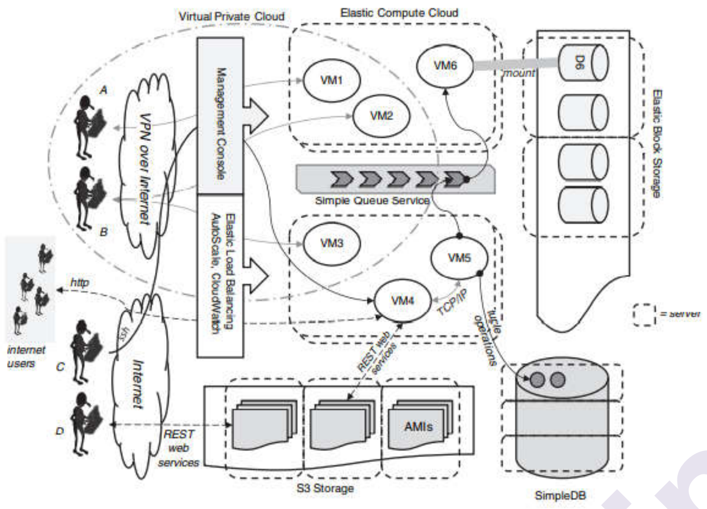


Figure 1: Amazon infrastructure cloud

In Figure 1, person C configures VM4 using the management interface. For a Linux server, ssh is used to access it. For a Windows server, remote desktop is used. When provisioning a server, users can select from a variety of VM (Amazon machine images, or AMIs). All AMIs are used to boot the required configuration of virtual servers.The user's account is charged hourly depending on actual use, or the amount of time the server is operational. The AMI that is utilised and the server capacity that is selected while provisioning affects the costs.Cloud users have full control over these servers because they have root or administrator access to them.

Numerous servers that can connect with one another across the quick internal network of the Amazon cloud can be provisioned and accessed by users. In Figure 1, for instance, user C has also provisioned VMs 5 and 6 in addition to VM4. VM5 may be a database server if VM4 is a web server, and the two of them can communicate using TCP/IP across the internal cloud network. As a database server, VM5 must be able to save and fetch information. With the help of the SimpleDBfacility, key-value pairs can be quickly saved and retrieved in an object store. The distinction between SimpleDB and relational databases must be made.

Businesses looking to use cloud computing must also solve corporate IT security issues. One of these issues, network security, is crucial: Firewalls, proxies, intrusion detection systems, and other security measures are typically used to protect an organization's computing resources. Obviously, safety demands that VM operating in the cloud be safeguarded, employing similar guidelines & security measures. With the help of a VPN, servers could be linked to an organization's network utilising the cloud service offered by Amazon EC2 (virtual private network).Users A and B, for instance, connect to virtual servers VM1, VM2, and VM3 via a

VPN that is active through online platform in Figure 1. The network operations centre for the company then manages the private IP addresses assigned to these machines.

4.2.2 Platform as a service: Google App Engine

Platform as a Service (PaaS), sometimes known as Google App Engine, is a service provided by Google. A PaaS solution conceals the real execution environment from customers. Instead software platform and an SDK are offered, allowing customers to create and upload apps to the cloud. The PaaS platform is in charge of operating the applications, handling external service requests and scheduling jobs that are part of the application.A PaaS platform can distribute application servers among users that want lower capacity and automatically scale resources allotted to apps that encounter high loads by hiding the real execution servers from the user.

The Google App Engine user interface is shown in Figure 2. Users upload files relevant to their codein either Java or Python.

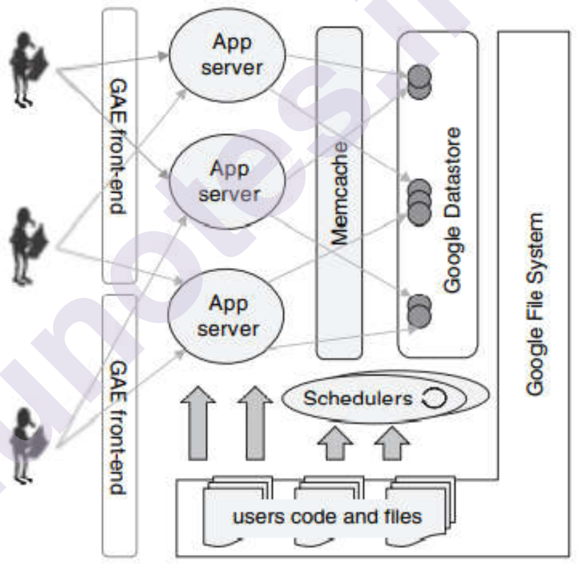


Figure 2: Google App Engine

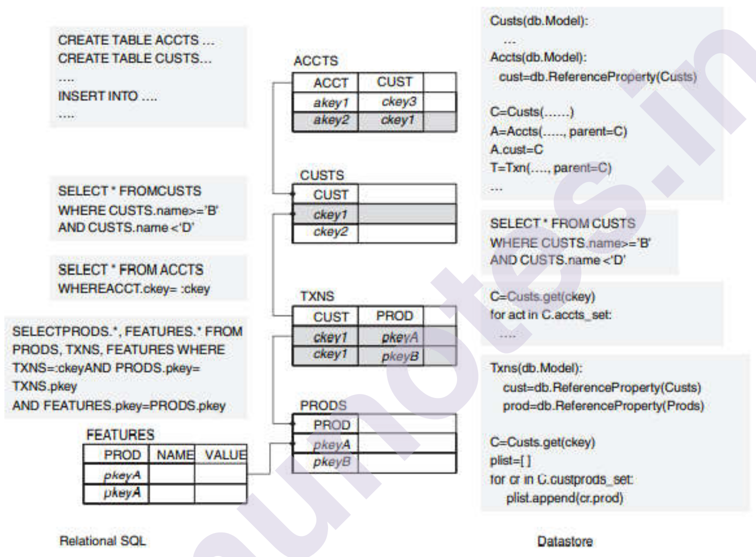
A PaaS is utilised and made available across the worldthroughout a day, but it is only charged when accessed (or if batch jobs run). Additionally, the development and testing of applications in Google App Engine is free, subject to use restrictions; charges only apply when a sufficient number of requests actually visit facility.Installed programmes that aren't used just require storage for their code and data and don't take any CPU time.

In Google's data centres, a sizable number of web servers that process requests from end users all around the world serve GAE apps. The web servers process these requests by loading GFS code into memory.There is no assurance that the same server will fulfil more than one request, even if they originate from the same HTTP session. Every access to a specific application is handled by any servers.Applications can also indicate which operations should be performed in batches by a scheduler.

This architecture makes it possible for programmes to grow naturally as load rises, but it also makes it difficult for application code to rely on in-memory data. To partially address this issue, Memcacheis provided: It is specifically used to implement HTTP sessions so that requests from the same session can often retrieve their session data even if they are sent to separate servers.

4.2.2.1 Google Datastore

It is a database where applications data are kept.They define types (referred to as "kinds") using the Datastore, and their instances (referred to as "entities") can be distributedly stored on the GFS file system. There are significant differences as shown in Fig 3.



```
CREATE TABLE ACCTS ...
CREATE TABLE CUSTS...
....
INSERT INTO ....
....
```

```
SELECT * FROMCUSTS
WHERE CUSTS.name>='B'
AND CUSTS.name <'D'
```

```
SELECT * FROM ACCTS
WHEREACCT.ckey= :ckey
```

```
SELECTPRODS.*, FEATURES.* FROM
PRODS, TXNS, FEATURES WHERE
TXNS=:ckeyAND PRODS.pkey=
TXNS.pkey
AND FEATURES.pkey=PRODS.pkey
```

```
Custs(db.Model):
   ...
Accts(db.Model):
  cust=db.ReferenceProperty(Custs)

C=Custs(......)
A=Accts(....., parent=C)
A.cust=C
T=Txn(...., parent=C)
...
```

```
SELECT * FROM CUSTS
WHERE CUSTS.name>='B'
AND CUSTS.name <'D'
```

```
C=Custs.get(ckey)
for act in C.accts_set:
  ....
```

```
Txns(db.Model):
   cust=db.ReferenceProperty(Custs)
   prod=db.ReferenceProperty(Prods)

C=Custs.get(ckey)
plist=[ ]
for cr in C.custprods_set:
    plist.append(cr.prod)
```

Figure 3: Google datastore

All entities of a "kind" need not have the same properties. Instead, every entity can have extra characteristics added to it. This function is very helpful when it is impossible to predict all of the possible attributes. A model for storing 'products' of various entities must permit everycategoryfor unique capacity of attributes. This would most likely be implemented in a relational model utilising a distinct FEATURES table, reflected on the lower left of Fig 3.This table ('type') is not necessary when using the Datastore; rather, each product object may be given a unique set of characteristics during runtime.

The Datastore supports basic conditional queries, like the one in Figure 4.3 that returns all customers with names that fall within a certain lexical range. With a few limitations, the query syntax is substantially similar as SQL. A query that included screened clients based on, prohibited in GQL & permitted in SQL since all unequal conditions in a query must be on a single property.

Here GFS distributed file system and several servers are used to keep the objects (entities) of all GAE apps. From the user's perspective, it's crucial to make sure that information is (a) properly fetched & (b) adapted despite being shared with numerous other users in a distributed storage system.The Datastore offers a method for classifying entities into hierarchical groups that can be utilised for both of these objectives.

4.2.2.2 Amazon SimpleDB

SimpleDB is similarly a database where "domains" are equivalent to "kinds," and "items" are equivalent to entities. SimpleDB domain queries support conditions on any number of attributes, including inequality criteria. Additionally, joins are not allowed, much like in the Google Datastore. However, unlike Google Datastore, SimpleDB does not support object associations.

It's crucial to remember that, like GFS, all information in this case is copied for layoff. Informationpropagate to minimum one replica, thanks to SimpleDB's "eventual consistency" concept. This can give the appearance of consistency as it's not always the case that reading something right after after writing it will produce the same outcome. In contrast, writes in Google Datastore only succeed once all replicas have been updated; this prevents consistency but slows down writing.

4.2.3 Microsoft Azure

Azureservice lately made available to the public on a commercial basis, while a community preview beta has been accessible to the public.

Azure is a PaaS service, much like Google App Engine. Microsoft development tools (such as Visual Studio and the supported languages, C#, Visual Basic, ASPs, etc.) are used by developers to create applications; very similar to how they are developed and deployed using Google App Engine's SDK.Similar to Amazon S3, Azure also supports the storage of unrestricted files and objects. It also offers a queuing service like Amazon SQS.

The user's perspective of Microsoft Azure is shown in Figure 4. The Windows Server operating system is installed on each of the virtual computers (referred to as instances) that run the application code when it is delivered to Azure.Users can indicate the nos of examples a specific programme will use, but they have no control over when instances boot or how long they remain active. As a result, compared to Google App Engine, Azure offers control level but not the same extent as an IaaS provider like Amazon EC2.

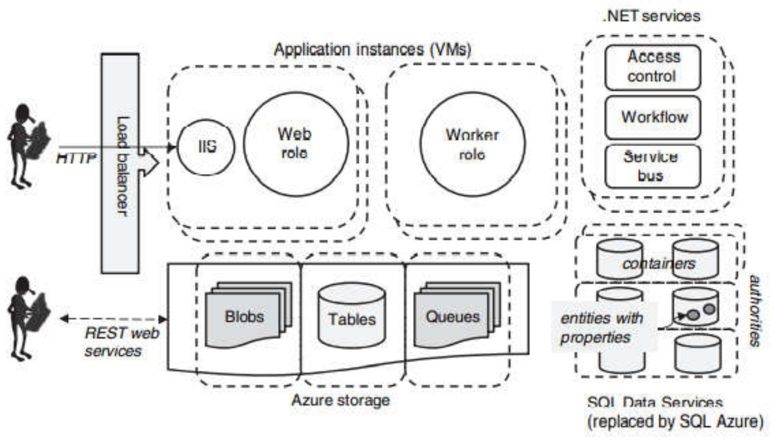


Figure 4: Microsoft Azure

On Azure, program is implemented as a worker or web role.A server that is built into runs code in role instances. Blobs, queues, and non-relational tables (similar to SimpleDB) can all be stored in Azure data storage. Additionally, SQL Data Services in the Azure beta edition included a different database-storage system. Authorities are Microsoft-owned data centres where SQL is deployed on a no of groups. Similar to SimpleDB or Google Datastore, each container contains entities that have attributes, and the types and numbers of properties for each item can vary. A SQL Azure database can only be less than 10 GB in size. Multiple virtual database instances must be provisioned in the event that higher data volumes need to be stored. Furthermore, since joins on larger data sets must be implemented in application code because cross-database searches are not allowed.

Microsoft Azure also offers what are referred to as.NET services in addition to compute and storage capabilities. These include web-based workflow orchestration services that may be customised offer across enabling globally published service end points. They are deployed on Microsoft's midware technologies just like SQL Data services and SQL Azure.

They have benefit of a sizable base apps that are currently in use within businesses, as opposed to Google's PaaS service.It is expected to get even simpler in the future to move current Microsoft applications to Azure. Therefore, it's possible that Microsoft Azure overtakes Google's App Engine as the preferred PaaS platform, at least among large businesses.

4.3 WEB SERVICES

The backbone for web-based cloud computing services is the set of protocols that make up the internet: HTTP, DNS, and TCP/IP. This chapter delves at three crucial web-based technologies that have proved crucial in enhancing the applicationsadaptability.

4.3.1 Web Services: SOAP and REST

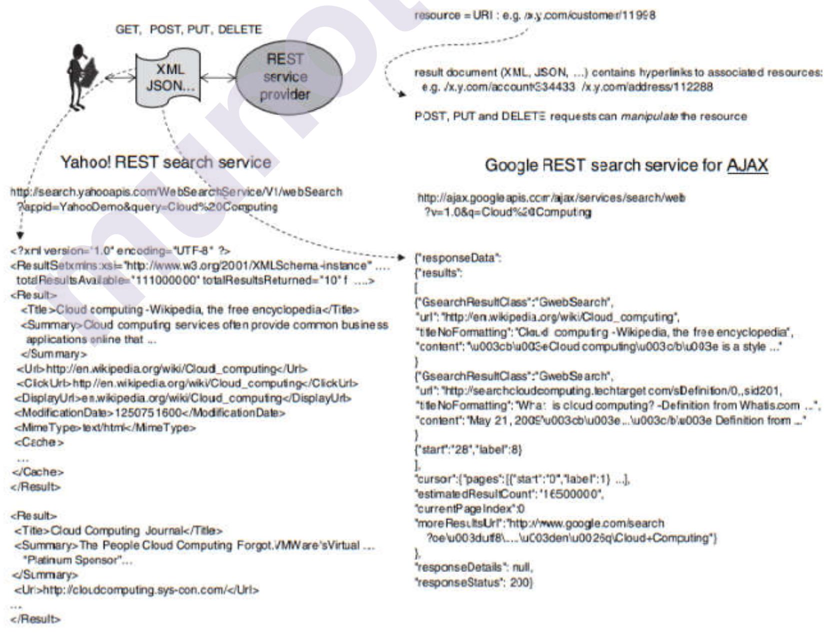
In this we demonstrate outwork of both as well as a comparison in creating cloud-based offerings.

4.3.1.2 REST web services

The HTML-based web itself is an example of the Representational State Transfer (REST) architectural style, which was initially developed for use in large-scale systems with distributed resources.

These are nothing more than HTTP queries to URIs, utilising exact 4 methodologies that the HTTP protocol permits: GET, POST, PUT, and DELETE. Each URI uniquely detects database record.This service delivers the customer record in XML format. Links to related recordslook somewhat like this: http://x.y.com/account/334433, if the record has links (foreign keys) to related entries.As an alternative, you can directly access these connections by using data handled in this "RESTful" fashion in order to retrieve data.

The above example, as well as two actual instances employing Yahoo! and Google, who both offer the service are used to illustrate REST web services in Figure 6. Observe that URLs contain parameters, this facility description vary from paradigm, which stipulates that data should only be associatedthrough URIs.


GET, POST, PUT, DELETE
XML JSON...
REST service provider
resource = URI : e.g. /x.y.com/customer/11998
result document (XML, JSON, ...) contains hyperlinks to associated resources:
e.g. /x.y.com/account/334433 /x.y.com/address/112288
POST, PUT and DELETE requests can *manipulate* the resource
Yahoo! REST search service
Google REST search service for AJAX

```
http://search.yahooapis.com/WebSearchService/V1/webSearch
  ?appid=YahooDemo&query=Cloud%20Computing

<?xml version="1.0" encoding="UTF-8" ?>
<ResultSetxmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" ....
  totalResultsAvailable="11100000" totalResultsReturned="10" f ....>
<Result>
   <Title>Cloud computing -Wikipedia, the free encyclopedia</Title>
   <Summary>Cloud computing services often provide common business
     applications online that ...
   </Summary>
  <Url>http://en.wikipedia.org/wiki/Cloud_computing</Url>
  <ClickUrl>http://en.wikipedia.org/wiki/Cloud_computing</ClickUrl>
 <DisplayUrl>en.wikipedia.org/wiki/Cloud_computing</DisplayUrl>
 <ModificationDate>1250751600</ModificationDate>
 <MimeType>text/html</MimeType>
 <Cache>
  ...
 </Cache>
</Result>

<Result>
 <Title>Cloud Computing Journal</Title>
 <Summary>The People Cloud Computing Forgot.VMWare'sVirtual ...
   "Platinum Sponsor"...
 </Summary>
 <Url>http://cloudcomputing.sys-con.com/</Url>
...
</Result>
```

```
http://ajax.googleapis.com/ajax/services/search/web
  ?v=1.0&q=Cloud%20Computing

{"responseData":
{"results":
[
{"GsearchResultClass":"GwebSearch",
"url":"http://en.wikipedia.org/wiki/Cloud_computing",
"titleNoFormatting":"Cloud computing -Wikipedia, the free encyclopedia",
"content":"\u003cb\u003eCloud computing\u003c/b\u003e is a style ..."
}
{"GsearchResultClass":"GwebSearch",
"url":"http://searchcloudcomputing.techtarget.com/sDefinition/0,,sid201,
"titleNoFormatting":"What is cloud computing? -Definition from Whatis.com ...",
"content":"May 21, 2009\u003cb\u003e...\u003c/b\u003e Definition from ..."
}
{"start":"28","label":8}
],
"cursor":{"pages":[{"start":"0","label":1} ...],
"estimatedResultCount":"1650000",
"currentPageIndex":0
"moreResultsUrl":"http://www.google.com/search
   ?oe\u003dutf8\....\u003den\u0026q\Cloud+Computing"}
},
"responseDetails": null,
"responseStatus": 200}
```

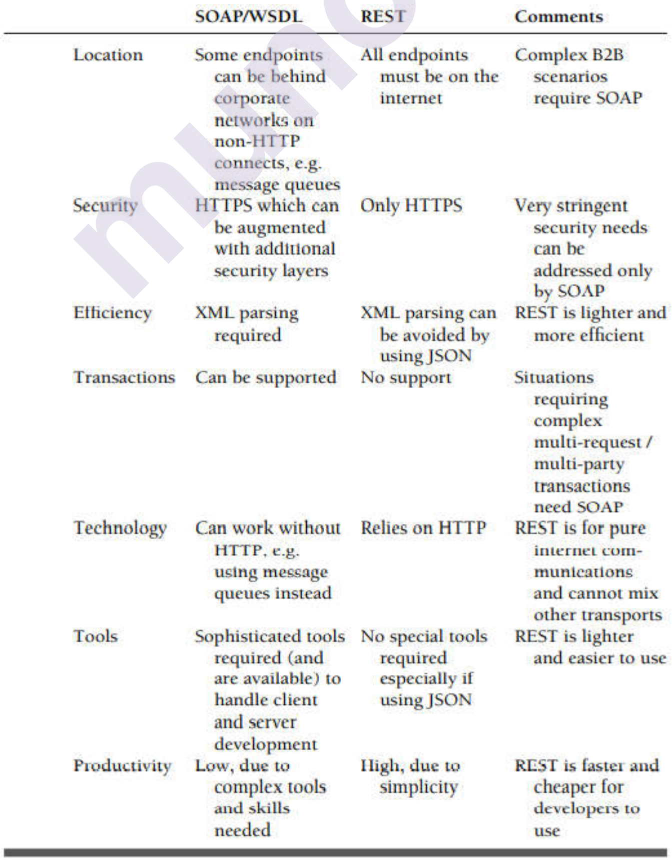
Figure 5: REST Web services

4.3.2 SOAP VERSUS REST

Many debates about encoding makes it difficult to provide a web service's semantics such that it may be readily and broadly understood, potentially allowing multiplefacilitators to provide similar opportunity.The best illustration is search. REST has the capacity to achieve standardisation:Multiple providers could make this available if e.g.,the REST std to searching was http://provider-URL>/query-string>. The reply doc in XML then be self-descriptive adhering publicly specified top-level schema while making use of provider-specific name spaces when necessary. Our examples make it clear right away how much easier it is to create and use REST-based services than it is to use the more complicated SOAP/WSDL method.Furthermore, by adopting alternatives like JSON, REST can avoid costly XML parsing. In light of this, our opinion is that REST should be the preferred option from both a simplicity and an efficiency standpoint, with the argument for using SOAP/WSDL needing to be expressly presented based on the circumstances.

