CLOUD COMPUTING

UNIT I:

Introduction: Network centric computing, Network centric content, peer-to –peer systems, cloud computing delivery models and services, Ethical issues, Vulnerabilities, Major challenges for cloud computing. Parallel and Distributed Systems: introduction, architecture, distributed systems, communication protocols, logical clocks, message delivery rules, concurrency, and model concurrency with Petri Nets.

Network-centric computing and network-centric content

The concepts and technologies for network-centric computing and content evolved through the years
and led to several large-scale distributed system developments:
• The Web and the semantic Web are expected to support composition of services (not necessarily
computational services) available on the Web.1
• The Grid, initiated in the early 1990s by National Laboratories and Universities, is used primarily
for applications in the area of science and engineering.
• Computer clouds, promoted since 2005 as a form of service-oriented computing by large IT companies,
are used for enterprise computing, high-performance computing, Web hosting, and storage for network-centric content.

Processing can be done more efficiently on large farms of computing and storage systems accessible via the Internet.

Grid computing – initiated by the National Labs in the early 1990s; targeted primarily at scientific computing

Utility computing – initiated in 2005-2006 by IT companies and targeted at enterprise computing.

The focus of utility computing is on the business model for providing computing services; it often requires a cloud-like infrastructure.

cloud computing is a path to utility computing embraced by major IT companies including: Amazon, HP, IBM, Microsoft, Oracle, and others. 3 Cloud Computing: Theory and Practice. Chapter 1Dan C. Marinescu

- Any type or volume of media, be it static or dynamic, monolithic or modular, live or stored, produced by aggregation, or mixed.
- The "Future Internet" will be content-centric. The creation and consumption of audio and visual content is likely to transform the Internet to support increased quality in terms of resolution, frame rate, color depth, stereoscopic information.

Data-intensive: large scale simulations in science and engineering require large volumes of data. Multimedia streaming transfers large volume of data.

Network-intensive: transferring large volumes of data requires high bandwidth networks.

Low-latency networks for data streaming, parallel computing, computation steering.

The systems are accessed using thin clients running on systems with limited resources, e.g., wireless devices such as smart phones and tablets.

The infrastructure should support some form of workflow management.

Evolution of Concepts and Technologies

The concepts and technologies for network-centric computing and content evolved along the years.

The web and the semantic web - expected to support composition of services. The web is dominated by unstructured or semi-structured data, while the semantic web advocates inclusion of sematic content in web pages.

The Grid - initiated in the early 1990s by National Laboratories and Universities; used primarily for applications in the area of science and engineering

PEER-TO-PEER SYSTEMS

The demand for services in the Internet can be expected to grow to a scale that is limited only by the size of the world's population. The goal of peer-to-peer systems is to enable

the sharing of data and resources on a very large scale by eliminating any requirement for separately managed servers and their associated infrastructure.

The scope for expanding popular services by adding to the number of the computers hosting them is limited when all the hosts must be owned and managed by the service provider

Peer-to-peer systems aim to support useful distributed services and applications using data and computing resources available in the personal computers and workstations that are present in the Internet and other networks in ever-increasing numbers.

This is increasingly attractive as the performance difference between desktop and server machines narrows and broadband network connections proliferate But there is another, broader aim: one author [Shirky 2000] has defined peer-to peer applications as 'applications that exploit resources available at the edges of the Internet – storage, cycles, content, human presence'.

Each type of resource sharing mentioned in that definition is already represented by distributed applications available for most types of personal computer. The purpose of this chapter is to describe some general techniques that simplify the construction of peer-to-peer applications and enhance their scalability, reliability and security.

Traditional client-server systems manage and provide access to resources such as files, web pages or other information objects located on a single server computer or a small cluster of tightly coupled servers. With such centralized designs, few decisions are required about the placement of the resources or the management of server hardware resources, but the scale of the service is limited by the server hardware capacity and network connectivity.

Peer-to-peer systems provide access to information resources located on computers throughout a network (whether it be the Internet or a corporate network). Algorithms for the placement and subsequent retrieval of information objects are a key aspect of the system design.

The aim is to deliver a service that is fully decentralized and self-organizing, dynamically balancing the storage and processing loads between all the participating computers as computers join and leave the service.

Their design ensures that each user contributes resources to the system.