Table 1 compares these on six different criteria, including the location of the servers that can provide the service, the level of interaction security, whether transactions can be supported, how reliant on HTTP technology the protocol is, the amount expected productivity using each. This examination leads us to the conclusion that they are easier, effective& less expensive to create than SOAP for the majority of requirements. It is clear that SOAP is being replaced by REST, especially in cloud environments: The REST API utilising JSON has mostly taken the place of the outdated Google SOAP service.Although SOAP and REST APIs are both published by Amazon Web Services, SOAP APIs are hardly ever used. In the near future, REST will also gain tools to address more complicated functionality, such as transactions.

Table 1: SOAP/WSDL Vs REST



	SOAP/WSDL	REST	Comments
Location	Some endpoints can be behind corporate networks on non-HTTP connects, e.g. message queues	All endpoints must be on the internet	Complex B2B scenarios require SOAP
Security	HTTPS which can be augmented with additional security layers	Only HTTPS	Very stringent security needs can be addressed only by SOAP
Efficiency	XML parsing required	XML parsing can be avoided by using JSON	REST is lighter and more efficient
Transactions	Can be supported	No support	Situations requiring complex multi-request / multi-party transactions need SOAP
Technology	Can work without HTTP, e.g. using message queues instead	Relies on HTTP	REST is for pure internet communications and cannot mix other transports
Tools	Sophisticated tools required (and are available) to handle client and server development	No special tools required especially if using JSON	REST is lighter and easier to use
Productivity	Low, due to complex tools and skills needed	High, due to simplicity	REST is faster and cheaper for developers to use

4.4 AJAX: ASYNCHRONOUS RICH INTERFACES

Traditional web applications send a series of HTTP GET and POST calls to their server components, refreshing the HTML page in the browser as they do so. Client-side JavaScript, or in-browser JavaScript, is only used for certain cases including animations, hiding or revealing page portions, and field validations. The browser is largely inactive barring any such modifications.

Without refreshing their main HTML page, JavaScript programmes that perform calls using AJAX.A "richer" user experience can be achieved by multiplexing client-side processing of user activities with heavy server-side processing to reduce the overall response time. Additionally, client-side Script can issue requests to other internet services in addition to the main web server, facilitating the integration of applications within browsers.

From the standpoint of S/W design, applications are not clients. Instead, they perform a significant amount of processing on the client, making use of the desktop's processing capacity exactly like client-server programmes did. Recall that while employing the client-server architecture, there was a chance that the interface would be combined in the code, increasing the difficulty of maintaining such software. Identical issues come up when employing the AJAX paradigm.

In contrast to conventional online applications, AJAX applications—also known as "rich internet applications" (RIA)—show how they operate in Figure 7.The remaining portions of the user interface are loaded along with a base HTML page. The "rich" user interface produced by this JavaScript software frequently resembles a classic client-server application. REST web services are used to make asynchronous requests for data from the server, which the server responds to with JSON structures that the JavaScript code running in the browser can use right away.

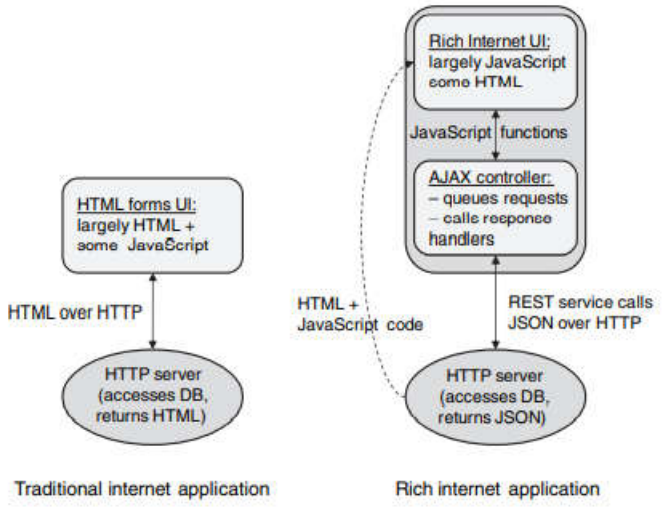

HTML forms UI: largely HTML + some JavaScript
HTML over HTTP
HTTP server (accesses DB, returns HTML)
Traditional internet application
Rich Internet UI: largely JavaScript some HTML
JavaScript functions
AJAX controller: – queues requests – calls response handlers
HTML + JavaScript code
REST service calls JSON over HTTP
HTTP server (accesses DB, returns JSON)
Rich internet application

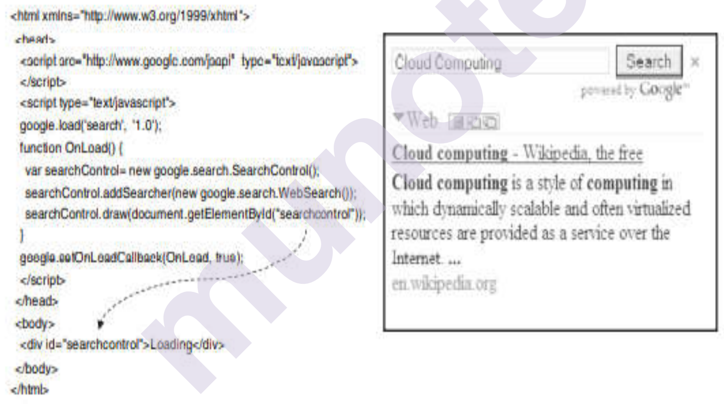
Figure 6: Rich internet application with AJAX

AJAX makes it feasible to create user interfaces that are entirely browser-based and extremely interactive. With the use of this method, SaaS apps can start givinginsights resembling programmes, often used within businesses. This makes SaaS products more appealing to business customers. Additionally, rather than relying on more complicated strategies.Furthermore, unlike server-side integration, which must be carried out by corporate IT, simple JavaScript-based integrations are frequently carried out by business units themselves, in a way similar to how Dev 2.0 platforms enable end users to create straightforward applications.

4.5 MASHUPS: USER INTERFACE SERVICES

Mashups raise the bar of integration by incorporating both the service and the presentation layer for remote services.

Figure 8 uses the Google search service to show mashups. A developer just needs to acknowledgeand make application using codeaccesses through browser in order to display a Google search bar. This code offers a "class" called google that has as its methods the Google AJAX API. After the HTML page has loaded, the methods are calledfor construction. Observe that user cannot see an AJAX controller or a REST service; all of this is concealed within the API methods.Since there doesn't need serialisation& thus acceptable in various services.



```
<html xmlns="http://www.w3.org/1999/xhtml">
 <head>
  <script src="http://www.google.com/jsapi" type="text/javascript">
  </script>
  <script type="text/javascript">
  google.load('search', '1.0');
  function OnLoad() {
   var searchControl= new google.search.SearchControl();
   searchControl.addSearcher(new google.search.WebSearch());
   searchControl.draw(document.getElementById("searchcontrol"));
  }
  google.setOnLoadCallback(OnLoad, true);
  </script>
 </head>
 <body>
  <div id="searchcontrol">Loading</div>
 </body>
</html>
```

Cloud computing - Wikipedia, the free
Cloud computing is a style of computing in which dynamically scalable and often virtualized resources are provided as a service over the Internet. ...
en.wikipedia.org

Figure 7: Mashup example

From the user's point of view, mashups simplify the process of consuming web services. The original service request not necessarily be REST; it can be a customised communication.

However, mashups' requirement to download and run foreign code raises legitimate security issues, particularly in business settings.In contrast to ActiveX components, which after installation have virtually full access to the desktop, JavaScript code often has access to only the browser and the network, giving the impression that there is no significant security issue. However, in the future, this might no longer be the case: For instance, Google Gears is a framework that allows apps to run offline by caching data on the client computer.Although less significant than ActiveX controls, this poses a potential security risk.

4.6 MULTI-TENANT SOFTWARE

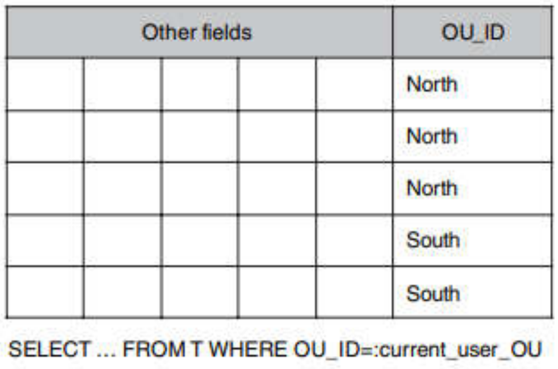
Applications are typically created for a single organisation, and similarly, enterprise software is also created so that it may be independently deployed in each customer's data centre. These programmes often access and create data that belongs to a single company. Multi-tenancy refers to the requirement for a solo code to execute on the information of several clientsin hosted SaaS platforms. This section looks at various approaches to incorporate multi-tenancy in S/W.

Here, we look towards alternatives to employing virtual machines at the system level to accomplish multi-tenancy, such as application software architecture. As a result, this category is sometimes called as application-level virtualization. Goal of both multi-tenancy and virtualization is resource sharingkeeping users apart of one another.

4.6.1 Multi-entity support

Large globally dispersed enterprises frequently need their systems to handle various organisational quantities in a sequential and well managed form long before ASPs and SaaS. Assume a bank with numerous branches that needs to specific S/W to centralised one. It is obvious that the programme can't be utilised to access information from multiple sites: Users for instance, must only view information relevant to its site.If the system needed to be improved, for as by adding a new field, the update would have to be supported by each site at the same instance supporting modifications. These prerequisites are nearly identical to those for multi-tenancy!There are additional requirements in a multi-entity scenario, such as the need to grant a subset of users or a selection of sites based on rank in an organisational behaviour. The advantages of data centralization might not be achieved without supporting some global processing.

Figure 9 illustrates modifications that must be done to an application in order to support fundamental multi-entity features and limit user access to data. Each record has a field (OU_ID) that identifies the organisational unit to which each data record belongs.A situation mentioning managed information based on user who is logged in must be added to each database query. Support for multi-tenancy can be provided using an identical technique, with the customer to who's the data records belong now represented by the OU ID.

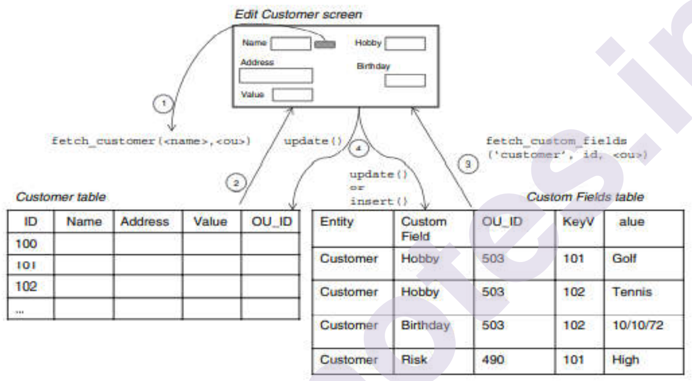


Other fields					OU_ID
					North
					North
					North
					South
					South

```
SELECT ... FROM T WHERE OU_ID=:current_user_OU
```

Figure 8: Multi-entity implementation

The single schema model was widely used in early SaaS deployments, particularly by those who created their SaaS systems from scratch. The ability to update application functionality, for as by adding a field, for all customers at once is one benefit of the single schema structure. However, there are drawbacks as well: The single schema technique requires extensive re-structuring of the application code when re-engineering an established application. This cost may be too high for a large software programme with millions of lines of code. Additionally, while changes employing a single schema structure to accommodate customer-specific extensions like custom fields becomes challenging.It is necessary to keep separate databases for the data in these additional fields as well as the meta-data characterising such adaptations. The application code must nevertheless read such meta-data in order to, for example, display and manage custom fields on the screen. Fig 10 that shows multipletenant design model utilising a single schema that providesentries exemplifies some of these problems more succinctly:



ID	Name	Address	Value	OU_ID
100				
101				
102				
...				

Entity	Custom Field	OU_ID	KeyV	alue
Customer	Hobby	503	101	Golf
Customer	Hobby	503	102	Tennis
Customer	Birthday	503	102	10/10/72
Customer	Risk	490	101	High

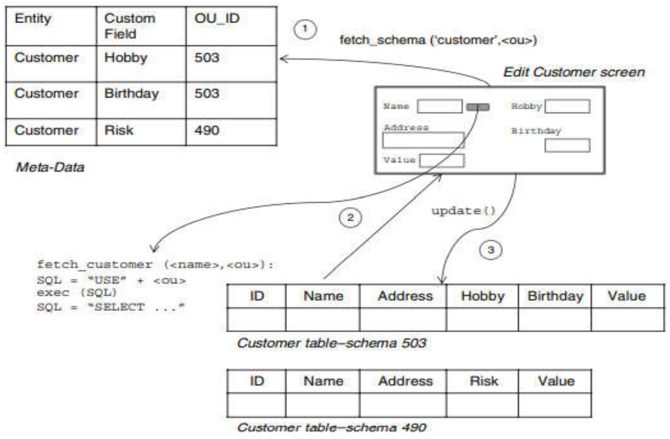
Figure 9: Multi-tenancy using single schema

A Custom Fields table houses metadata and data values for all of the application's tables. Typically, variations of this method are used to handle custom fields in a single schema design. Take a look at a screen that is employed to access and modify Customer table records. The important record is first fetched using name and appropriately selected using the OU attribute. The custom fields and values are then obtained for specific entity.For instance, there are two custom fields in OU 503 as they are shown on the screen, but only one in OU 490 and none elsewhere. Additionally, certain records might not contain values for these fields.

The aforementioned scenario is an easy one, but there are also more difficult needs that must be handled, like displaying a list of records having option to select customisedfields.

4.6.2 Multi-schema approach

It is sometimes simpler to change an existing application to use numerous schemas, as are supported by the majority of relational databases, rather than insisting on a single schema. In this paradigm, the programme determines which OU the user who is now signed in belongs to before connecting to the proper database schema. Figure 11 depicts such a structure.



Entity	Custom Field	OU_ID
Customer	Hobby	503
Customer	Birthday	503
Customer	Risk	490

```
fetch_customer (<name>,<ou>):
SQL = "USE" + <ou>
exec (SQL)
SQL = "SELECT ..."
```

ID	Name	Address	Hobby	Birthday	Value

ID	Name	Address	Risk	Value

Meta-Data
1
fetch_schema ('customer',<ou>)
Edit Customer screen
Name
Hobby
Address
Birthday
Value
2
update()
3
Customer table–schema 503
Customer table–schema 490

Figure 10: Multi-tenancy using multiple schemas

A different database schema is kept for each client in the multiple schema approach so that each schema can directly apply customer-specific customizations. The core schema's meta-data is likewise kept in a separate database, but unlike the Custom Fields table in Figure 10, this table only contains meta-data and does not contain field values for specific records. As a result, the application design is simpler, and it is likely that there will be fewer and simpler changes required to re-engineer a legacy programme for multi-tenancy.

Using a multiple schema strategy, think about implementing the Edit Customer screen that was previously discussed: The application uses data from the Meta-Data table to render the appropriate fields on the screen. Before issuing data manipulation (i.e., SQL) statements during a database query, the application sets the database schema in order to access the proper schema. Be aware that implementing components of an interpretative architecture, which is very similar to the Dev 2.0 paradigm, is necessary to enable the many schema model.

4.6.3 Multi-tenancy using cloud data stores

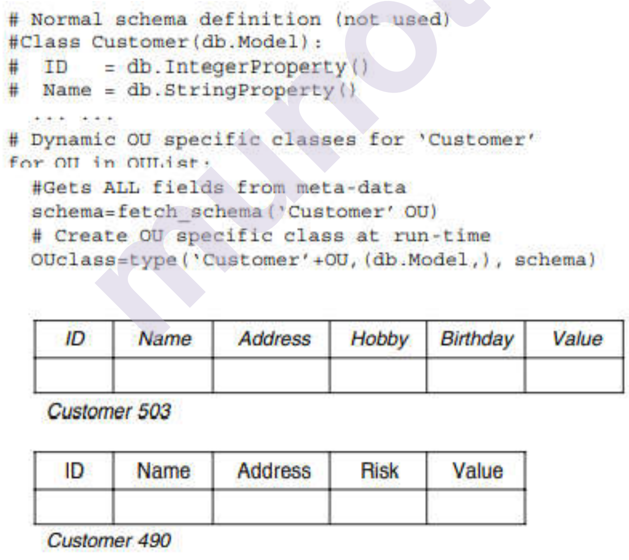
Non-relational storage models are present in cloud data repositories. Additionally, each of these data stores is multitenant from the ground up because a single instance of a large-scale distributed data store successfully supports numerous applications developed by cloud customers. While each Google App Engine user can only develop a certain number of applications, each of which looks to have its own data store, the underlying distributed infrastructure is the same for all Google App Engine users.