• Although they may differ in the resources that they contribute, all the nodes in a peer-to-peer system have the same functional capabilities and responsibilities.
• Their correct operation does not depend on the existence of any centrally administered systems.
• They can be designed to offer a limited degree of anonymity to the providers and users of resources.
• A key issue for their efficient operation is the choice of an algorithm for the placement of data across many hosts and subsequent access to it in a manner that balances the workload and ensures availability without adding undue overheads.

Computers and network connections owned and managed by a multitude of different users and organizations are necessarily volatile resources; their owners do not guarantee to keep them switched on, connected and fault-free. So the availability of the processes and computers participating in peer-to-peer systems is unpredictable.

Peer-to-peer services therefore cannot rely on guaranteed access to individual resources, although they can be designed to make the probability of failure to access a copy of a replicated object arbitrarily small. It is worth noting that this weakness of peer-to-peer systems can be turned into a strength if the replication of resources that it calls for is exploited to achieve a degree of resistance to tampering.

Computer clouds

Cloud computing delivers infrastructure, platform, and software (application) as services, which are made available as subscription-based services in a pay-as-you-go model to consumers. The services provided over the cloud can be generally categorized into three different service models: namely IaaS, Platform as a Service (PaaS), and Software as a Service (SaaS).

These form the three pillars on top of which cloud computing solutions are delivered to end users. All three models allow users to access services over the Internet, relying entirely on the infrastructures of cloud service providers.

These models are offered based on various SLAs between providers and users. In a broad sense, the SLA for cloud computing is addressed in terms of service availability, performance, and data protection and security. Below Figure illustrates three cloud models at different service levels of the cloud.

SaaS is applied at the application end using special interfaces by users or clients. At the PaaS layer, the cloud platform must perform billing services and handle job queuing, launching, and monitoring services.

At the bottom layer of the IaaS services, databases, compute instances, the file system, and storage must be provisioned to satisfy user demands.

Infrastructure as a Service(IaaS)

• The capability provided to the consumer is to provision processing, storage, networks, and other fundamental computing resources.

• Consumer is able to deploy and run arbitrary software, which may include operating systems and applications.

• The consumer does not manage or control the underlying cloud infrastructure but has control over operating systems, storage, deployed applications, and possibly limited control of select networking components (e.g., host firewalls)

IaaS providers

• Amazon Elastic Compute Cloud (EC2)
– Each instance provides 1-20 processors, upto 16 GB RAM, 1.69TB storage
• RackSpace Hosting
– Each instance provides 4 core CPU, upto 8 GB RAM, 480 GB storage
• Joyent Cloud
– Each instance provides 8 CPUs, upto 32 GB RAM, 48 GB storage
• Go Grid
– Each instance provides 1-6 processors, upto 15 GB RAM, 1.69TB storage

Platform as a Service (PaaS)

• The capability provided to the consumer is to deploy onto the cloud infrastructure consumer created or acquired applications created using programming languages and tools supported by the provider.

• The consumer does not manage or control the underlying cloud infrastructure including network, servers, operating systems, or storage, but has control over the deployed applications and possibly application hosting environment configurations.

PaaS providers

• Google App Engine
– Python, Java, Eclipse
• Microsoft Azure
– .Net, Visual Studio
• Sales Force
– Apex, Web wizard
• TIBCO,
• VMware,
• Zoho

Software as a Service (SaaS)

• The capability provided to the consumer is to use the provider's applications running on a cloud infrastructure.

• The applications are accessible from various client devices through a thin client interface such as a web browser (e.g., web-based email).

• The consumer does not manage or control the underlying cloud infrastructure including network, servers, operating systems, storage, data or even individual

application capabilities, with the possible exception of limited user specific application configuration settings.

SaaS providers

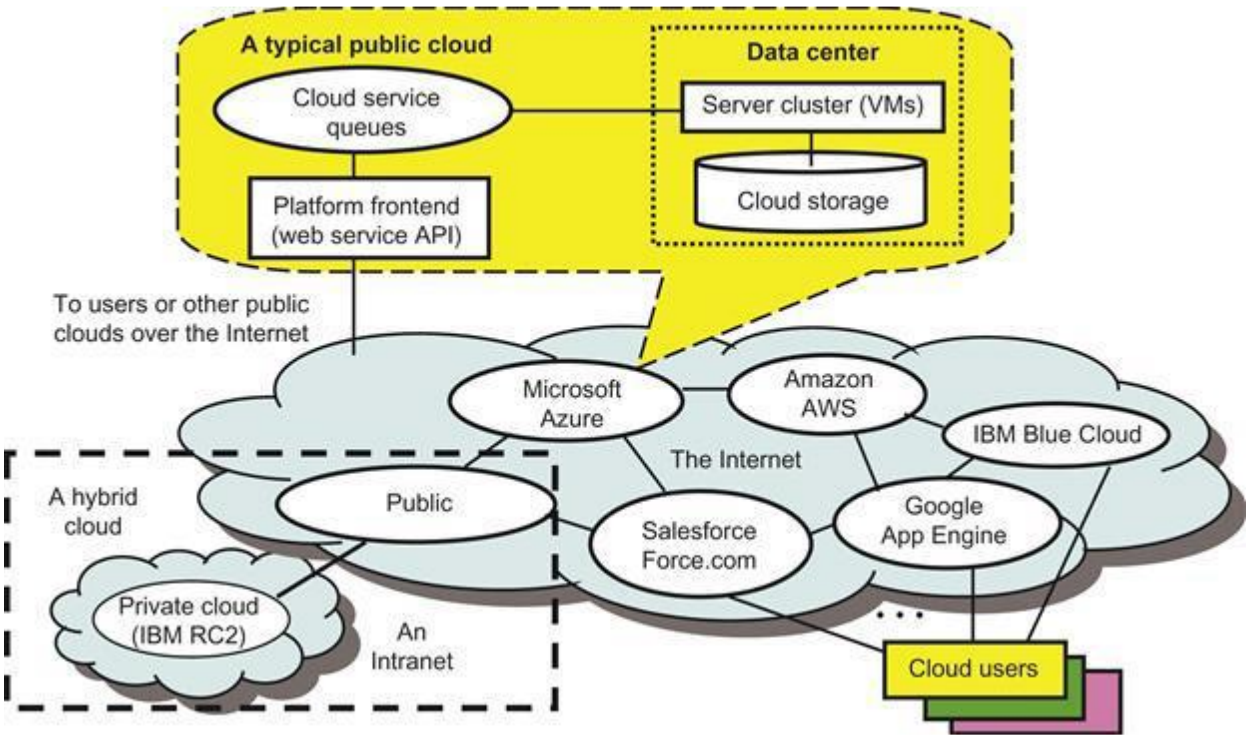
• Google's Gmail, Docs, Talk etc
• SalesForce,
• Yahoo
• Facebook

Cloud deployment models

A cloud deployment model represents a specific type of cloud environment, primarily distinguished by ownership, size, and access.

There are four common cloud deployment models:

1. Public Clouds
2. Community Clouds
3. Private Clouds
4. Hybrid Clouds



Public Clouds

A **public cloud** is built over the Internet and can be accessed by any user who has paid for the service. Public clouds are owned by service providers and are accessible through a subscription.
Many public clouds are available, including Google App Engine (GAE), Amazon Web Services (AWS), Microsoft Azure, IBM Blue Cloud. The providers of the aforementioned clouds are commercial providers that offer a publicly accessible remote interface for creating and managing VM instances within their proprietary infrastructure. A public cloud delivers a selected set of business processes. The application and infrastructure services are offered on a flexible price-per-use basis.

The advantages of a public cloud are:
- Unsophisticated setup and use
- Easy access to data
- Flexibility to add and reduce capacity
- Cost-effectiveness
- Continuous operation time
- 24/7 upkeep
- Scalability
- Eliminated need for software

The disadvantages of a public model:
- Data security and privacy
- Compromised reliability
- The lack of individual approach

Private Clouds

A private cloud is built within the domain of an intranet owned by a single organization. Therefore, it is client owned and managed, and its access is limited to the owning clients and their partners. Its deployment was not meant to sell capacity

over the Internet through publicly accessible interfaces. Private clouds give local users a flexible and agile private infrastructure to run service workloads within their administrative domains. A private cloud is supposed to deliver more efficient and convenient cloud services. It may impact the cloud standardization, while retaining greater customization and organizational control.