Here, we concentrate on a different issue: How does a user (application developer) build their own multi-tenant application on a cloud platform?The answer is simple in the case of Amazon EC2 because this infrastructure cloud provides users with direct access to (virtual) servers that can be used to replicate the multi-tenant architectures that were previously mentioned utilising common application servers and database systems.

However, utilising a PaaS platform like Google's App Engine with its Datastore, Amazon's SimpleDB, or even Azure's data services, the scenario is different. One data store name space, or schema, for each App Engine application, for instance, means that if we establish one model named "Customer," we cannot construct another model with the same name in the same application. Therefore, it initially seems as though we are forced to employ the ineffective single schema strategy.

Entities in the Google Datastore, however, are effectively schema-less, which is an intriguing feature. As a result, the language API is in charge of defining how the data storage is used. Particularly, the dynamic and object-oriented Model class in the Python API for App Engine. All entities of a "kind" acquire their properties from a class definition that derives from the Model class. Additionally, because Python is an entirely interpretative language, new classes and the accompanying data storage "kinds" can be defined at runtime.

Figure 12 illustrates a hypothetical Python implementation of multi-tenancy with Google App Engine utilising different schemas. At runtime, distinct classes are instantiated for each schema. By using table names that depend on the schema, this method is comparable to simulating several schemas in a relational database.



```
# Normal schema definition (not used)
#Class Customer(db.Model):
#  ID   = db.IntegerProperty()
#  Name = db.StringProperty()
  ... ...
# Dynamic OU specific classes for 'Customer'
for OU in OUList:
  #Gets ALL fields from meta-data
  schema=fetch_schema('Customer' OU)
  # Create OU specific class at run-time
  OUclass=type('Customer'+OU,(db.Model,), schema)
```

ID	Name	Address	Hobby	Birthday	Value

Customer 503

ID	Name	Address	Risk	Value

Customer 490

Figure 11: Multi-tenancy using google datastores

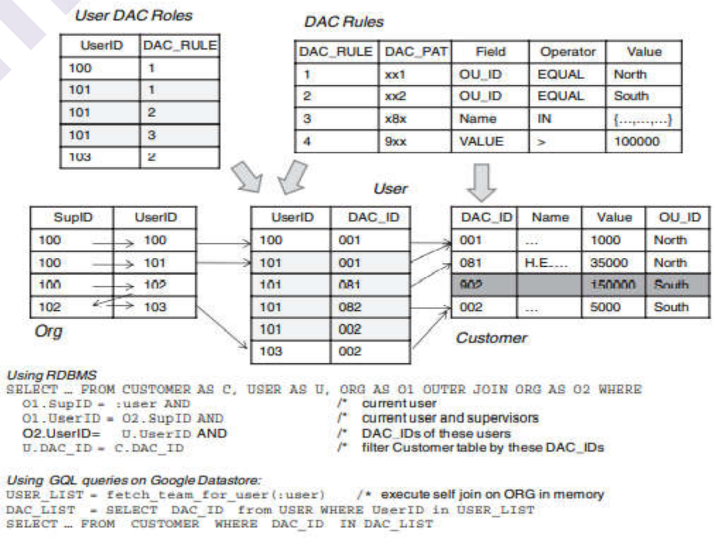
A similar approach may be utilised with Amazon's SimpleDB, where domains, which are the relational equivalent of "kind" in the Google Datastore and act as tables in relational terminology, can be generated dynamically using any of the supplied language APIs.

4.6.4 Data access control for enterprise applications

We have discussed the traditional methods for achieving multi-tenancy so far from the viewpoint of allowing a single application code base to deal with data from numerous customers while operating in a single instance, hence reducing administrative expenses across a potentially large number of customers.

Multi-tenancy as it was previously mentioned seems to be most useful in a software as a service paradigm. Additionally, there are several situations when multi-tenancy might be advantageous for the company. Supporting various entities, like bank branches, is fundamentally a multi-tenancy need, as we have already seen. If a workgroup-level application must be distributed to numerous autonomous teams, who often do not need to share data, similar needs may arise. In certain situations, modifications to the application schema may be required to handle differences in business processes.Supporting various legal entities, each of which may operate in a distinct regulatory environment, also raises similar requirements.

As we previously indicated, based on their position in the organisational unit hierarchy, a subset of users in a multi-entity scenario may need access to data from all branches or a selection of branches. In a broader sense, access to data may need to be restricted based on the values of any field in a table, such as making certain users the only ones who can see high-value transactions or making some customer names opaque unless given specific permission.These requirements, known as data access control demands, are widespread but less frequently handled in a reusable and generic way. Data access control, often known as DAC, is a generalisation of multi-tenancy because it may frequently be used to implement the latter.Figure 13 shows how data access control can be implemented generically within a single schema to allow relatively general rules for restricting access to records depending on field values.


User DAC Roles

UserID	DAC_RULE
100	1
101	1
101	2
101	3
103	2

DAC Rules

DAC_RULE	DAC_PAT	Field	Operator	Value
1	xx1	OU_ID	EQUAL	North
2	xx2	OU_ID	EQUAL	South
3	x8x	Name	IN	{...,...,...}
4	9xx	VALUE	>	100000

Org

SupID	UserID
100	100
100	101
100	102
102	103

User

UserID	DAC_ID
100	001
101	001
101	081
101	082
101	002
103	002

Customer

DAC_ID	Name	Value	OU_ID
001	...	1000	North
081	H.E....	35000	North
902		150000	South
002	...	5000	South

Using RDBMS

```
SELECT … FROM CUSTOMER AS C, USER AS U, ORG AS O1 OUTER JOIN ORG AS O2 WHERE
  O1.SupID = :user AND                     /*  current user
  O1.UserID = O2.SupID AND                 /*  current user and supervisors
  O2.UserID=   U.UserID AND                /*  DAC_IDs of these users
  U.DAC_ID = C.DAC_ID                      /*  filter Customer table by these DAC_IDs
```

Using GQL queries on Google Datastore:

```
USER_LIST = fetch_team_for_user(:user)    /* execute self join on ORG in memory
DAC_LIST  = SELECT  DAC_ID  from USER WHERE UserID in USER_LIST
SELECT … FROM  CUSTOMER  WHERE  DAC_ID  IN DAC_LIST
```

Figure 12: Data access control

Every application table, including Customer, has a new field called DAC ID added to it. The values of the DAC ID in each Customer record are populated through a batch process using patterns based on value ranges of arbitrary variables that are listed in the DAC Rules table. According to the information provided in the User DAC Roles table, users are given the authority to access records that satisfy one or more of these DAC rules. A batch method is used to expand this data to the User table, which contains records for each value of the DAC ID that a user can access.For instance, the User table contains five records when the User 101 has access to three DAC rules. This calculation determines which subset of these a specific user has access based on the User DAC Roles information after computing the entire set of mutually exclusive and unique DAC range combinations based on the DAC Rules; take note that this computation is independent of the actual DAC ID values in the Customer or other application tables.It is simple to use a join to restrict access to Customer table data to only those that a specific user is allowed, as indicated in the User table. In the example of Figure, we add a new complexity where users are also given access to all of their direct reports' DAC permissions, as shown in the Org table.

By adding a generic join and a self-join on the Org table to locate all of a user's direct reports, which is then joined to the User table and the Customer table, SQL queries on the Customer database in a typical relational database can be adjusted to allow data access control. Joins are not supported in cloud databases like Google Datastore or Amazon's SimpleDB, though. As a result, the functionality depicted in the picture must be implemented in code: The self-join on Org is performed in memory, providing a list of reportees that includes the user. This list is used as a filter to obtain the User table's list of allowable DAC IDs.

Finally, the application query on the Customer table is filtered using this list.The DAC ID must be recalculated based on the DAC Rules whenever a customer record is added or modified; this computation must also be optimised, particularly if there are many DAC Rules. Re-computation and updates to the DAC ID values in the Customer database are also necessary for the addition of new DAC Rules or the modification of existing ones.Additionally, when filling up the DAC Rules table, care must be given to guarantee that DAC ranges on the same field are never overlapping.

4.7 SUMMARY

Through their internet-connected PCs, smartphones, tablets, and wearables, users of cloud computing technologies can access storage, files, applications, and servers. Data is processed and stored by cloud computing services away from end users.Delivering computing resources through the internet, such as storage, processing power, databases, networking, analytics, artificial intelligence, and software applications, is known as cloud computing (the cloud). In this module we have discussed various technologies related to cloud computing. We have also learnt fundamentals of cloud computing platforms and infrastructures. We have seen the core concepts of web services along with the concept of mashup. Multi-tenant software concepts are also introduced.

4.8 LIST OF REFERENCES

1. Enterprise Cloud Computing Technology, Architecture, Applications, Gautam Shroff, Cambridge University Press, 2010.

2. Mastering In Cloud Computing, Rajkumar Buyya, Christian Vecchiola And ThamariSelvi S, Tata Mcgraw-Hill Education, 2013.

3. Architecting the Cloud: Design Decisions for Cloud Computing Service Models (SaaS, PaaS, and IaaS), Michael J. Kavis, Wiley CIO, 2014.

4. Cloud Computing: SaaS, PaaS, IaaS, Virtualization, Business Models, Mobile, Security and More, Kris Jamsa, Jones & Bartlett Learning, 2013.

4.9 UNIT END EXERCISES

1. Define cloud computing and discuss on cloud computing platforms.
2. What do you mean by infrastructure as a service. Explain with the help of Amazon EC2.
3. Write a note on Google Datastore.
4. Explain Amazon SimpleDB.
5. Discuss on web services.
6. Write a note on SOAP/WSDL web services.
7. Illustrate the concept of REST web services.
8. Compare SOAP Vs REST.
9. Write a note on AJAX.
10. Explain multi-tenant software.
11. Explain multi-schema approach.
12. Write a note on data access control for enterprise applications.

❖❖❖❖

5

CLOUD TECHNOLOGIES-II

Unit Structure :

5.0 OBJECTIVES

When you finish reading this chapter, you

- Recognize the idea of a thread
- Know how to create and write applications with multiple threads
- be able to use the Runnable interface and the Thread class
- comprehend the thread's lifecycle
- comprehend thread synchronization
- to understand the fundamentals of task and map-reduce programming

5.1 INTRODUCTION

Managing multiple tasks at once is the topic of this chapter. In our everyday lives, multitasking is a normal occurrence. Let's say, for illustration purposes, that you had cereal, toast, and a cup of coffee for breakfast today. Breakfast consists of three simultaneous activities: cereal, toast, and coffee.

Actually, you perform these actions sequentially rather than "at the same time": You eat some toast, then a bite of toast, and finally a drink of coffee. You continue eating toast, eating cereal, drinking coffeeis over. If there is a call when eating, there might be a situationof picking up & carry on with your meal—or at the very least, sip your coffee. By doing more "at the same time," you imply. Numerous instances of multitasking occur frequently in daily life.

Our previous computer programmes only completed singleapplication. However, at many instances when programme requires performing numerous tasks simultaneously. For instance, if you created a chat application for the internet, it would enable multiple users to participate in a discussion. The software needs simultaneously read data and disseminate them to the other group members. It would be necessary to do the reading and broadcasting responsibilities simultaneously.

5.2 CONCURRENT COMPUTING: THREAD PROGRAMMING

Threads and concurrent programming:

Threads can be thought of as methods that run "concurrently" with other methods. Typically, when writing a computer programme, we think in sequential fashion. From this angle, just one thing is in motion at once. However, many processes can really be running simultaneously while sharing the same memory thanks to modern multi-core computers.

5.2.1 What is a thread?

Executable applications are organised into single thread, often known as a thread. Imagine making list of the programs as the computer's central processing unit executes them in order to visualise a thread (CPU).

Assume splitting a programme into multiple separate threads. Every set of instructions will be unique. The applicationisperformed sequentially within a thread, as is customary. Computer can execute many threads simultaneously. Although CPU only processes single instruction, it may manage numerous threads at once by rapidly switching between them. Concurrency's key benefit is that it enables the computer to perform multiple tasks at once. The CPU might either download & perform computation, for instance.From our vantage point, it can appear like the computer is running multiple CPUs simultaneously, but it is really an illusion produced by threads that are properly scheduled.

5.2.3 Thread states and life cycle

Figure 2 and Table 1 summarise the various stages that each thread can be in during its life cycle. Labelled ovals denote the different thread states, while labelled arrows denote the transitions between the states of ready, running, and sleeping. The Java Virtual Machine and the operating system are mostly in charge. Those software can control thread transitions denoted by methodologiese.g.,start(), stop(), wait(), sleep(), and notify(). Because stopping a thread in the middle of its operation is inherently risky, JDK 1.2 deprecated the stop() method among these.The CPU scheduler is in charge of other transitions, including dispatch, I/O request, I/O done, time expired, and finished sleeping. In ready state when it is first generated. It is waiting for its turn to run on the CPU. Similar to a waiting line is a queue. The first thread in the lineis given the CPU, when the CPU becomes available. When that happens, it will be running.

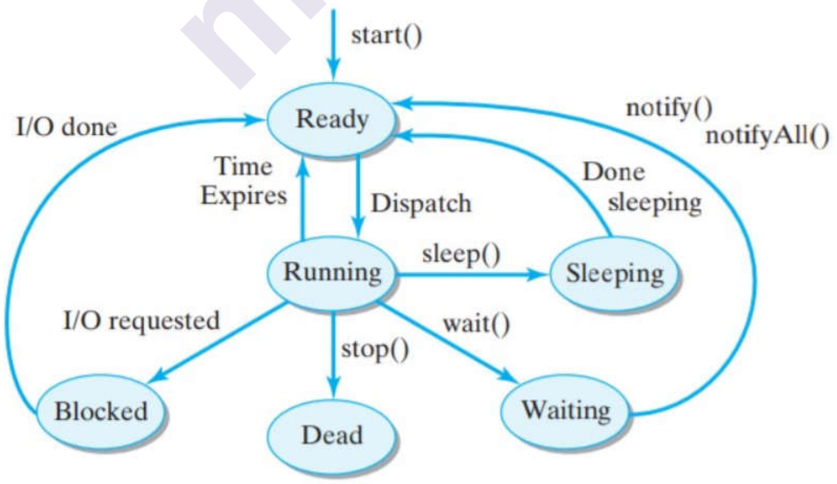


Figure 1: A demonstration of thread's life cycle

Table 1: Summary of thread's state

State	Description
Ready	The thread is ready to run and waiting for the CPU.
Running	The thread is executing on the CPU.
Waiting	The thread is waiting for some event to happen.
Sleeping	The thread has been told to sleep for a time.
Blocked	The thread is waiting for I/O to finish.
Dead	The thread is terminated.

The CPU scheduler, an integral component of the runtime system, controls the transitions between the ready and running phases. Scheduling numerous threads effectively and fairly is comparable to having several kids share a single bicycle. Ready kids wait in line for their turn to pedal the bike. The older person (the scheduler) gives the first youngster (the thread) a few minutes to ride the bike before taking it away and passing it to the next person in line. This is comparable to one of the kids choosing to take a quick break while waiting their turn. The young person would rejoin the queue after the rest was finished.When a thread uses the wait() criteria, it leaves CPU voluntarilyuntil some other thread notifies it.

The architecture controls change from blocked state to the ready state. In comparison to the CPU, I/O devices like disc drives, modems, and keyboards operate exceedingly slowly. As a result, controllers—separate processors—handle I/O operations. For instance, the system will send the disc controller instructions on where to put the data requests. The thread is halted and another thread is permitted to run because it cannot do anything until the data are read.

5.2.4 Enhancing Interface Responsiveness using Threads

A multithreaded software can be used to improve user interface responsiveness. A programme that is running statements in a lengthy (perhaps indefinite) loop in a single-threaded application does not respond to user input until the loop is ended. As a result, there will be a visible and occasionally annoying wait between when the user initiates an action and when the software really handles it.

5.2.5 Advantages of multithread design

Having changed a single-threaded software into a multi-application by adding a separate thread for Dotty. The drawing task is handled by the second thread. We ensure that thread will continue to reply to commands by requiring on sleep in every iteration. Distinction between single and multithreaded designs is seen in Figure 3. Be aware that the GUI thread runs dotty.clear and begins. Simply said, drawing thread runs its draw() procedure. All of these operations are carried out by a single thread in the single-threaded version.

Drawing N random dots will take longer with this design for a brief period after each iteration. The extra time doesn't really matter, though. The programme becomes significantly more responsive as a result of splitting into 2 independent control, 1 handling drawing work & 1 managingapplication interface.

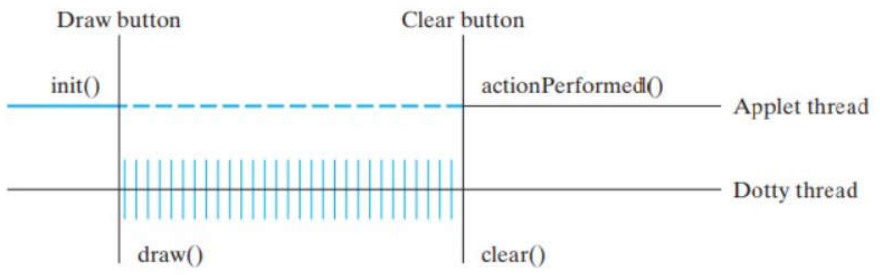


Figure 2: 2 independent threads

5.2.6 Case study: Cooperating threads

The behaviour of threads needs to be synchronised and coordinated in some applications in order for them to work together to complete a task. The producer/consumer model serves as the foundation for many cooperative applications. This approach assumes that two threads work together to produce and consume a specific information. Consumer thread reads & uses message or result that the producer thread created. The producer must be careful of not replacing anoutcomethat's not utilized, and the consumer shall await for the outcomes to get generated. The consumer model is applicable to many different kinds of coordination issues.

Controlling the visualization of information that is accesses by ourgateway / portal is one use case for this architecture. The producer thread writes information as it comes in from the Internet to a buffer. Information is read through the buffer and shown in the gatewaytab by a separate consumer thread. The two threads must, of course, be precisely synced.