Advantages

– Provides virtualized services
– Maximizes hardware usage
– Reduces complexity
– Trust on Data (secured data)

Disadvantages

– Higher Cost

Hybrid Clouds

A hybrid cloud is built with both public and private clouds. Private clouds can also support a hybrid cloud model by supplementing local infrastructure with computing capacity from an external public cloud. For example, the *Research Compute Cloud* (RC2) is a private cloud, built by IBM, that interconnects the computing and IT resources at eight IBM Research Centers scattered throughout the United States, Europe, and Asia. A hybrid cloud provides access to clients, the partner network, and third parties. In summary, public clouds promote standardization, preserve capital investment, and offer application flexibility. Private clouds attempt to achieve customization and offer higher efficiency, resiliency, security, and privacy. Hybrid clouds operate in the middle, with many compromises in terms of resource sharing.

Community Clouds

The cloud infrastructure is shared by several organizations and supports a specific community that has shared concerns (e.g., mission, security requirements, policy, or compliance considerations).
– e.g….IEEE standards on cybernetics (community is all cyber crime detecting agencies)

– Banks and their users (traders)

– Health services (Hospitals , Medical colleges and Health dept)
GoogleApp for Government

– Microsoft's Govt Community Cloud

Advantages

Cost of setting up a communal cloud versus individual private cloud can be cheaper due to the division of costs among all participants.

– Management of the community cloud can be **outsourced** to a cloud provider. The advantage here is that the provider would be an **impartial** third party that is bound by contract and that has no preference to any of the clients involved other than what is contractually mandated.

– Tools residing in the community cloud can be **used to leverage** the **information** stored to serve consumers and the supply chain, such as return tracking and just-in-time production and distribution.

Drawbacks of community cloud:
– Costs higher than public cloud.

– Fixed amount of bandwidth

– Fixed Data storage

Ethical issues in cloud computing:
Cloud computing is based on a paradigm shift with profound implications on computing ethics. The main elements of this shift are:

(i) the control is relinquished to third party services;

(ii) the data is stored on multiple sites administered by several organizations; and

(iii) multiple services interoperate across the network.

Unauthorized access, data corruption, infrastructure failure, or unavailability are some of the risks related to relinquishing the control to third party services; moreover, it is difficult to identify the source of the problem and the entity causing it.

The complex structure of cloud services can make it difficult to determine who is responsible in case something undesirable happens. In a complex chain of events or systems, many entities contribute to an action with undesirable consequences, some of them have the opportunity to prevent these consequences, and therefore no one can be held responsible, the so-called "problem of many hands."

Cloud service providers have already collected petabytes of sensitive personal information stored in data centers around the world. The acceptance of cloud computing therefore will be determined by privacy issues addressed by these companies and the countries where the data centers are located. Privacy is affected by cultural differences, while Western cultures favor privacy, other cultures emphasis community.

Accountability is a necessary ingredient for cloud computing; adequate information about how data is handled within the cloud and about allocation of responsibility are key elements to enforcing ethics rules in cloud computing. Recorded evidence allows us to assign responsibility.

Unwanted dependency on a cloud service provider, the so-called vendor lock-in, is a serious concern and the current standardization efforts at NIST attempt to address this problem. Another concern for the users is a future with only a handful of companies which dominate the market and dictate prices and policies.

Cloud computing vulnerabilities

When deciding to migrate to the cloud, we have to consider the following cloud vulnerabilities.

Session Riding: Session riding happens when an attacker steals a user's cookie to use the application in the name of the user. An attacker might also use CSRF attacks in order to trick the user into sending authenticated requests to arbitrary web sites to achieve various things.

Virtual Machine Escape: In virtualized environments, the physical servers run multiple virtual machines on top of hypervisors. An attacker can exploit a hypervisor remotely by using a vulnerability present in the hypervisor itself – such vulnerabilities are quite rare, but they do exist.

Additionally, a virtual machine can escape from the virtualized sandbox environment and gain access to the hypervisor and consequentially all the virtual machines running on it.

Reliability and Availability of Service: We expect our cloud services and applications to always be available when we need them, which is one of the reasons for moving to the cloud.
But this isn't always the case, especially in bad weather with a lot of lightning where power outages are common.
The CSPs have uninterrupted power supplies, but even those can sometimes fail, so we can't rely on cloud services to be up and running 100% of the time. We have to take a little downtime into consideration, but that's the same when running our own private cloud.

The following are the top security threats in a cloud environment:

Malicious Insiders: Employees working at cloud service provider could have complete access to the company resources. Therefore cloud service providers must have proper security measures in place to track employee actions like viewing a customer's data.
Since cloud service provides often don't follow the best security guidelines and don't implement a security policy, employees can gather confidential information from arbitrary customers without being detected.

Shared Technology Issues: The cloud service SaaS/PasS/IaaS providers use scalable infrastructure to support multiple tenants which share the underlying infrastructure. Directly on the hardware layer, there are hypervisors running multiple virtual machines, themselves running multiple applications.

Data Loss: The data stored in the cloud could be lost due to the hard drive failure. A CSP could accidentally delete the data, an attacker might modify the data, etc. Therefore, the best way to protect against data loss is by having a proper data backup, which solves the data loss problems.

Denial of Service: An attacker can issue a denial of service attack against the cloud service to render it inaccessible, therefore disrupting the service. There are a number of ways an attacker can disrupt the service in a virtualized cloud environment: by using all its CPU, RAM, disk space or network bandwidth.

Challenges for cloud computing:

The development of efficient cloud applications inherits the challenges posed by the natural imbalance between computing, I/O, and communication bandwidths of physical systems;

These challenges are greatly amplified due the scale of the system, its distributed nature, and by the fact that virtually all applications are data-intensive.

One of the main advantages of cloud computing, the shared infrastructure, could also have a negative impact as perfect performance isolation is nearly impossible to reach in a real system, especially when the system is heavily loaded. The performance of virtual machines fluctuates based on the load, the infrastructure services, the environment including the other users.

Many applications consist of multiple stages; in turn, each stage may involve multiple instances running in parallel on the systems of the cloud and communicating among them. Thus, efficiency, consistency, and communication scalability of communication are major concerns for an application developer.

The data storage plays a critical role in the performance of any data-intensive application; the organization of the storage, the storage location, as well as, the storage bandwidth must be carefully analyzed to lead to an optimal application performance.

Many data-intensive applications use metadata associated with individual data records; for example, the metadata for a JPEG audio file may include the name of the song, the singer, recording information and so on.

Metadata should be stored for easy access, the storage should be scalable, and reliable. Another important consideration for the application developer is logging.

Performance considerations limit the amount of data logging, while the ability to identify the source of unexpected results and errors is helped by frequent logging.

Logging is typically done using instance storage preserved only for the lifetime of the instance thus, measures to preserve the logs for a post-mortem analysis must be taken.

Parallel and Distributed Systems: Cloud computing is based on a large number of ideas and experience accumulated since the first electronic computer was used to solve computationally challenging problems.

Cloud computing is intimately tied to parallel and distributed computing. All cloud applications are based on the client-server paradigm with a relatively simple software, a thin-client, running on the user's machine while the computations are carried out on the cloud. Many cloud applications are data-intensive and use a number of instances which run concurrently.

Transaction processing systems, e.g., Web-based services, represent a large class of applications hosted by computing clouds; such applications run multiple instances of the service and require reliable and an in-order delivery of messages.

Parallel computing

As demonstrated by nature, the ability to work in parallel as a group represents a very efficient way to reach a common target; the human beings have learned to aggregate themselves, and to assemble man-made devices in organizations where each entity may have modest ability, but a network of entities can organize themselves to accomplish goals that an individual entity cannot.

Parallel computing allows us to solve large problems by splitting them into smaller ones and solving them concurrently. Parallel computing was considered for many years the holly grail for solving data-intensive problems encountered in many areas of science, engineering, and enterprise computing;

it requires major advances in several areas including, algorithms, programming languages and environments, and computer architecture.

Parallel hardware and software systems allow us to solve problems demanding more resources than those provided by a single system and, at the same time, to reduce the time required to obtain a solution.

The subtasks of a parallel program are called processes while threads are light-weight subtasks. Concurrent execution could be very challenging, e.g., it could lead to race conditions, un undesirable effect when the results of concurrent execution depend on the sequence of events. Often, shared resources must be protected by locks to ensure serial access.

Another potential problem for concurrent execution of multiple processes is the presence of deadlocks; a deadlock occurs when processes competing with one another for resources are forced to wait for additional resources held by other processes and none of the processes can finish.

Distributed systems

The systems we analyze are distributed and in this section we introduce basic concepts necessary to understand the problems posed by the design of such systems.