5.3 HIGH-THROUGHPUT COMPUTING: TASK PROGRAMMING

The broad field of distributed system programming known as "task computing" includes numerous techniques for creating distributed applications.A task depicts a software that needs input files and generates output files as it runs. Then, applications are made up of a number of tasks. These are set into action, and when they're finished, the output data is gathered.This section describes the task abstraction and gives a quick rundown of the distributed application models that are built on it.

5.3.1 Task computing

An operation that produces a unique o/p& can be separated as auniquerationalentity is referred to as a job. In actuality, a job is expressed as a separate piece of software, or programmeoperated in a distantdomain.

The basic goal of multithreaded programming is to facilitate parallelism on a single processor. By combining the processing capability of numerous computing nodes, task computing offers distribution. As a result, this paradigm makes the existence of a distributed infrastructure obvious.Cloud computing has now become a popular way to get a lot of processing power for the use of dispersedtasks. It requires the right interface to accomplish. Figure 4 shows a typical task computing reference scenario.

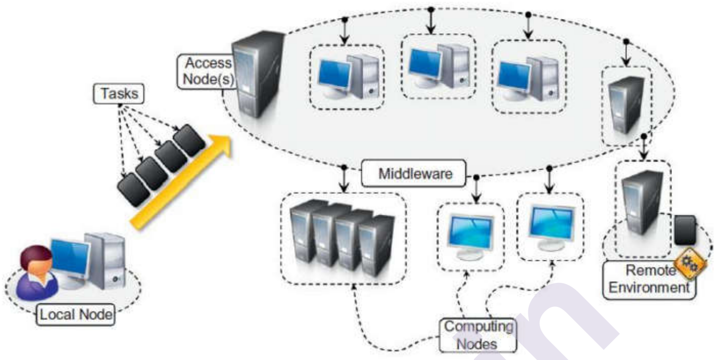


Figure 3: Task computing scenario

A software layer called middleware allows for the synchronizeapplication of numerousassets, whether they come from centralizedinformation centre or a network of PC's that are spread out geographically. The middleware's access point(s) receive a user's collection of tasks and handle task scheduling and task execution monitoring. Every computing tool offers a suitable runtime environment. The middleware's APIsare used fortasks submission.

Additionally, suitable APIs are offered for task status monitoring and result collection after task completion. A group of familiar activities required by the interface requirements to assess the development and formulation of event-based programs can be identified. The procedures include:

- Scheduling& coordinating applications for development on a number of distant junctions
- Managing dependencies and shifting programmes across distant junctions
- Establishing a setting for the remote nodes' task execution
- Tracking the progress of every application completion & updating the customer on it
- Having approach to task's o/p.

5.3.2 Characterizing a task

A task is an application component that can be conceptually isolated and run independently. Various components can be used to represent a task:

- Shell scripts that combine the execution of many apps
- Only one programme

- A piece of code that runs in the context of a particular runtime environment, such as a Java/C11/.NET class
- I/Pfolders, feasible code (programmes, etc.), and O/Papplication define a task.

Operating system or a comparable sandboxed environment serves as the tasks' runtime environment. Additionally, a task might require particular software appliances on the remote execution nodes.

5.3.3 Computing categories

These classifications offer a broad overview of the traits of the issues. They unwittingly place demands the middleware & infrastructure.

Applications are classified as:

1] High-performance computing (HPC)

HPC is the application of allocation computer resources to the solution of computationally intensive problems. A sizable number of computationally expensive activities that must be completed quickly make up the overall profile of HPC applications.FLOPS now tera-FLOPS, or even peta-FLOPS, which count the nos of FLOPS, are the metrics used to assess HPC systems.

Example: HPC applications created to address "Grand Challenge" challenges in science and engineering can be supported by supercomputers and clusters.

2] High-throughput computing

Using distributed computing resources for applications that need a lot of processing power over a lengthy period of time is known as high-throughput computing (HTC). HTC systems must be reliable and sturdy enough to last for a very long time. The typical feature of HTC apps is that they are composed of numerous activities, each of which can take a long time to complete.

Example: Statistical or scientific simulations.

Individualisticapplication that are arranged in remote expedient since they doesn't require communication are extremely prevalent. The jobs that are done each month are how HTC systems gauge their performance.

3] Many-task computing

High-performance computations known as MTC include a number of unique activities connected by bindermanipulations. MTC is diversity of applicationsconsisting of several types: Compact or big, single or multiprocessor, compute- or data-intensive, static or dynamic, single or multiple tasks are all possible. Apps that use the message-passing interface yet are loosely connected and communication-intensive fall under the category of MTC tasks.

5.3.4 Frameworks for task computing

The following are some well-known software programmes that assist framework. Each of thismodel share the overall reference architecture shown in Figure 4 in terms of architecture. A scheduling (single or multiple) & worker junctions make up their two main parts. The way that the system's parts are organised can change.

1. Condor

The most popular and durable interface for monitoring groups, inactivebureau, & a group of agglomeration is called Condor. Having the ability to control heavily occupied junctionstheysupport the capabilities of batch-queuing systems. It offers a strong task expediteorganisation that only organizetask on expedite with right continuancesurrounding. On a range of resources, Condor can manage both serial and parallel jobs. It is used to manage infrastructures by hundreds of organisations in business, government, and academia. A version of Condor called Condor-G enables interaction with grid computing resources, including those overseen by Globus.

2. Globus Toolkit

Grid computing is made possible by the technologies included in the Globus Toolkit.It offers a complete set of tools for collaborating across corporate, institutional, and geographical barriers to manage expedite&different services. In addition to security and file management, it also includes s/wfunctionalities, libraries &frameworks for expedite management& administration. The inter-operational toolkit specifies a set of interfaces and protocols that allow various parts to collaborate with one another &exploreexpediteacrossones boundaries.

3. Sun Grid Engine (SGE)

Previously known as SGE, Oracle Grid Engine is interface for managing distributed resources and workloads. SGE was first created to assist the execution of jobs on clusters, but after integrating new features, it can currently handle a variety of resources and functions as middleware for grid computing. It has extensive scheduling features like budget & collaborativemanagement, monitoringtasks that have constraint, frameworks and prior bookings. It allows execution of parallel, serial, interactive, and parametric activities.

4. BOINC

Grid and volunteer computing are supported via the Berkeley Open Infrastructure for Network Computing (BOINC) platform. It enables us to use desktop computers as clientmanipulating junctions used for executing tasks when they are dormant. Both job check pointing and duplication are supported.The BOINC server and BOINC client are the two primary parts of BOINC. The primary node for managing all of the available resources and assigning tasks is the BOINC server. The firmware element installed on computers & establishes the execution task allocation is known as the BOINC client. By building computing grids with specialised machines, these systems are quickly configured in offering muchstagnant support for task allocation.One needsto select & give their computer's CPU cycles while installing BOINC clients. The BOINC infrastructure is currently supporting a number of applications, spanning from astronomy and medicine to astronomy and cryptography.

5. Nimrod/G

Tool for parameter sweep application modelling and execution on large computational grids. For expressing parametric experiments, it offers a straightforward declarative parametric modelling language. It makes use of cutting-edge expedite monitoring and processingframeworks that are established. It enables stringent and fund limited application processing on diversifiedexpedite to reduce processing costs and achieve measurable outcomes on schedule. Over the years, it has been utilised for a very diverse range of purposes, from policy and environmental effect to quantum chemistry.

5.3.5 Task-based application models

Multiple systems are built on idea that a task serves as primary building block for creating diverse applications. The manner in which tasks are generated, their connections with one another, and the existence of dependencies or other conditions are what distinguish these models from one another. Based on the task notion, we briefly review the most prevalent models in this section.

1] Applications that are astonishingly parallel

The type of distributed applications that is the easiest to understand and consist of embarrassingly parallel programmes. The tasks do not have to communicate with one another and can be of the same type or of various types. The majority of distributed computing frameworks support this class of applications. There is a great deal of flexibility in how jobs are scheduled because they are not required to communicate. There is no requirement that tasks be completed simultaneously; they may be carried out in any order. These applications' scheduling is streamlined, and it focuses on assigning tasks to resources in the best way possible. The Globus Toolkit, BOINC, and Aneka frameworks and tools support embarrassingly parallel applications.

Rendering of images and videos, evolutionary optimization, and model forecasting are among the issues. The task of rendering a pixel or a frame, respectively, is used in both image and video rendering. When using evolutionary optimization meta heuristics, a job is found by running the algorithm just once with a specific set of parameters. The same holds true for applications of model forecasting. The majority of embarrassingly parallel applications come from scientific applications.

2] Applications for parameter sweeps

The jobs are identical in nature and the only difference between parameter sweep apps and other classes of embarrassingly parallel programmes is the execution parameters. A template job and a set of parameters are used to identify them. The operations that will be carried out on the remote node for task execution are defined by the template task. The parameter set identifies the group of variables whose assignments tailor the template task to a particular case. The execution of parameter sweep programmes can be supported by any distributed computing system that offers support for embarrassingly parallel applications.The sole distinction is that all permissible parameter combinations are iterated over in order to generate the jobs that will be executed.Aneka offers client-based tools for visualising creating a template task, setting parameters, and iterating over all possible combinations. Nimrod/G is naturally built to facilitate the execution of parameter sweep applications. This applies to a wide variety of applications. Domain of scientific computing: applications of computational fluid dynamics, evolutionary optimization strategies, and weather forecasting models

3] MPI applications

A standard for creating parallel applications that communicate by sending messages is called Message Passing Interface (MPI). As a way to bring together the many distributed shared memory and message-passing infrastructures that are available for distributed computing, MPI was first developed. Today, the de facto industry standard for creating portable and effective message-passing HPC programmes is MPI.

MPI gives programmers a collection of routines that:

- Manage the distributed environment in which MPI programmes are executed;
- Provide facilities for point-to-point and group communication;
- Support the definition of data structures and memory allocation;
- Provide basic support for synchronisation with blocking calls.

4] Workflow applications with task dependencies

Applications that use workflow are characterised by a group of tasks that show relationships between one another. These dependencies, which are typically data dependencies, determine the manner and location of scheduling for the applications.A workflow is the whole or partial automation of a business process in which tasks, information, or documents are transmitted from one resource (a human or a machine) to another for action in accordance with a predetermined set of rules. The idea of scientific workflow originated from the concept of workflow, which can be thought of as the structured execution of tasks that are interdependent. Workflow has proven to be beneficial for expressing many scientific studies. In scientific workflows, the participants are primarily computer or storage nodes, and the process is identified by an application to run. Tasks and data are the most common materials transmitted between participants.A workflow definition scheme that directs the scheduling of the application defines the collection of procedural rules. Data management, analysis, simulation, and middleware supporting the workflow's execution are typically included in a scientific workflow. A directed acyclic graph (DAG), which specifies the interdependencies among activities or procedures, is typically used to depict a scientific workflow. In a workflow application, the nodes on the DAG represent the tasks that need to be completed; the arcs linking the nodes show which tasks are dependent on one another and which data channels connect them.Data dependency, which occurs when the output files of one activity serve as the input files for another task, is the most prevalent type of dependency that may be achieved through a DAG.

5.3.7 Developing applications with the task models

Several components are required for task-based programme execution.Such apps can only be created via the following actions:

- Creating classes that use the ITask interface
- Creating an AnekaApplication instance that is appropriately configured
- Creating AnekaTask objects and wrapping ITask instances around them
- Running the programme and watching for it to finish

5.4 DATA INTENSIVE COMPUTING: MAP-REDUCE PROGRAMMING

The focus of data-intensive computing is on a group of applications that work with a lot of data. Large amounts of data are generated by a variety of application domains, from computational science to social networking, and these volumes need to be effectively stored, accessed, indexed, and evaluated. These activities become more difficult as knowledge gathers and grows at a faster rate over time. In order to overcome these issues, distributed computing offers more scalable and effective storage designs as well as improved data computation and processing capabilities.

5.4.1 What is data intensive computing?

Production, manipulation, and analysis of massive amounts of data, ranging from hundreds of megabytes (MB) to petabytes (PB) and beyond, are the focus of data-intensive computing. A collection of information pieces that are pertinent to one or more applications is frequently referred to as a dataset. Datasets are frequently kept up-to-date in repositories, which are infrastructures for storing, retrieving, and indexing massive amounts of data. Relevant data points, known as metadata, are attached to datasets to aid with classification and search.

Numerous application domains involve data-intensive processing. The most well-liked one is computational science. Many times, those who run scientific simulations and experiments are eager to generate, examine, and interpret enormous amounts of data. Every second, telescopes mapping the sky generate hundreds of gigabytes of data; over the course of a year, the collection of these photos easily exceeds a petabyte in size. Applications for bioinformatics mine databases that could ultimately hold terabytes of data. Massive amounts of data are processed by earthquake simulators as a result of the Earth's tremors being recorded all around the world.

5.4.2 Characterizing data-intensive computations

Applications that deal with large amounts of data frequently also have computationally intensive characteristics. The two top quadrants of the graph are where data-intensive computing is found, according to Figure 5.

Datasets that are many terabytes and petabytes in size are handled by data-intensive applications. Datasets are frequently dispersed across various sites and stored in a variety of formats. These applications use multi-stage analytical pipelines to process data, including transformation and fusion steps. The processing demands increase practically linearly with the amount of data, and parallel processing is simple. They also require effective data management, filtering, and fusion, as well as effective querying and delivery systems.

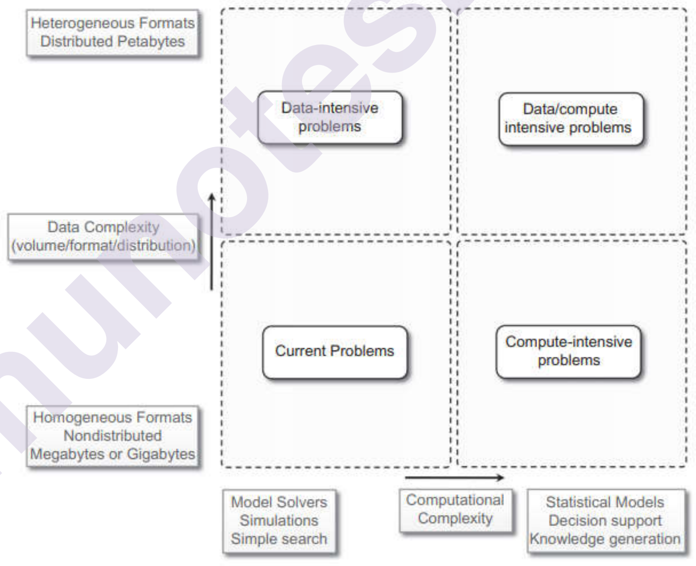


Figure 4: Data-intensive research issues

5.4.3 Challenges

The enormous volume of data generated, processed, or stored places demands on the middleware and supporting infrastructures that are rarely met by conventional distributed computing solutions. For instance, the location of data is essential because high-performance computations must avoid having to move terabytes of data. Scalable algorithms, content replication, and data partitioning all aid in enhancing the performance of data-intensive applications.

There are still open challenges in data-intensive computing:

- Massive datasets can be searched and processed using scalable algorithms
- Adaptable new methods for managing metadata that can handle complicated, heterogeneous, and remote data sources
- Platform improvements for high-performance computing that attempt to improve access to in-memory multitera-byte data structures
- Petascale, high-performance, and dependable distributed file systems
- Data signature-generation methods for quick and efficient data processing
- There are new methods for software mobility that can give algorithms that can transfer computation to the location of the data
- Specialised hybrid interconnection topologies that better enable filtering multigigabyte datastreams from fast networks and scientific equipment
- Techniques for combining software modules running on several platforms to quickly assemble analytical pipelines that are adaptable and performant

5.4.4 Technologies for data-intensive computing

The creation of applications that are primarily concerned with processing massive amounts of data is known as data-intensive computing. Thus, the technologies enabling data-intensive computing can be naturally categorised as storage systems and programming models.

1] Storage systems

Database management systems have historically served as the de facto storage support for a variety of application kinds. The relational model in its original form does not appear to be the preferable method for supporting data analytics on a wide scale due to the proliferation of unstructured data in the form of blogs, Web pages, programme logs, and sensor readings. There are fresh chances as database research and the data management sector reach a turning point. This shift is caused by a number of things, including:

- **Growing interest in big data**

A number of industries, including scientific computing, enterprise applications, media entertainment, natural language processing, and social network analysis, now routinely manage massive amounts of data. New and more effective strategies for data management are required due to the huge amount of data.

- **The increasing significance of data analytics in the supply chain**

Data management is now seen as a crucial component of business profit rather than a cost. This problem occurs in well-known social networks like Facebook, which rely on the administration of user profiles, interests, and connections between people. In order to support data analytics, the enormous volume of data that is constantly mined necessitates new technologies and approaches.

- **Presence of data in formats other than just structured**

As was already established, today's relevant information is diverse and comes in a variety of shapes and formats. The use of traditional enterprise applications and systems has resulted in the growth of structured data, but at the same time, technological advancements and the democratisation of the Internet as a platform where anyone can access information have produced a vast amount of unstructured data that does not naturally fit into the relational model.

- **Modern methods and technology in computing**

Access to a huge quantity of processing power is promised by cloud computing. As a result, software engineers can create systems that scale progressively to any level of parallelism. Building software that is dynamically deployed on hundreds or thousands of nodes, some of which may be part of the system for a few hours or days, is no longer unusual. Traditional database infrastructures are not made to sustain such an unstable environment.

These elements collectively point to the demand for innovative data management technology. This not only suggests a fresh research agenda for database technology and a more all-encompassing method of information management, but it also creates potential for rival (or complementary) models to the relational model. The main prospects for supporting data-intensive computing include, in particular, developments in distributed file systems for the administration of raw data in the form of files, distributed object stores, and the expansion of the NoSQL movement.

5.5 SUMMARY

In this chapter we have discussed about various aspects of computing such as concurrent computing, high-throughput computing and data intensive computing. Concurrency refers to the simultaneous execution of several calculations. Programming in the present day is rife with concurrency. We have also discussed about the task abstraction and gives a quick rundown of the distributed application models that are built on it.

The primary characteristics of data-intensive computing were covered in this chapter. High volumes of data are processed or produced by data-intensive applications, which can also display compute-intensive characteristics.Along with the technology, programming languages, and storage structures utilized for data-intensive computing, the data volumes that led to the definition of data-intensive computation have varied throughout time. The area of data-intensive computing first gained popularity in high-speed WAN applications.

The term "Big Data" currently refers to data dimensions that exceed the size of terabytes or even petabytes and are stored in storage clouds. This phrase refers to the enormous amount of data that is generated, analyzed, and mined by organizations that offer Internet services like search, online marketing, social media, and social networking, in addition to scientific applications.