A distributed system is a collection of autonomous computers, connected through a network and distribution software called middleware, which enables computers to coordinate their activities and to share the resources of the system; the users perceive the system as a single, integrated computing facility.

A distributed system has several characteristics:

its components are autonomous, scheduling and other resource management and security policies are implemented by each system, there are multiple points of control and multiple

points of failure, and the resources may not be accessible at all times. Distributed systems can be scaled by adding additional resources and can be designed to maintain availability even at low levels of hardware/software/network reliability.
Distributed systems have been around for several decades. For example, distributed file systems and network file systems have been used for user convenience and to improve reliability and functionality of file systems for many years. Modern operating systems allow a user to mount a remote file system and access it the same way a local file system is accessed, yet with a performance penalty due to larger communication costs.

The middleware should support a set of desirable properties of a distributed system:

• Access transparency - local and remote information objects are accessed using identical operations;
• Location transparency -information objects are accessed without knowledge of their location;
• Concurrency transparency - several processes run concurrently using shared information objects without interference among them;
• Replication transparency - multiple instances of information objects are used to increase reliability without the knowledge of users or applications; 25
• Failure transparency - the concealment of faults;
• Migration transparency - the information objects in the system are moved without affecting the operation performed on them;
• Performance transparency - the system can be reconfigured based on the load and quality of service requirements;

Scaling transparency - the system and the applications can scale without a change in the system structure and without affecting the applications.

Parallel computer architecture

parallel computer architectures starts with the recognition that parallelism at different levels can be exploited. These levels are:
Bit-level parallelism. The number of bits processed per clock cycle, often called a word size, has increased gradually from 4-bit processors to 8-bit, 16-bit, 32-bit, and, since 2004, 64-bit. This has reduced the number of instructions required to process larger operands and allowed a significant performance improvement. During this evolutionary process the number of address bits has also increased, allowing instructions to reference a larger address space.
Instruction-level parallelism. Today's computers use multi-stage processing pipelines to speed

up execution. Once an n-stage pipeline is full, an instruction is completed at every clock cycle.

A "classic" pipeline of a Reduced Instruction Set Computing (RISC) architecture consists of five stages2: instruction fetch, instruction decode, instruction execution, memory access, and write back. A Complex Instruction Set Computing (CISC) architecture could have a much large number of pipelines stages; for example, an Intel Pentium 4 processor has a 35-stage pipeline.

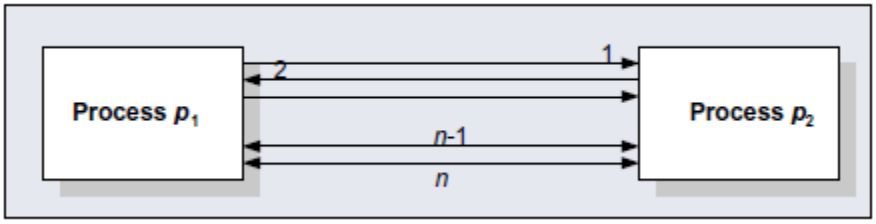
Data parallelism or loop parallelism. The program loops can be processed in parallel.

Task parallelism. The problem can be decomposed into tasks that can be carried out on currently. A widely used type of task parallelism is the Same Program Multiple Data (SPMD) paradigm. As the name suggests, individual processors run the same program but on different segments of the input data. Data dependencies cause different flows of control in individual tasks.

Communication protocols

A major concern in any parallel and distributed system is communication in the presence of channel failures. There are multiple modes for a channel to fail, and some lead to messages being lost. In the general case, it is impossible to guarantee that two processes will reach an agreement in case of channel failures.
Given two processes p1 and p2 connected by a communication channel that can lose a message with probability $\varepsilon > 0$, no protocol capable of guaranteeing that two processes will reach agreement exists, regardless of how small the probability ε is.

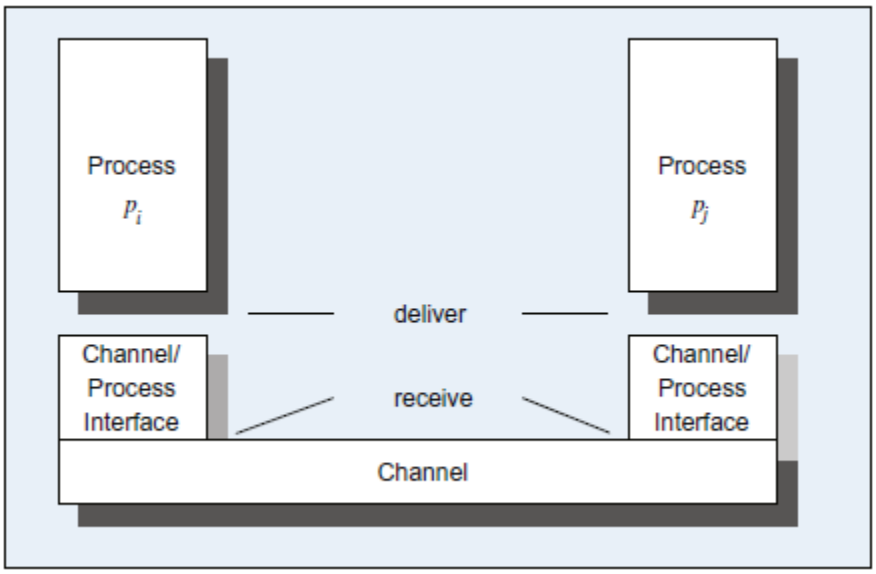
error detection and error correction allow processes to communicate reliably though noisy digital channels. The redundancy of a message is increased by more bits and packaging a message as a code word; the recipient of the message is then able to decide if the sequence of bits received is a valid code word and, if the code satisfies some distance properties, then the recipient of the message is able to extract the original message from a bit string in error



Message delivery rules

The communication channel abstraction makes no assumptions about the order of messages; a reallife network might reorder messages. This fact has profound implications for a distributed application. Consider for example a robot getting instructions to navigate from a monitoring facility with two messages, "turn left" and "turn right," being delivered out of order.

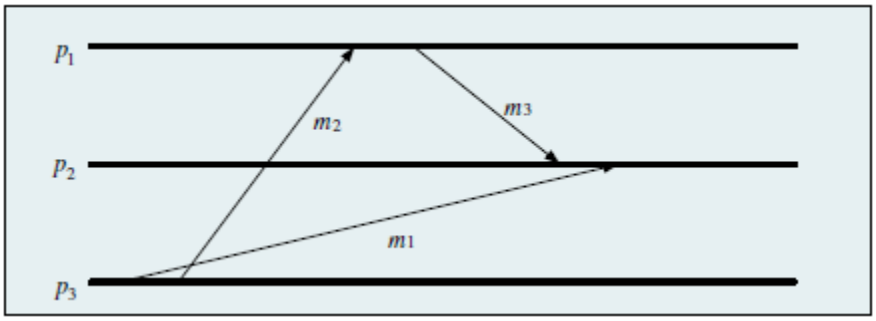
Message receiving and message delivery are two distinct operations; a delivery rule is an additional assumption about the channel-process interface. This rule establishes when a message received is actually delivered to the destination process. The receiving of a message m and its delivery are two distinct.



Message receiving and message delivery are two distinct operations. The channel-process interface implements the delivery rules (e.g., FIFO delivery).

When more than two processes are involved in a message exchange, the message delivery may be FIFO but not causal

- *deliver(m3)* → *deliver(m1)*, according to the local history of process *p2*.
- *deliver(m2)* → *send(m3)*, according to the local history of process *p1*.
- *send(m1)* → *send(m2)*, according to the local history of process *p3*.



Violation of causal delivery when more than two processes are involved. Message m1 is delivered to process p2 after message m3, though message m1 was sent before m3. Indeed, message m3 was sent by process p1 after receiving m2, which in turn was sent by process p3 after sending message m1.

- *send(m2) → deliver(m2).*
- *send(m3) → deliver(m3).*

Logical clocks

Logical Clocks refer to implementing a protocol on all machines within your distributed system, so that the machines are able to maintain consistent ordering of events within some virtual timespan.

A logical clock is a mechanism for capturing chronological and causal relationships in a distributed system. Distributed systems may have no physically synchronous global clock, so a logical clock allows global ordering on events from different processes in such systems.

A logical clock is an abstraction necessary to ensure the clock condition in the absence of a global clock. Each process pi maps events to positive integers.