5.6 LIST OF REFERENCES

1. Enterprise Cloud Computing Technology, Architecture, Applications, Gautam Shroff, Cambridge University Press, 2010.
2. Mastering In Cloud Computing, Rajkumar Buyya, Christian Vecchiola And ThamariSelvi S, Tata Mcgraw-Hill Education, 2013.
3. Architecting the Cloud: Design Decisions for Cloud Computing Service Models (SaaS, PaaS, and IaaS), Michael J. Kavis, Wiley CIO, 2014.
4. Cloud Computing: SaaS, PaaS, IaaS, Virtualization, Business Models, Mobile, Security and More, Kris Jamsa, Jones & Bartlett Learning, 2013.

5.7 UNIT END EXERCISES

1. What is a thread?
2. Explain the process of concurrent execution of threads.
3. Explain thread states and its lifecycle.
4. State advantages of multithread design.
5. Define and explain task computing.
6. Discuss on characterizing a task.
7. State the framework for task computing.
8. Write a note on task-based application models.
9. Explain task programming model.
10. What is data intensive computing?
11. Describe characterization of data-intensive computations.
12. State the challenges of data intensive computing.
13. Describe the technologies for data-intensive computation.

❖❖❖❖

6

SOFTWARE ARCHITECTURE-I

Unit Structure :

6.0 OBJECTIVES

- To understand the various distributed system models and evolving computing paradigms
- To gain knowledge in virtualization of computer resources
- To realize the reasons for migrating into cloud
- To introduce the various levels of services that can be achieved by a cloud
- To describe the security aspects in cloud and the services offered by a cloud

6.1 INTRODUCTION

The scale that is achievable in a massivediversifiedsituation where H/W& network problems can arise more frequently than not has been the focus of the cloud development paradigms.

This class of applications is technically referred to as "forms-based transaction-processing" but it is constrained by scalability, so we omit those that must support extremely high transaction and data volumes. There have been various attempts over the years to create common abstractions to represent such systems in order to make them more easily constructed.

6.2 DEV 2.0 PLATFORMS

Now, we focus on commercialusage, which doesn't need high amount of processing but make up a significant portion of software development efforts: Enterprise IT departments devote a lot of development and maintenance time to the multiple inter commercialframework. This class of applications often has computing needs that are orders lower than those of target critical applicationor massive visualization tasks. Due to vast number of such systems in operation, these applications represent a significant class of applications.

6.2.1 SALESFORCE.COM'S FORCE.COM Platform

It hosted multiple tenant customer relationship management (CRM) system was among first successful software as a service products. One of the factors for the hosted model's appeal was the ability for highly mobile salespeople to access CRM data online.However, the configurability of Salesforce.com's CRM system played a more significant role in its growth. The main CRM product's screens have always allowed end users to add custom fields with ease; these fields are automatically uploaded to the database, but only for the specific client who created them. Additionally, as a configuration setting, this process might be carried out while utilising the CRM software from a web browser.

here was no need for programming or a different development tool. Similar to this, straightforward frameworks can be implemented, which incorporated in tasks added to users' to-do tasksdepending on the development & update of information meeting specific criteria: For instance, a head may be alerted during a salesperson filed an opportunity with a value higher than a predetermined threshold. Through configuration, end users were able to modify CRM model to suit their requirements without any help. We illustrate this with an example below as shown in figure 1:

Consider an application for employee services that keeps track of each employee's name, contact information, &holidayremaining, or the no of unused vacation. The application enables the creation of Leave Request (LR) records. Additionally,computation must be done when submitting a new vacationapplication that rejects the request. However, if there is enough leave remaining, the request is approved and the amount of leave requested is subtracted from the leave balance.

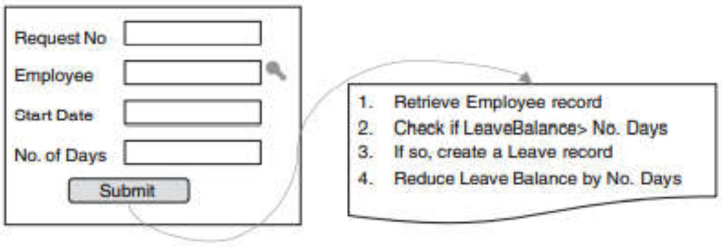

Request No
Employee
Start Date
No. of Days
Submit
1. Retrieve Employee record
2. Check if LeaveBalance> No. Days
3. If so, create a Leave record
4. Reduce Leave Balance by No. Days

Figure 1: Leave Request example

This straightforward illustration offers functions that address several requirements for fundamental forms-based office automation: forms that can link fields from one application to different one, etc. are examples of forms with fields of many sorts (including dates). Thisserves as an example of Dev 2.0, demonstrating that a somewhat complicated task may organized together instead of created using straightforward webportal rather than a programming environment. Such configuration is made possible by Dev 2.0 platforms in a user-friendly way.

Figure 2 illustrates few of actions required to create the LR application using Force.com, e.g., incorporating a certain change. This entails numerous phases during which different information needs to be incorporated and addressed&they may accomplish using the application.

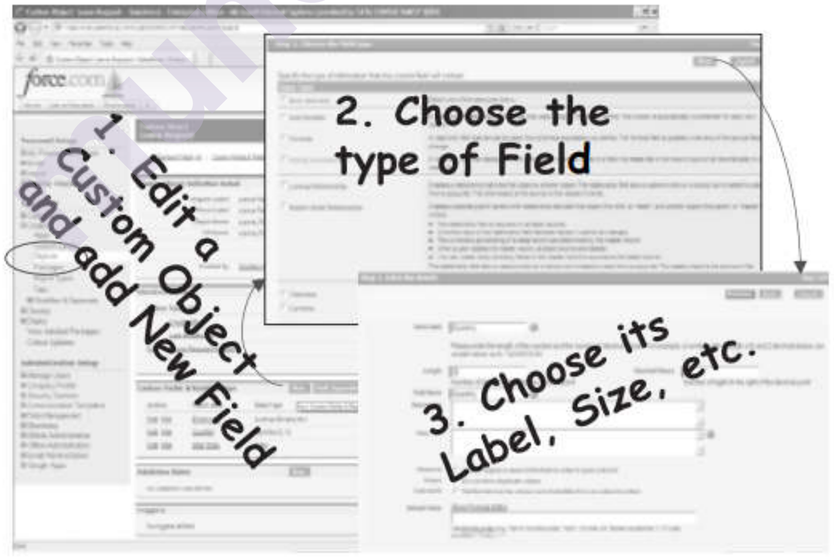
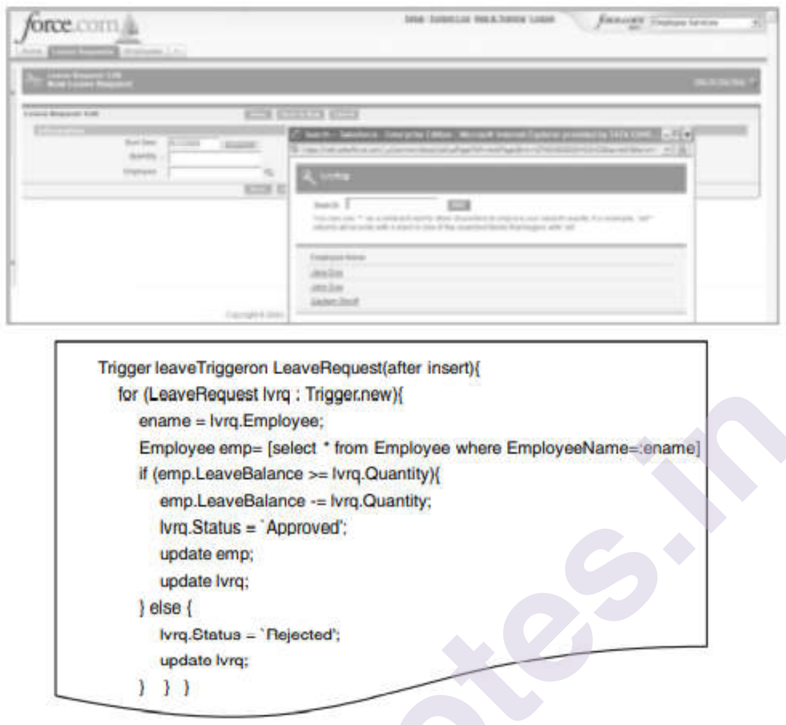

1. Edit a Custom Object and add New Field
2. Choose the type of Field
3. Choose its Label, Size, etc.

Figure 2: Adding fields in Force.com

The LRapplication generated using aforementioned procedures is shown in Figure 3, along with the APEX code needed to carry out the server-side calculations necessary when a contemporary LR is developed. This creates a "set off" on the LR that will run the moment a record of this type is entered. They use SQL to approach the directory index&improvise data in the index. You can conduct pretty general computations by uploading code that uses triggers. APEX development, on the other hand, is a task for programmers as opposed to end users, and is typically carried out in Eclipse.The platform stops being a tool that end users can customise and starts functioning as a hosted development environment for programmers.



```
Trigger leaveTriggeron LeaveRequest(after insert){
  for (LeaveRequest lvrq : Trigger.new){
    ename = lvrq.Employee;
    Employee emp= [select * from Employee where EmployeeName=:ename]
    if (emp.LeaveBalance >= lvrq.Quantity){
      emp.LeaveBalance -= lvrq.Quantity;
      lvrq.Status = `Approved';
      update emp;
      update lvrq;
    } else {
      lvrq.Status = `Rejected';
      update lvrq;
    } } }
```

Figure 3: LRwith APEX application

6.2.2 TCS instant apps on Amazon cloud

The software as a service (SaaS) paradigm is the foundation provided by Force.com. In this approach, estimation and informationincorporating on firmware&executing on servers owned & maintained by the SaaS provider. End users restrict ingress to and visibility that houses the information. Additionally, the SaaS platform provider is solely responsible for ensuring that their apps perform as demand rises: Users are unable or unwilling to manage resources allotted in advance of changes in demand.Here, we outline an alternative strategy used by the InstantApps, which was created by the research team at TCS R&D.

It is packaged like many other development tools and made accessible on EC2 infrastructure. The deployment model is similar to many aspects of a SaaS product, such as a method that allows customers to receive regular upgrades and also forces them to do so. This is in contrast to SaaS models, which do upgrades in a way that is invisible to the end user.

Figure 4 compares the deployment and use of InstantApps on Amazon framework to more conventional Dev 2.0 offerings. Here it only permits apps to run against production data; the InstantApps Designer AMI includes tools that enable application development and customization. The user retains control over the size and number of servers to boot, as well as the production data, in this way.Users can connect other programmes to their data or use SQL to access their production data.

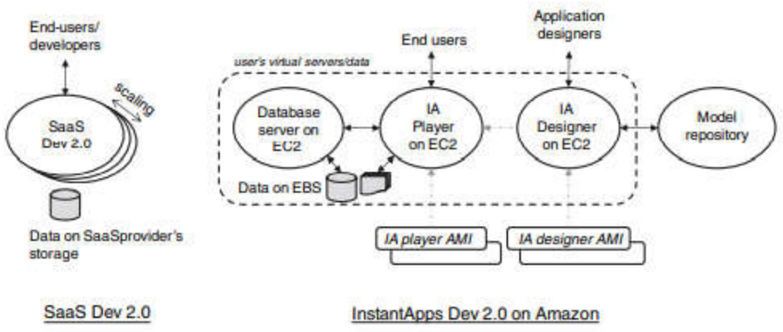


Figure 4: InstantApps on Amazon EC2

The InstantApps Designer establishes a connection with one of the Model Repository servers; as designers build their applications, this repository is filled with application meta-data. Additionally, customers are advised of newer InstantApps versions via this link, and when such upgrades occur, which can happen very frequently, exactly like in a SaaS model.

This will protect production servers from upgrades until they are genuinely needed. As a result, when it comes to upgrades, the cloud-based InstantApps deployment strategy benefits from both SaaS and conventional installed software solutions.

6.2.3 More Dev 2.0 platforms and related efforts

The FADS architecture, which was created for a VAX VMS platform but reflects insufficient exploration in the domain, is a platform that is quite similar to contemporary Dev 2.0 platforms! Later, the scholarly PICASSO framework & ABF tool also made use of a similar methodology. In contrast to the interpretative, multi-tenant designs of Dev 2.0 platforms, these are what we refer to as "first-generation" forms interpreters.

Similar to Force.com, there are now other web-hosted Dev 2.0 platforms available from new businesses. Some of these have either shut down, like Coghead, accessed by hugebusinesses, including Jotspot&Nsite..Each of these platforms supports the development of straightforward form-based workflows and is built on an interpretive, multi-tenant architecture. Similar to Force.com's APEX, many also provide the scripting of server-side business logic. Coghead offered a comparable and more platforms will undoubtedly follow suit in the future.

There have also different visualization editors and interpretative runtimes put forth in literature and on the market: WebRB that describes online workframe that work with entities and is well-suited for graphical editing and visual display. Statesoft is a for-profit tool that, like WebRB, models user interface interactions using state-charts.

6.2.4 Pros, Applications and disadvantages

Small- to medium-sized business application development could undergo a paradigm shift thanks to Dev 2.0 platforms, which could also result in considerable gains in efficiency. Their accessibility as cloud services makes it easier for end users to use them, overcoming any potential resistance from corporate IT. However, the types of apps that may be developed using the Dev 2.0 paradigm and the interpretive method are constrained.

We first look at the reasons why developing web-based applications is challenging. A distributed design that demands a multiple layerdesign format and has an impact on the software process, particularly downstream, adds to the complexity of forms-based application development (coding and test). The technology needed for each tier varies, such as writing HTML pages etc. As a result, numerous developers are typically (rather than occasionally) into creation of every entity.

The deploymentstage can be skipped if it is known that the code generators in each tier produce code that is (a) accurate and (b) correctly interacts with code produced in other layers. However, it is impossible to completely avoid the deployment, reinitializing, and restarting steps.

An alternate strategy is the interpretative Dev 2.0 architecture.Code generation does speed up development, however Dev 2.0 is more effective: Dev 2.0's "create and immediately play" methodology enables functional changes to be performed while an application is already running, allowing for immediate testing of the modification. The modifications made by other developers are immediately visible to each developer.

All usual benefits of fast prototyping platforms, such as decreased cycle times and decreased errors as a result of better requirements elicitation, are now available with WYSIWYG application development in platforms like InstantApps.

It allows important entities to be generated by client rather than coders multiplies all these advantages. Access is made possible without any infrastructure investments thanks to a cloud-based deployment architecture. Finally, the hybrid cloud deployment architecture used by InstantApps allays the frequently voiced worry about data ownership.

Despite all of this, the Dev 2.0 model has several drawbacks. First off, in our experience, there are inherent limitations to how well a method incorporates the functionality required by modest commercial tasks.

In a scripting language (which is sometimes proprietary), to execute processing logic. The functionality that can be modelled those seen in older versions is still constrained. For instance, file I/O and complex string processing cannot currently be implemented using such models. Apps also adhere to few predefined patterns. Since fundamentally altering these patterns is either difficult or necessitates scripting in JavaScript or another proprietary language, the end-user still has no influence over this level of customisation.

Finally, we haven't seen any extremely complicated applications using Dev 2.0 yet, such ones with thousands of panels and hundreds of tables.The WYSIWYG approach's seeming simplicity might actually result in increased complexity.

6.3 ENTERPRISE SOFTWARE: ERP, SCM, CRM

We considered enterprise designs primarily from a technical standpoint, charting their development burgeoning concepts.Now, we'll concentrate on business apps and the tasks they carry out within a huge organisation. We focus on the data kept in cross-enterprise programmes including enterprise resource planning (ERP), supply chain management (SCM), and customer relationship management (CRM).

6.3.1 Anatomy of a large enterprise

First task is to extract the structure of a huge firm, regardless of the industry to which it may belong. In actuality, this illustration also applies to organisations focus on corporations here for the sake of clarity. We start by thinking about the manufacturing of MRP.

Consequently, what does a manufacturing company do? It schedules what things to produce, when, and how much. The company then carries out the product process, manages production among factories, give it to clients, & offers service once they purchased the by-product. In addition to manufacturing, these fundamental activities are also prevalent in other industries.

Since they are referred to as "information" systems, corporate applications must, in the first place, maintain track of data pertaining to business activities. By creating and modifying this information, the primary business operations are then controlled. Our illustration for a company is shown in Figure 5.

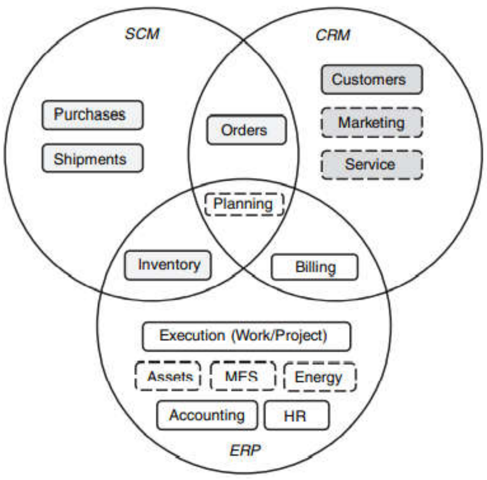


Figure 5: Core enterprise data

An organisation has customers, and it uses marketing and sales activities to connect with them. These clients put orders for merchandise. The company then ships products to clients after delivering them from an inventory. Keep in mind that if an organization's business model is to "create for stock," then products are stocked as inventory.

Executing a procedure—or, more generally, putting out some effort—is the method by which a product is manufactured. Together with other necessary material inputs, the cost of such activity needs to be monitored. Additionally, certain material inputs must be purchased. Finally, bills are sent to clients and post-sale services are performed. To determine the corporation's profit, both incoming revenue and incurred costs must be taken into account. Along with managing the material assets possessed and the energy used, the organisation also needs to manage the people (human resources) within it.

All of the aforementioned information requirements are equally relevant to manufacturing and other industries, but with appropriately modified semantics. Manufacturing execution systems, or MES, or detailed information tracking, analysis, and optimization, have recently focused on processes in context of manufacturing industries. Last but not least, the company adds strategic value by organising and controlling all action planswith the help of budgets.

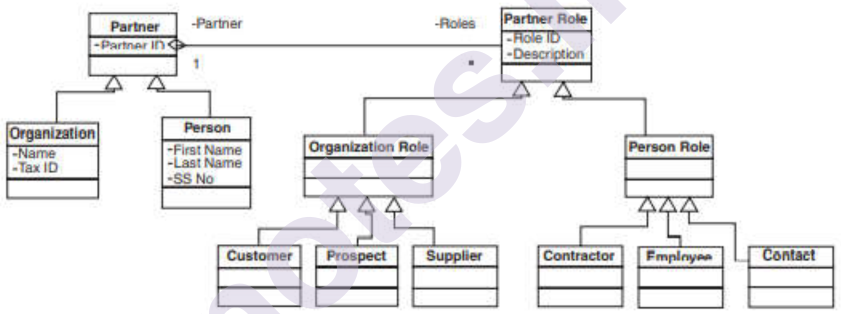
The first widespread cross-industry applications were manufacturing MRP systems. Later, these systems were known as ERP, or "enterprise resource planning" systems, because they extended beyond just manufacturing to include accounting, sales, and suppliers. Eventually, the roles of the customer and supplier grew more intricate and started to assume distinct identities.