Call LC(e) the local variable associated with event e. Each process time-stamps each message m sent with the value of the logical clock at the time of sending, T S(m) = LC(send(m)).
The rules to update the logical clock are specified by Equation

LC(e) := ½ LC + 1 if e is a local event or a send(m) event max(LC, T S(m) + 1) if e = receive(m).

The concept of logical clocks is illustrated in Figure using a modified space-time diagram where the events are labelled with the logical clock value.

Messages exchanged between processes are shown as lines from the sender to the receiver; the communication events corresponding to sending and receiving messages are marked on these diagrams.

Each process labels local events and send events sequentially until it receives a message marked with a logical clock value larger than the next local logical clock value.

Concurrency

Concurrency means that several activities are executed simultaneously. Concurrency allows us to reduce the execution time of a data-intensive problem

Concurrency is a critical element of the design of system software. The kernel of an operating system exploits concurrency for virtualization of system resources such as the processor and the memory.

Hiding latency and performance enhancement (e.g., schedule a ready-to-run thread when the current thread is waiting for the completion of an I/O operation).

• Avoiding limitations imposed by the physical resources (e.g., allow an application to run in a virtual address space of a standard size rather than be restricted by the physical memory available on a system).

• Enhancing reliability and performance

Sometimes concurrency is used to describe activities that appear to be executed simultaneously, though only one of them may be active at any given time, as in the case of processor virtualization, when multiple threads appear to run concurrently on a single processor. A thread can be suspended due to an external event and a context switch to a different thread takes place. The state of the first thread is saved and the state of another

thread ready to proceed is loaded and the thread is activated. The suspended thread will be reactivated at a later point in time.

Modelling concurrency with Petri Nets

In 1962 Carl Adam Petri introduced a family of graphs, the so-called Petri Nets (PNs) [199]. PNs are bipartite graphs populated with tokens that flow through the graph and used to model the dynamic rather than static behavior of systems, e.g. detect synchronization anomalies.

A bipartite graph is one with two classes of nodes; arcs always connect a node in one class with one or more nodes in the other class.

In the case of Petri Nets the two classes of nodes are places and transitions thus, the name Place-Transition (P/T) Nets often used for this class of bipartite graphs; arcs connect one place with one or more transitions or a transition with one or more places

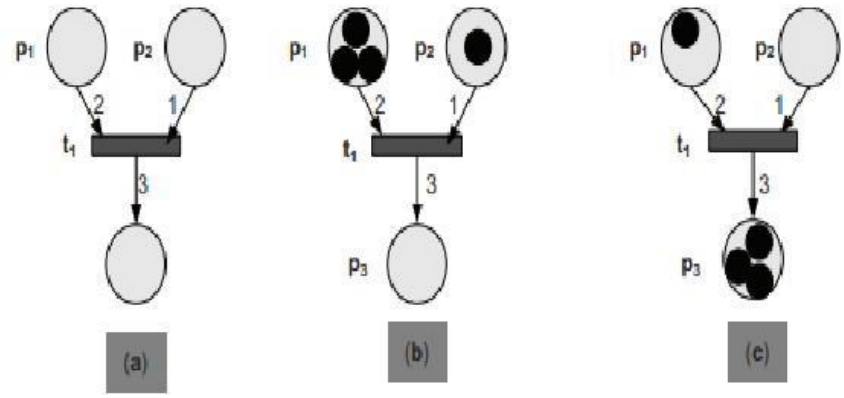


Figure : Petri Nets firing rules

(a) An unmarked net with one transition t1 with two input places, p1 and p2, and one output place, p3.
(b) The marked net, the net with places populated by tokens; the net before firing the enabled transition t1.
(c) The marked net after firing transition t1, two tokens from place p1 and one from place p2 are removed and transported to place p3.

To model the dynamic behavior of systems, the places of a Petri Net contain tokens; firing of transitions removes tokens from the input places of the transition and adds them to its output places, see Figure .

Petri Nets can model different activities in a distributed system; a transition may model the occurrence of an event, the execution of a computational task, the transmission of a packet, a logic statement, and so on.
The input places of a transition model the preconditions of an event, the input data for the computational task, the presence of data in an input buffer, the pre-conditions of a logic statement.

The output places of a transition model the post-conditions associated with an event, the results of the computational task, the presence of data in an output buffer, or the conclusions of a logic statement.

PNs are very powerful abstractions and can express both concurrencies.