As shown in Figure, there are imbrications in categorization. For example, order administration&payment, which have significant involvement but are typically categorised under SCM and ERP, respectively, share many similarities and interact with one another. Additionally, there are other situations in which packaged solutions might not be appropriate, as well as particularly unique processes that require software that has been specifically designed for them.

6.3.2 Partners: People and Organizations

Corporations must interact with other businesses, such as their suppliers or consumers. Both the employees of these firms and the people who buy their products must be dealt with. Early information systems frequently used different applications to keep track of customers, suppliers, and personnel individually. Important details could thus frequently be missed during routine operations; for instance, a customer complaining who also happened to be a senior executive of a significant corporate client would not receive special treatment from a call centre staff. Additionally, there were missed chances to upsell and cross-sell to current clients.

Nowadays, it's standard practise to combine customer, supplier, and employee model whenever possible. This entails portraying individuals and organisations as "partners," each of whom may play a variety of roles. A straightforward partner model is shown as a UML class diagram in Figure 6. Since either a person or an organisation might be a partner, this relationship is generalised in the UML by using class inheritance. In this relationship, partners may take on one or more partner responsibilities, and it is portrayed as an aggregation.Through a role hierarchy, the partner role is also subclassified into other real roles, like that of a client, supplier, or employee.The PARTNER table and PARTNER-ROLE table, with partner ID serving as a foreign key to indicate a one-to-many link between these tables, are a partial representation of this class model in terms of relational tables, as illustrated in Figure. Notably, we have not demonstrated how the role hierarchy could be implemented relationally and how that would affect the PARTNER-ROLE table for the sake of simplicity. The process of converting a logical class model into a relational representation involves a number of steps, including several methods for converting class inheritance to relational tables.



PARTNER TABLE

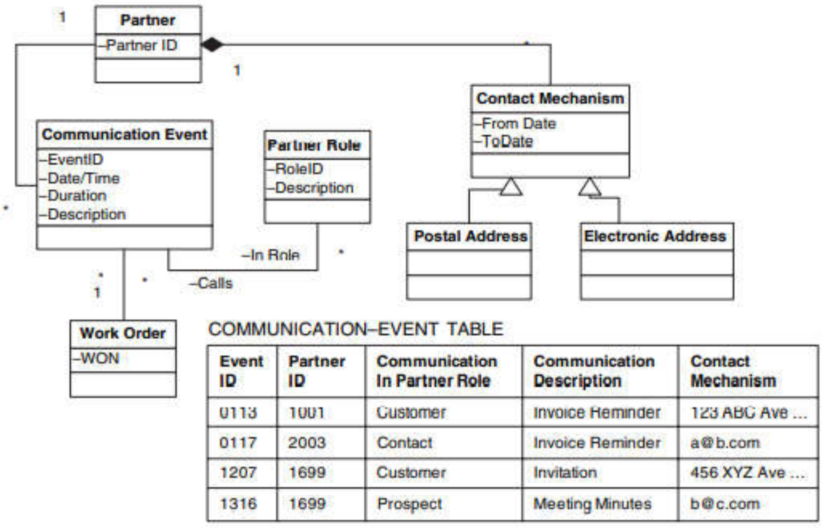
Partner ID	First Name	Last Name
1001		ABC Inc
2003	John	Doe
1040		XYZ Supplies
1699		EZ Software

PARTNER–ROLE TABLE

Partner ID	Role ID	Description
1001	1	Customer
2003	5	Employee
2003	6	Contact
1699	1	Customer
1699	2	Prospect

Figure 6: Partner model

As shown in Figure 7, communication with partners occurs through established contact methods, which might be postal or electronic. As before, a partial relational representation of this model is shown, in which the Partner-ID foreign key in the COMMUNICATION-EVENT table is used to express the one-to-many link between the partner and communication event classes. Additionally, this table's Communication-In-Partner-Role property denormalizes and captures the many-to-many relationship between communication event and partner role.Usually, communications also have additional aspects of the organisation model attached to them, such as a "work order" for billing any charges incurred in order to carry out the communication.



COMMUNICATION–EVENT TABLE

Event ID	Partner ID	Communication In Partner Role	Communication Description	Contact Mechanism
0113	1001	Customer	Invoice Reminder	123 ABC Ave ...
0117	2003	Contact	Invoice Reminder	a@b.com
1207	1699	Customer	Invitation	456 XYZ Ave ...
1316	1699	Prospect	Meeting Minutes	b@c.com

Figure 7: Partner communication model

We have greatly streamlined the partner model. The internal organisational structure of organisations, their staff hierarchies, the history of communications preserved as threads, and many more notions are commonly included in a real enterprise's highly complicated model.

6.3.3 Products

As seen in the model in Figure 8, an organisation both creates and consumes products, which might be either goods or services. A business frequently produces or consumes a large number of items, making it crucial to categorise them in a variety of ways, such as by model, grade, segment, or other product category. Product categories may contain or be contained within other categories; for instance, the categories "paper" and "office supplies" both include pencils and paper. Take note of how a UML association class is used to depict a many-to-many relationship like this. This would be equivalent to a "link table" in a normalised relational data model with the product-ID and category-ID as foreign keys pointing to the PRODUCT and PRODUCT-CATEGORY tables.

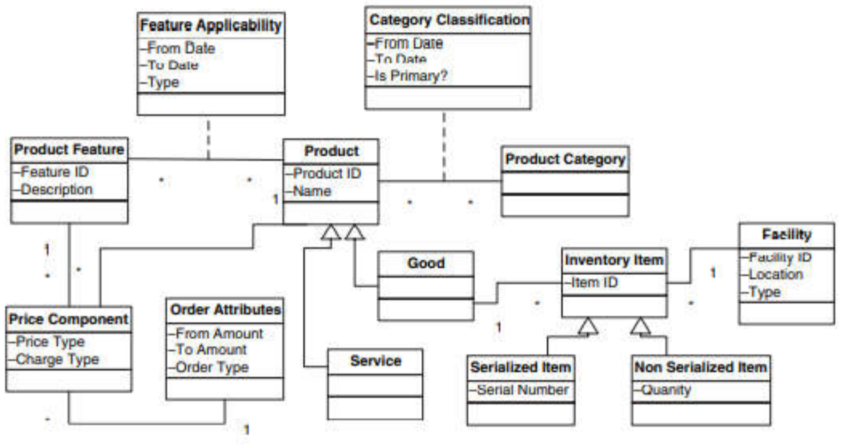


Figure 8: Product model

One or more features may be present in a given product. The price of a product can vary depending on these aspects, the product itself, or other order parameters like the value or type (for example, retail vs. bulk) of an order.

Models of genuine products and their supply are shown in Figure 8. Physical commodities are preserved as inventory items, which are kept in storage facilities like warehouses or manufacturing plants, as opposed to services. Inventory items that aren't made by the company itself must be bought from some supplier businesses. Suppliers of these products are tracked, along with their ratings, preferences, and contract prices. Additionally, details on the products that a specific provider is capable of creating are also recorded. Finally, reorder criteria for each product set policies on how much inventory is to be retained.

6.3.4 Orders: Sales and Purchases

In order to conduct business, a firm must first take sales orders before shipping products or rendering services in response to them. Additionally, it makes procurements, or orders for goods that it could need. Both sorts of orders require tracking of many of the same factors, including who placed the order, where and when it is to be delivered, at what price and in comparison, to what quotation, how the order was placed, and who gave their approval.

A combined model for sales and purchase orders is shown in Figure 9. A number of order elements make up each order. A product or a product characteristic may be the subject of each order item. Each order involves multiple parties performing various roles, including who placed the order, who will be billed, and who will receive each order item. Additionally, orders are placed using a contact method provided by the relevant partner, and each order item includes the contact method for shipping requirements.Several shipment elements are included in shipments that are made to fulfil orders. It is crucial to remember that a shipment item may end up fulfilling several order items in perhaps distinct orders, or vice versa, a shipment item may satisfy numerous order items in one order. As a result, a class association order shipment that tracks the quantity in a specific shipment item against a certain order item connects order items and shipment items.

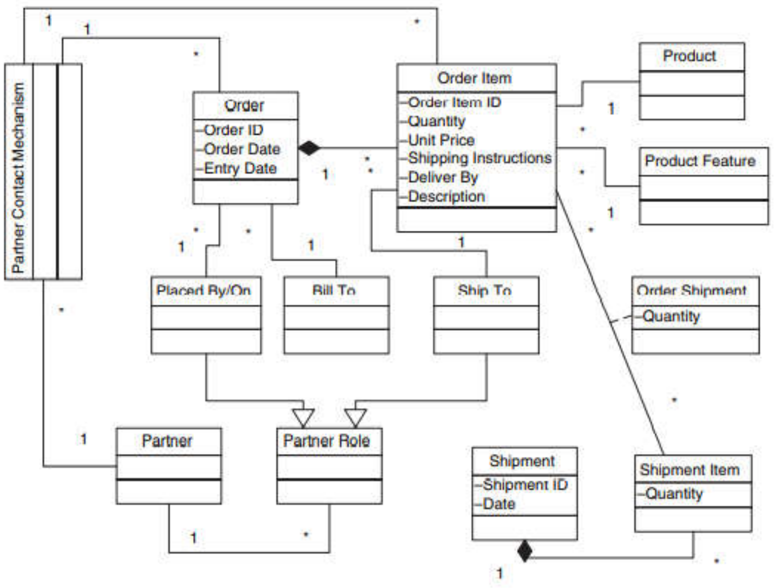


Figure 9: Order model

A supply chain application typically handles the order model shown in Figure 9. However, the sales process, which is often managed by a CRM application, is responsible for moving a client request from a request to a sales order. The procurement element of SCM typically includes the corresponding process for purchasing. In either scenario, a request is often made to one or more suppliers or received from the client prior to placing an order.Figure 10 shows a quote model that uses quotations and agreements to follow inquiries to orders: Many request items may be included in a request. The responding company provides a quote that includes quote items, each of which is for some products or services that the company can offer for a given price. Following the exchange of quotes, negotiations take place, and a contract (or agreement) is eventually drafted between the two businesses. The agreement specifies the pricing structure, which consists of several agreement items that correspond to price components for the requested products, including any reductions and terms that have been agreed upon. The identical pricing components found in the agreement items are then used to base orders against the quote.

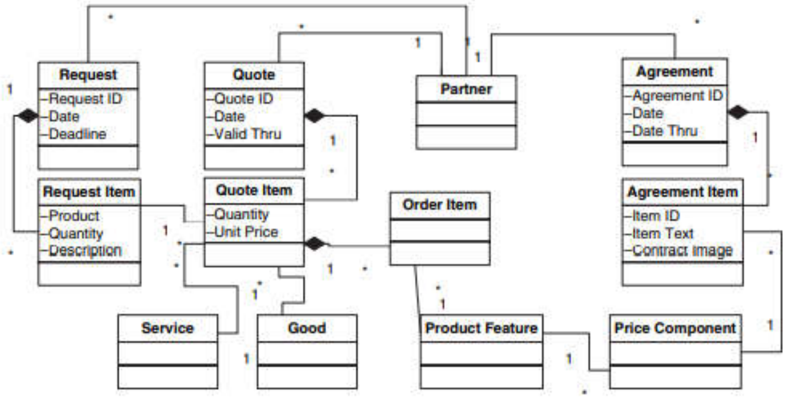


Figure 10: Quote model

6.3.5 Execution: Tracking work

In addition to providing commodities, businesses also offer services that necessitate the use of their staff's labour and may be billed either directly or indirectly. Additionally, costs linked with product design and manufacturing, R&D, and other such internal operations must be documented. Figure 11's architecture keeps track of work orders, which might be generated in reaction to incoming sales orders or internal needs.Note that we have used multiple inheritance to illustrate the reality that work orders may occur in two separate ways. Additionally, this implies that our model does not distinguish between a work order and a sales order or an internal requirement. Work instructions frequently go by the moniker "projects" and have specific names. Thus, our methodology guarantees that a project to assure an order's completion is likewise started as soon as an order is received (or an internal requirement is accepted).

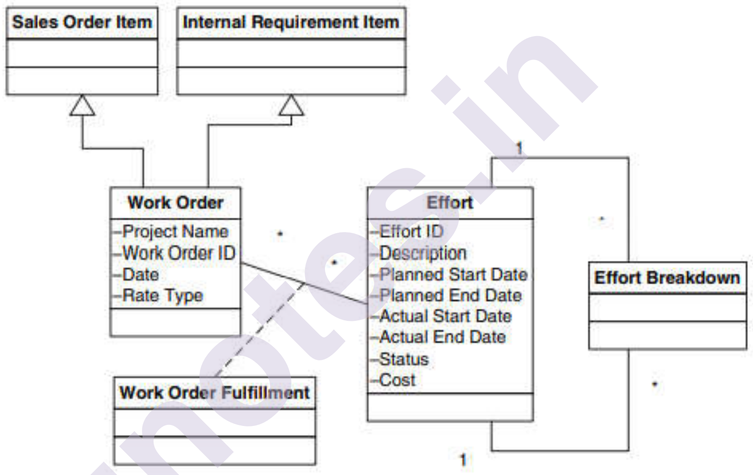


Figure 11: Work model

By putting forth effort to complete work orders, the specifics of the intended and executed efforts are recorded. As illustrated by the effort breakdown class, efforts can be divided into smaller efforts. It should be noted that a work order can be completed by numerous efforts, and the model permits an effort to contribute to multiple work orders. For simplicity's sake, a work order is typically connected with a single effort in practise.

Our work model accurately depicts the key elements of effort tracking, such as how a need leads to work orders that may involve a complicated series of planned and completed efforts. It does not, however, include additional costs related to completing a work order, such as the usage of tangible assets, consumables, and travel costs. Furthermore, internal requirements that result from planning can be fairly complex because they might arise for a variety of reasons, including projected sales, scheduled maintenance, sales and marketing initiatives, or research and development, to which linkages may need to be maintained.

People who are working on their given jobs expend effort. The party assigned to a specific effort is depicted in the model in Figure 12, where parties might be either organisation personnel or outside contractors. An associated rate may be assigned to an effort when it is assigned to a party. Alternately, the rate (cost or fee) might be determined by the pay scale of a particular employee or contractor. The rate class includes the actual cost-rate or billed-rate. Each rate may have a corresponding rate-type, often known as a rate "schedule."The rate schedule to use for a specific work order is also recorded in the work order class, as was previously demonstrated in Figure 11, allowing for the use of the proper charging or costing rates while completing the work order.

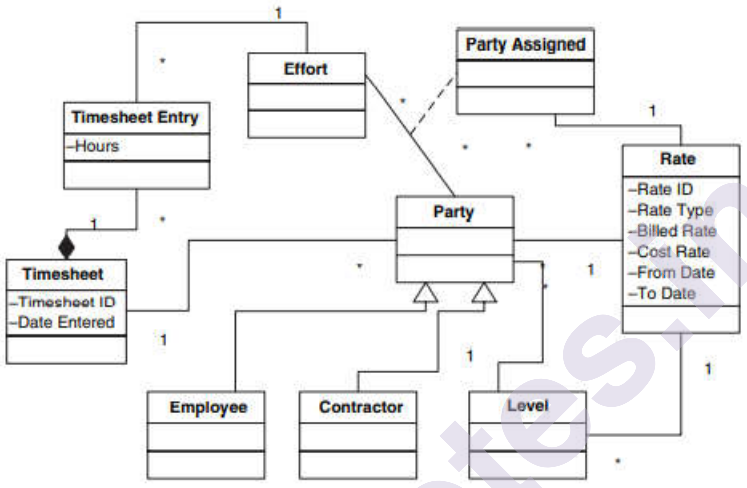


Figure 12: Rates and timesheets

Once parties have finished their work, they record it in a timesheet that contains timesheet elements. Each timesheet item represents the amount of time devoted to a specific work task. Timesheets are essential for invoicing, costing, and accounting in many areas of the service sector as well as shop floor activities in industrial facilities.

6.3.6 Billing

Customers must be billed and payments must be made after things have been delivered. Similar to this, payments must be made for supplier purchases. In either scenario, payments must be linked back to the invoices that billed for the billable goods or services for which they were issued. A fundamental billing structure is shown in Figure 13. Invoices can be sales invoices or purchase invoices because billing entails sending an invoice to a partner (or receiving one from a supplier). We incorporate all of them via a class association invoice role between invoice and partner since there may be more partners linked with an invoice, such as the "ship to" or "ordered by" partner, which may differ from the "bill to" partner.

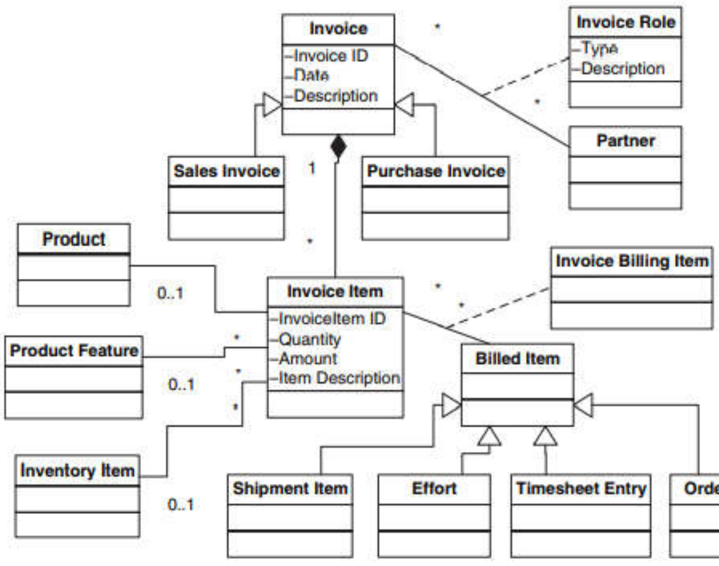


Figure 13: Billing model

An invoice may have a number of invoice items, each of which includes the quantity and description of the products or services being billed as well as the total price due. Each invoice item may relate to a particular inventory item, a product feature, or both; these are consequently displayed as optional linkages to the respective classes. In addition, each line item on an invoice is associated with the actual delivery of some products or services, i.e., one or more billed items. A billed item may be a shipment item, an effort, or a timesheet entry. Invoices can also be invoiced on order; in which case a billed item might be an order item.

Payments are credited to an invoice once it's been sent. (For purchases, this is the other way around; payments are made against invoices received.) A payment model is displayed in Figure 14. Payments are made by a partner and might take the form of receipts or disbursements. In a payment, the sum being paid, a reference number, and the day the payment becomes effective are all recorded (as opposed to when it is physically received). One or more invoice items may be used as the recipient of a payment, and several payments may be made to one invoice item. This many-to-many link is modelled as a class association for payment invoices.Last but not least, every payment triggers a financial transaction in a bank account; revenues trigger deposits, and payments made result in withdrawals. Keep in mind that financial transactions between legal entities occur in the actual world through a third-party financial institution (such as a bank).

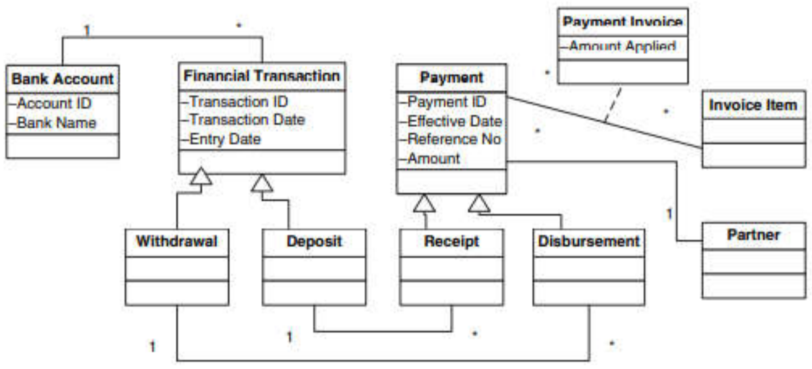


Figure 14: Payment model

6.3.7 Accounting

The financial situation of an organisation is undoubtedly impacted by financial transactions like payments. But so are unpaid invoices that have been raised, as well as inventory accumulation that could result in a loss if it cannot be sold.The accounting model of the business captures the financial position of the company, and accounting transactions are made to reflect the effects of each business activity (such as actual payments or simple invoices) on the company's financial position.

The chart of accounts, which consists of a collection of "general ledger" (GL) accounts, serves as a visual representation of an organization's financial structure. Each GL account serves as a container for posting accounting transactions.Figure 15 shows a straightforward accounting approach. Each account in the general ledger has a GL-ID, a name, and a description. Examples include "travel expenses," "accounts receivable," "cash" accounts, and "payments owed."A GL account also has an account-type that describes where it belongs on the balance sheet of the company, indicating whether it monitors assets, liabilities, revenues, or expenses.

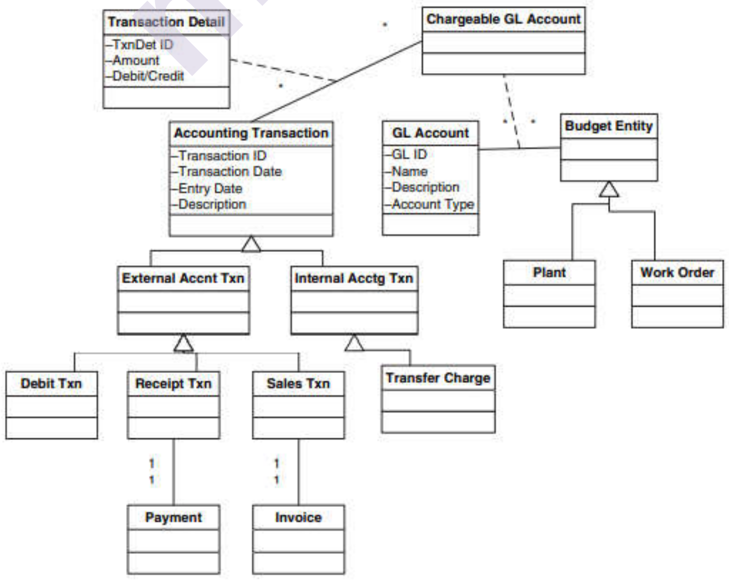


Figure 15: Accounting model

The enterprise's method for capturing financial data is determined by the chart of accounts, which consists of the set of GL accounts. A large organisation, however, requires financial reporting at the level of each budget entity, or an entity with the capacity to spend or receive money, such as a manufacturing facility or project. Chargeable accounts, against which actual accounting transactions are posted, connect GL accounts to such budget entities.A profit and loss statement or comprehensive balance sheet can be created for any chargeable account or collection thereof, including the organisation as a whole, using chargeable accounts and the relationships among them (not shown in the illustration).

As was already indicated, certain business transactions are recorded as accounting transactions. An accounting transaction can be both internal and external, such as a transfer charge between two divisions of the company. Such external transactions may involve actual money exchanges, such payments, and are thus recorded in a GL account designated as "cash." Alternatively, these could be invoices that are sent to an "asset" GL account because they represent a future obligation by another party to pay. When an invoice is unpaid, it turns into a bad debt and must be moved to another GL account designated as a "liability" by matching credit and debit accounting transactions.

Even from the incredibly simplified model above, it is clear that accounting is a complicated function. It is also a fundamental role of any enterprise's IT systems. Since generic accounting systems are provided via packaged software, accounting functions across businesses are quite similar. As a way to automate the capturing of business transactions and tie them to accounting transactions in a flexible way so that one system could meet the demands of a company, the first ERP systems emerged from such accounting packages. This is both a benefit and a drawback of integrated ERP programmes.

6.3.8 Enterprise processes, build Vs buy and SAAS

The accounting function was the focus of the initial ERP systems, which aimed to broaden it to include various company operations. Users would have access to various business transaction kinds depending on their positions. As a result, salespeople would have access to the production of orders and the preparation of invoices but not to the tracking of efforts or shipments. In such a case, it was difficult to follow an order from its start until payment was collected. For a company to operate effectively, it is crucial to track such an "order to cash" process. Managing inventory or procurements involves similar end-to-end procedures. Each of these "horizontal" processes typically spans many areas of the company data model.

An organisation should ideally implement end-to-end operations on a uniform underlying data model if it were to construct its systems from the ground up.However, using packaged software makes the problem more challenging. Consider the scenario where a core ERP package, for instance, does not track shipments and specialised software is created to do so. We have already seen the necessity to correlate shipments to order items. The order items must match those kept in the primary ERP system. Using the different APIs that a core ERP system typically provides; all of this is still possible. Now imagine that an SCM and a core ERP system are the two key software applications being used.Additionally, order items and customer shipping information will be maintained by the SCM and synchronised with the primary ERP system. Customer information is now being kept in three systems that require constant synchronisation after the addition of a CRM package.

Domain-specific functionality causes additional issues, suchcompared to the cross-industry roles we have discussed thus far, for example: Organizations providing financial services could have to deal with trading and securities and transactions. These demand a sophisticated partner model with brokers, clients, stock exchanges, and banks among others. A typical CRM package would includeunlikely to include these components. At the same time, it would be costly to redevelop the essential CRM functionality included in a CRM package. Thus, decisions regarding "build vs. buy" must take integration issues into accountbetween software programmes, old computer systems, and unique applicationscan be contrasted with the substantial expenses associated with creating and sustaining a singlea combined application using a single data model.

When choosing which enterprise applications are suitable for software as a service, additional factors come into play (SaaS). In this case, in addition to data security concerns (or worries), it is crucial to take into account both the volume of data that would need to be routinely transmitted from the SaaS system to applications implemented on-premises as well as the degree of data replication. CRM systems are advantageous for SaaS from this perspective because the only point of replication is certain client information. As new customers are acquired and customer information is updated more gradually than during business transactions, there is less of a burden on the SaaS product to synchronise client data with on-premise applications.

Another obvious SaaS contender is order management, particularly when web-based orders are involved. Another instance where SaaS is a viable choice is in human resources: A SaaS HR system may provide a monthly payroll and even facilitate direct debits and credits from the organization's and employees' bank accounts. However, if SCM or other fundamental ERP components are taken into account, the scenario changes because a larger amount of data may be transferred. Furthermore, these systems lose their appeal as candidates for SaaS, or cloud-based implementation, the more closely they are connected to tangible assets like manufacturing facilities and warehouse inventory.

6.4 SUMMARY

In this module we have discussed about the various software architecture associated with cloud computing technologies. We have learnt Dev 2.0 platform fundamentals such as TCS instant apps on Amazon cloud, along with its advantages, disadvantages and its applicability. Followed by enterprise software: ERP, SCM, CRM. Here we have discussed about the anatomy of a large enterprise, the products and orders, its execution, billing and accounting process.

6.5 LIST OF REFERENCES

1. Enterprise Cloud Computing Technology, Architecture, Applications, Gautam Shroff, Cambridge University Press, 2010.
2. Mastering In Cloud Computing, Rajkumar Buyya, Christian Vecchiola And ThamariSelvi S, Tata Mcgraw-Hill Education, 2013.
3. Architecting the Cloud: Design Decisions for Cloud Computing Service Models (SaaS, PaaS, and IaaS), Michael J. Kavis, Wiley CIO, 2014.
4. Cloud Computing: SaaS, PaaS, IaaS, Virtualization, Business Models, Mobile, Security and More, Kris Jamsa, Jones & Bartlett Learning, 2013.

6.6 UNIT END EXERCISES

1. Discuss on Dev 2.0 platforms.
2. Write a note on SALESFORCE.COM's FORCE.COM platform.
3. Explain TCS instant apps on Amazon cloud.
4. State the advantages and disadvantages of DEV 2.0.
5. Illustrate the applicability of DEV 2.0.
6. Write a note on enterprise software.
7. What do you mean by anatomy of a large enterprise?]Explain the products, orders and execution process of an enterprise software.
8. Write a note on billing and accounting process.
9. Explain the concept of enterprise process, build Vs buy and SAAS.

❖❖❖❖

7

SOFTWARE ARCHITECTURE-II

Unit Structure :

7.0 OBJECTIVES

The objective of the section is:

- To get acquaint with the fundamentals of software architecture
- To get familiar with the operating of custom enterprise applications and Dev 2.0
- To understand the concepts and terms associated with cloud applications

7.1 INTRODUCTION

We looked at the various information requirements of a typical business. We observed that these requirements can, be put into place employing goods or SaaS services. While custom application development may be necessary for some commercialframework or realm distinct capabilities.

7.2 CUSTOM ENTERPRISE APPLICATIONS AND DEV 2.0

Here, we will examine the construction of such enterprise systems, whether they are created as part of a bundled good or a specially designed application. We will look at ideas needed to create contemporary apps during this process. We have discussed the developing Dev 2.0 paradigm, which makes it possible to create applications largely without the use of bespoke code. Understanding the internal architecture of Dev 2.0 platforms will also be made possible by our examination practises.

7.2.1 Software architecture for enterprise component

It is possible to think of enterprise applications as being made up of tasksentities, in charge of group of business activities that have approach to a common features of information objects.

Technically, each component of an application can be thought of as either a standalone application or as a subsetof a big corporate task, such as CRM. The following tasks are carried out by the application (component), which is crucial for the discussion that follows: It keeps certain data, facilitates clientreciprocity with the data using display level, & does some business logic calculations while doing so.This type of layered construction is seen in Figure 1. These architectural layers match the conventional "model-view-controller" (or MVC) architecture, in which a control layer divides the user interface from the information model.

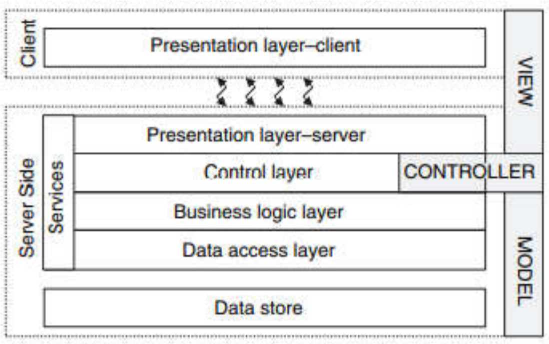


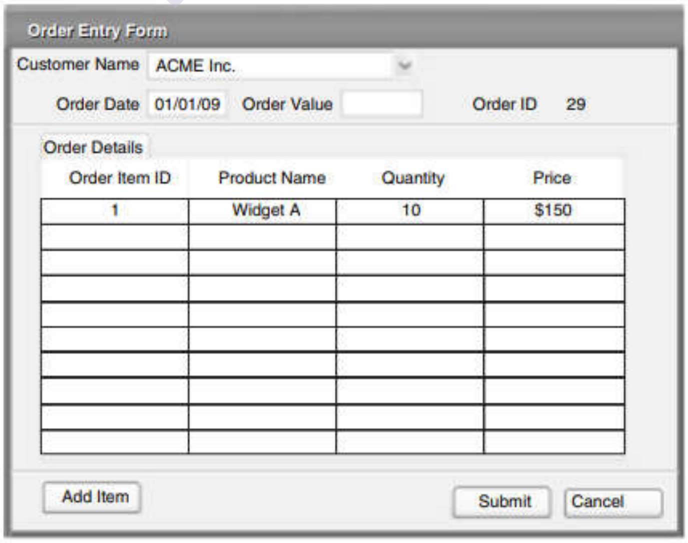
Figure 1: Architecture layers

Consider the following architecture as that of Apache Tomcat as an illustration of how the levelled framework of Fig 1 is deployed in practise: A portalaccessed to server through HTTP is referred to as the "client." The application installed on a framework like EC2connected to the browser of a private network within an organisation. JSPs that gets executed both make up the presentation layer.Information access layer provides rational level with access to information storage. The controller layer is in charge of implementing access control, also known as application-level security, controlling. The complete programme is also accessible to varied frameworks like web or mashups, in addition to these levels.

7.2.2 User interface patterns and basic transactions

7.2.2.1 Layered MVC and the AJAX Paradigm

Informationaccessed and computational screensare used by users to view, enter, and modify information stored in the data store in order to perform tasks. Let's think about the actions that are performed duringaddition of a new entity. We assure a multi-leveldesign, such that shown in Fig 1built on browsers: The client enters information after accessing a URL that produces an HTML page. This information must be verified, for instance to make sure that numerical data is entered into numerical fields.


Order Entry Form

Customer Name: ACME Inc.

Order Date: 01/01/09 Order Value: Order ID: 29

Order Details

Order Item ID	Product Name	Quantity	Price
1	Widget A	10	$150

Figure 2: An order entry form

Typically, these validations are carried out using JavaScript functions. The information is then transmitted to the presentation layer server. The user level, or the informationlaunched into or changed is managed by server-side presentation layer before delivered to the other levels. It's effective to keep part of input information than repeatedly exchanging because the HTTP protocol is fundamentally memoryless and asynchronous. We point out, however, that in a standard client-server architecture, state might be preserved within the "fat-client" programme itself, negating the requirement for such server-side functionality. As a result, the server-side presentation layer is once more rendered partly unnecessary.

Let's now examine the controller layer's motive: Think about ainformation capturing operation once more that requires numerous "pages." How does the user move around the pages? Such are developed by the user application in a client-server design and the AJAX paradigm. Due to this architectural limitation, complex controller frameworks like "Struts" and "Spring" were created.

Let's look at the layered web design. Many of tasks required by MVC are carried out by the controller layer, which is described above. Layered web architecture falls short of a key requirement of classic MVC, which is that the controller must immediately reflect any changes to model information in the "view" or presentation layer whenever the model's value changes: When a client approachentity, its most recent record is obviously displayed. Theyfrequently make connections in the AJAX paradigm, making a genuine MVC architecture much simpler to implement.

Let's now compare the controller layer to the conventional layered to see how it differs: It is frequently possible to navigate from one page to another using AJAX without requesting a new HTML page while still in the same logical database transaction. As a result, similar to the classic client-server design, part of the controller's navigational functions is moved to the client side. The server side continues to be used for the controller's remaining features, such as interacting with other layers. Figure 3 depicts the updated layered architecture in the AJAX paradigm.

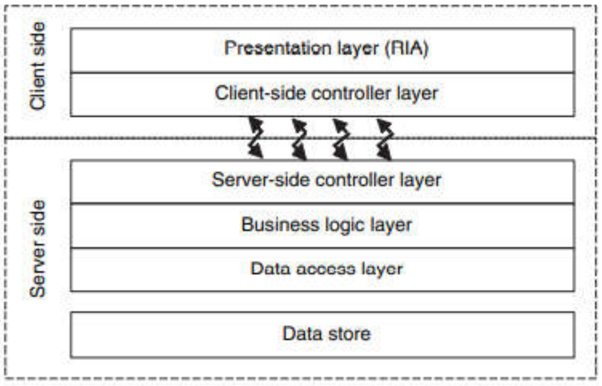


Figure 3: MVC layering with AJAX

7.2.2.2 Common UI patterns

We have seen how the user interface of a browser interacts with a server. Concentrating on the features that such an interface should have is equally crucial. There are numerous data model patterns that businesses use frequently. These can serve as a starting point for outlining the data that a business application keeps track of. One of these classes is "invoice-generating" tasks, in which customerapproach, alter, and look for information while carrying out a business process like "order management." In contrast, "decision-support" systems use sophisticated data visualisations based on many types of analytics.

Recollect the order-entry process from Fig 2. When an upcomingentityrequires to be written, it must be made sure that latestentity and any inclined transaction are produced. It is necessary to search for, select, and link to the order object being produced information as well as the proper billing and delivery addresses.The complexity of the product price and shipping costs, as well as other factors, should make it evident that a straightforwardoperation may contain possibly varied formulations. As a result, even a straightforward order entry transaction may have a complex user interface that required bespoke creation. The creation of these user interfaces accounts for a sizable portion of application programming work.

7.2.2.3 Formal models and frameworks

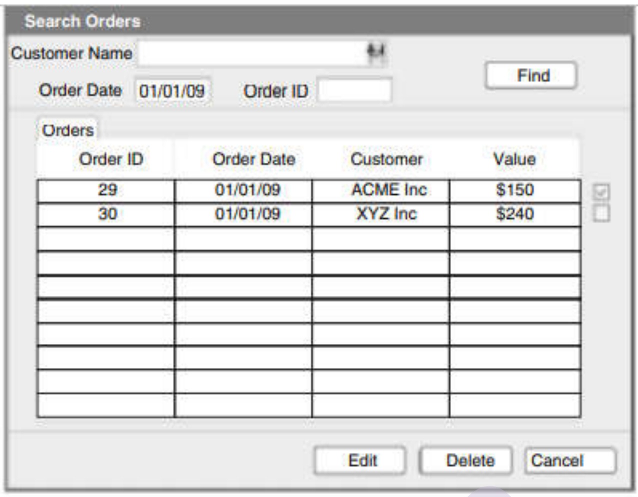
Let's try to identify some abstractions that could help us more formally model the behaviour of the order entry transaction. Let's discuss form technically as collection of fields that map columns in the application entities. The "order" form, for instance, consists of the columns customer name, order date, and order id, each of which corresponds to a column of the ORDER database.Customer name & Address fields, which correspond to columns may also be found on another "customer" form. The "order item" form has fields like order id, quantity, etc. For each form F, let's declare the basic page

Pages of a form: (1)

1. Searching page returns R aa a collection of entries that match any values put into the fields of F.
2. A result page allows users to choose any of shown entity for manipulating or removing while also displaying the outcome of R.
3. A user may alter any F fields for a specific record on an Edit page and submit their changes to the server.
4. A Create page enables the creation of a new record

Fig 4 shows an illustration of the Search & Result "pages", which is a "search orders" screen: Here, despite having the same form, or "order," two "pages," namely Search and Result, are combined onto one screen.

Generally speaking, a "page" of a form may contain other "pages".Keep in mind that the common field order id connects the two pages from different forms ensuring it match the specific input.



Customer Name			
Order Date	01/01/09	Order ID	

Order ID	Order Date	Customer	Value
29	01/01/09	ACME Inc	$150
30	01/01/09	XYZ Inc	$240

Figure 4: Search orders form

We now demonstrate how the formal definition of the straightforward order entry screens shown in Figures 2 and 4 may be achieved.

$$
\begin{aligned}
order &\xrightarrow{has\ fields} order\_date,\ customer\_name,\ order\_id\\
Create(order) &\xrightarrow{invokes} Create(order\_item)\\
Edit(order) &\xrightarrow{invokes} Create(order\_item)\\
Create(order) &\xrightarrow{includes} Result(order\_item)\\
Edit(order) &\xrightarrow{includes} Result(order\_item)\\
customer\_id &\xrightarrow{type} LOV(customer,\ customer\_id)\\
menu &\xrightarrow{has\ items} New\ Order, Search\ Orders\\
New\ Order &\xrightarrow{invokes} Create(order)\\
Search\ Orders &\xrightarrow{invokes} Search(order)
\end{aligned}
$$

Here, each line is produced using an abstraction-based formal grammar's production rule. The type of each token is implicitly defined. With the help of such a formalism, it is possible to specify the kinds of the fields that the order form contains as well as the forms that are either linked to or contained in it. For instance, the order_item form's Create and Edit pages both have aoutcomelayout that lists specific transaction that is generated & updated.Additionally, both include aninterface that may be used to launch the Create page, allowing users to add new order items while generating or amending an order. The order_id column, which is shared by the order and order_item forms, as previously indicated, guarantees that the OrderItem record being generated or changed in either case relates to the right Order record.

Such technical architectural frameworks are frequently created and maintained in the sector. It is alluring to believe that there might be a general framework that firms could apply more frequently.Unfortunately, any such structure would quickly become outmoded due to the development of technology.

7.2.3 Business logic and rule-based computing

7.2.3.1 What does business logic do?

In MVC terminology, the'model' layer includes business logic. Aspects purely connected to presentation are categorically excluded from this layer. In a tiered MVC design, the controller layer is in charge of transferring once purified of components. As an illustration, when an application is sent via a POST, the controller layer parses it, extracts the field values from the REQUEST object that is received, and converts them & are then accessed and forwarded to the proper functions.In contemporary web architectures utilising the AJAX paradigm, the server is required to execute requests in addition to sending up new pages and processing page submissions.

Even within the same application, the business logic layer's functionality handling can differ greatly. However, similar to user interfaces, there are some widespread primitive abstractions that we may also spot in this situation.

We can list the duties that a business logic method must perform in brief as follows:

1. Verifications
2. Calculations
3. Agreement monitoring, which involves completing each & every necessary task within itscontext.
4. Informationmanoeuvres and its correspondence mapping
5. Invoking different business logic techniques, such as publishing an accounting transaction.
6. Complex Search (such as getting several orders and the products associated with them

7.2.3.2 Rule-based computing

The abstraction is dependent on fixedrational. In addition to many other computations, such as assessing risk or rating insurance policies, rules are particularly helpful for modelling validation rules. When assessed against facts, a regulationmodel comprises rational claimsthat might or might not turn out to be true.

For instance, the following guidelines could be used to define a "valid" order:

$$\begin{aligned} R1 : \forall x \forall y \forall z\ & Order(x) \wedge OrderItem(y) \wedge OrderOf(y,x) \\ & \wedge ValidOrderItem(y) \wedge CustomerOf(x,z) \wedge Customer(z) \\ & \Rightarrow ValidOrder(x) \end{aligned}$$

$$\begin{aligned} R2 : \forall x \forall y \forall z \forall p\ & Order(x) \wedge OrderItem(y) \wedge \neg OrderOf(y,x) \\ & \wedge CustomerOf(x,z) \wedge Customer(z) \\ & \wedge ProductOf(x,p) \wedge Product(p) \\ & \Rightarrow ValidOrder(x) \end{aligned}$$

Predicates like Order(x) and Customer(z) in the example above evaluate to T if x, y& z, respectively, are Order, OrderItem, and Customer objects. Numerous codes exists for assessing given input as reflected from the example above.

Rule systems are assessed using rule engines. Two different types of rule engines exist: Backward-chaining engines analyse all potential rules that are necessary to validate if a specific predicate, such as ValidOrder(x), is true or false. The whole set of available facts are determined by forward-chaining engines, on the other hand. Applications have typically employed backward-chaining rule engines.

7.2.3.3 Modelling business logic using MapReduce

We have seen that the business functions of data access and transaction management may be handled utilising architectural frameworks that incorporate desired patterns to consistently solve these issues. Similar to what we saw in the previous section, rule-based computing can be used to abstract validation rules. Computations, data manipulation, calling other business methods, and complex search are what are left. Some Dev 2.0 platforms also make an effort to abstract these business logic functions. For instance, TCS InstantApps leverages the MapReduce cloud programming paradigm and the Map-reduce-merge model, which extends MapReduce for relational processing, to create a visual formulation called Logic Map.

A logic map is a graph that contains nodes for create, search, and update that each represent "pages" of an application form. Without a user interface, these nodes manipulate relational records in the same way as the corresponding pages of their connected forms. These nodes can also be thought of as map nodes in the MapReduce context; they read and manipulate database tables while producing new pairs of key/value pairs (relational records) as input. Additionally, there are "compute nodes" called merge and reduction that execute calculations on data received after merging (for merge) or aggregating (for reduce) records flowing along the edges of a logic map.

The logic map in Figure 5, for instance, executes a billing operation for one or more customer entries. As instances of a customer form C with the fields cid,...> (cid serving as the primary key of the customer database), customer records flow into the logic map. This collection of "form instances" is directed toward a search node connected to the order form O. The search node gets all orders (from the order table) that were placed by any of the customers flowing into the node because the attribute cid is shared by the order and customer forms. Keep in mind that a set of instances of the order form are now the output of the Search Orders node.A second search node on a product form P is used in parallel to retrieve all of the items' current prices. Then, using a temporary intermediate "valued order" form VO, these parallel record sets are linked together using a merge node. The product prices and quantities ordered are utilised to calculate the value of each order during this merging operation, and records from the two incoming sets are linked together using the common attribute pid. Instances of the VO form finally flow into a reduce node on the customer form C, where valued orders are summed to determine the total billing for each client.

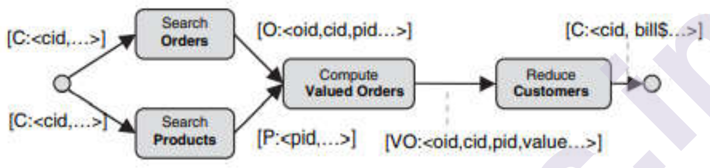


Figure 5: Logic map

Decision nodes comparable to compute nodes are also permitted by logic maps, however they filter records in the flow rather than manipulating them. Loops can be created using decision nodes, allowing for the modelling of a reasonably broad class of business logic functions. By utilising the MapReduce paradigm, it is possible to efficiently use cloud data stores even though they do not explicitly enable joins because the join capability is included into the merge abstraction. This allows for the parallel execution of logic maps in cloud environments.

7.2.4 Security, Error handling, Transactions and Workflow

Technical features of an enterprise application that are not immediately related to functional components also require architectural support. The elements of the technical architecture shown in Figure 6 carry out the following tasks:

1. Application security must be used to restrict access to programme functionality so that only authenticated users can access the application and use the functions for which they have been granted permission.

2. For properly handling various classes of problems, controlling error messages, and informing users about issues, uniform error handling is crucial.

3. In addition to preserving data integrity via underlying database transactions, transaction management is required to guarantee that each logical user interaction is atomic in terms of concurrency control.

4. Workflow management is required to handle the work that moves from user to user to complete a business process.

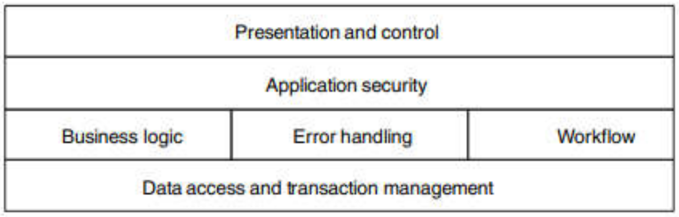


Figure 6: Technical architecture layers

7.2.4.1 Application Security

Whether a client-server architecture or a web-based design, secure user authentication in a distributed context is a complex topic. The Kerberos protocol was created to provide authentication across an unsafe network while still protecting users against replay attacks and eavesdropping.A user's password is never sent over the network using this protocol, not even when it is encrypted or hashed. Instead, a trusted server creates short-lived tickets and session keys that are used by clients and servers to connect and to authenticate one another.Currently, Kerberos variants are implemented as a fundamental component of distributed security at the operating system level, in both Windows and Linux, and in Windows Active Directory and LDAP-based identity management servers.

Each HTTP request sent once a user has successfully logged in comprises a "user context" that contains the security tokens or tickets needed by the selected authentication method. Every application service must validate the user context on the server before processing it; typically, the controller layer ensures this as it processes each request made to the server.

The web application server's underlying features or the application architecture can be used to provide function access control. Access control lists (ACLs), which are frequently dependent on the URL or server-side code packages (such as a Java.jar file) being requested, are typically used by web application servers to control access to application resources. However, the fine-grained level of access control required by the majority of enterprise applications is frequently insufficiently supported by this. Furthermore, these functionalities cannot be easily used by rich internet interfaces (RIA), which heavily rely on JavaScript. As a result, function access control is typically implemented at the application level along with data access control.

A straightforward data model for function access control is shown in Figure 7. Through the User Role class, Users have Roles, which are connected to numerous other Users. Through the Role Function class, roles have access to functions. Each function is either a "page" of a specific "form" (such as Create, Edit, Search, etc.) or a menu item. During login, the application security layer consults the function access control model and adds the user's list of available functions to the user context. In order to restrict user access to only those forms and menus allowed by the function access control model, security must then be checked in both the presentation layer and controller layer.Due to the fact that it is implemented across many architecture layers, application security is an example of a "cross-cutting" concern.

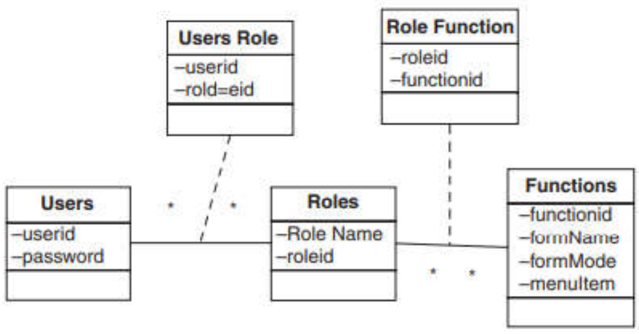


Figure 7: Function access control

7.2.4.2 Error handling

Users will encounter a wide variety of faults while conducting commercial transactions, and they must be made aware of these. These include business failures as well as technical errors that may result from programme defects, such as when validation tests fail or a notification that the data being searched for is not available. Additionally, signals like those that confirm a transaction's success must be communicated together with pertinent data like the order-id that the system will automatically produce. An error handling strategy is a set of mechanisms that allow an application to notify these errors and provide the user the opportunity to react; an error management framework uniformly applies this strategy to all user interactions in an application.

An error-handling framework's implementation specifics are directly related to the technological elements and application architectural design principles. Error handling is thus also a cross-cutting issue, even more so than application security: All architecture levels between the point where the issue occurs and the presentation layer are involved when an error or combination of errors needs to be communicated to the user.

7.2.4.3 Transaction management

From the standpoint of concurrency management, every user interaction in a transaction-processing application that involves adding, deleting, or altering data must be atomic. It's crucial to understand the distinction between an atomic database transaction implemented by the controller layer and an atomic user interaction. The controller compiles all data that a user might have sent over the course of potentially several HTTP requests. In order to ensure database integrity, it then sends each of these to the business logic layer to be carried out as a single database transaction. However, in a scenario with multiple users, this might not be sufficient to guarantee concurrency control:

Consider a scenario where two users use an Edit form to access a database record at the same time to reserve a certain resource (like a conference room).The independent user sessions each perform their own updates before sending the changes to the server via HTTP POST requests. To prevent lengthy transactions, timeouts, and deadlocks, reading a database record is purposefully kept outside the scope of a database transaction in the layered MVC design. So far as sending a distinct database transaction to the server is concerned, both of these user activities seem to be successful.

Web-based architectures most frequently employ 'optimistic' concurrency control using version numbers. Every table has an extra field called ver that gets bigger with every update. The version number of a record is also read when it is read. The structure of the update statement as it is carried out by the data access layer is as follows:

```
UPDATE <table> SET <col1>=:1, <col2>=:2, ver=ver+1
WHERE <key>=:key AND ver=:ver
```

In such circumstances, the version number mismatch will result in no data being discovered (an error), which can be communicated to the user as a "transaction failed" notice. This happens when the version number has changed between the time a record is read and the attempt to publish it. As a result, only one user—the one who acts first—will be successful in making a reservation for the conference room.

Since each architecture tier needs to keep track of the version number, which is also conveyed across HTTP requests, even though the actual check is only at the data access layer, optimistic concurrency control, often known as "soft locking," is a cross-cutting issue.

7.3 CLOUD APPLICATIONS

An application that performs the same functions as a native application but runs in the cloud and is accessed by web browsers and APIs is referred to as a cloud application. Typically, when we refer to the cloud, we are referring to public cloud Infrastructure-as-a-Service platforms; however, IaaS entered the cloud market much later than Software-as-a-Service, with cloud applications coming first.

7.3.1 What are cloud based applications?

Service-based software Since the late 1990s, cloud apps have existed, evolving from more basic web programmes that made use of Flash and Java to offer rudimentary "desktop-like" capabilities available through a web browser.

"Cloud application" is a vague term. Providing functionality through a network where computation and storage take place on servers in data centers is the subject of a definition that everyone can agree on. The cloud, in the broadest definition, is anything that takes place online as opposed to locally. However, a cloud computing application is typically used in a more limited sense: it involves infrastructure and applications that are used and maintained through the internet, has a web-based user interface, and frequently — but not always — involves virtualization. This definition covers both cloud apps that operate in a remote data center and cloud infrastructure platforms that offer virtual servers, networks, and other infrastructure.

7.3.2 Examples of cloud applications

A prime example of a cloud application is Google Docs or Office 365. All you need for Google Docs or Office 365 is a computer that can run a web browser and an internet connection. Remote servers provide the user interface and all of the functionality, including data storage. Numerous distinct cloud apps can be hosted by your company using cloud application servers.

Cloud applications Vs Native applications

Additionally, Google Docs offers a helpful point of contrast between contemporary cloud applications and the more antiquated native application paradigm. Bandwidth was limited in the early days of the internet. Delivering feature-rich applications via the internet while maintaining a positive user experience was unattainable.

On local computers, programmes like Microsoft's Office were downloaded once or purchased on DVDs. The local machine served as the sole processing and storage facility.

The local application model has some advantages, but in an era with plentiful bandwidth and a web platform with significantly more functionality than ever before, many developers opt to design for the cloud first. Even businesses like Microsoft that gained fame for their desktop applications are moving toward the cloud for the distribution of applications.

More and more companies are utilizing the cloud and cloud software as access to cloud services and data center IT infrastructure increases for service providers.

Cloud applications Vs Web applications

Nearly as long as the web, web applications have existed. Early web browsers incorporated JavaScript so that programmers could add capabilities that went beyond static pages. You'll remember Java-applet and Flash games and applications if you've been using the internet as long as I have.

What distinguishes cloud applications from web applications then? First, and rather counter-intuitively, the majority of contemporary cloud applications leverage web-native APIs and technologies. You are not need to download a browser plugin in order to access a cloud app because it makes use of browser functionality.

Second, while cloud applications compete with native programmes despite frequently being simpler and providing a more intuitive user experience, online apps frequently have less capabilities than desktop applications.

7.4 SUMMARY

All of the architectural concerns covered in this section are "cross-cutting concerns," as we have stressed. Such cross-cutting issues can be introduced and managed in complicated systems using aspect-oriented programming (AOP), which makes it easier to change the implementation techniques for such concerns even after an application architecture has been implemented. AOP, for instance, makes it possible to effectively change cross-cutting functionality for the platform as a whole in Dev 2.0 designs.

7.5 LIST OF REFERENCES

1. Enterprise Cloud Computing Technology, Architecture, Applications, Gautam Shroff, Cambridge University Press, 2010.
2. Mastering In Cloud Computing, Rajkumar Buyya, Christian Vecchiola And ThamariSelvi S, Tata Mcgraw-Hill Education, 2013.
3. Architecting the Cloud: Design Decisions for Cloud Computing Service Models (SaaS, PaaS, and IaaS), Michael J. Kavis, Wiley CIO, 2014.
4. Cloud Computing: SaaS, PaaS, IaaS, Virtualization, Business Models, Mobile, Security and More, Kris Jamsa, Jones & Bartlett Learning, 2013.

7.6 UNIT END EXERCISES

1. Explain the software architecture for enterprise component.
2. Write a note on user interface patterns and basic transactions.
3. Discuss on layered MVC and the AJAX paradigm.
4. Explain common UI patterns.
5. Explain formal models and frameworks.
6. What do you mean by business logic and rule-based computing?
7. What does business logic do?
8. Write a note on rule-based computing.
9. Explain the concept of modelling business logic using MapReduce.
10. Elaborate the following concepts: Security, error handling, transactions and workflows

11. Explain the concept of application security.
12. What do you mean by error handling?
13. Write a note on transaction management.
14. What are cloud-based applications?
15. State the examples and benefits of cloud apps.
16. Explain the types of cloud servers.

❖❖❖❖