Distributed Operating Systems

FINAL REPORT

Kartikey Garg, Richa Sharma, Visalakshi Gopalakrishnan
CEON 283 - Operating Systems, Fall-2015

School of Engineering, Santa Clara University, California 95053-0583

Abstract:

Distributed processing though seems like a recent development has been in focus of research institutions from the late 1980s when changes in hardware, rapid price drop and increase in affordability of computers proliferated the markets.

While the differentiation between distributed systems and distributed operating systems was fuzzy at that time, later distributed OS has been described by the literature as (i) spanning multiple independent, autonomous and communicating CPUs, and (ii) perceived by the user as a single machine [1]. Distributed Operating Systems take advantage of multiplicity of resources along with current networking technologies to provide the benefits of resource sharing across autonomous computers, reliability of operation, speed up of application and communication between geographically distant users.

Our study will explore three main features of Distributed Operating Systems: *Transparency or single image* - Hides the existence of multiple machines and make the collection of networked computers appear to the user as a virtual uniprocessor. *Autonomy* - A system wide operating system where all the computers in the system work as part of this global operating system and *Fault-Tolerance* - Distributed control is an essential feature in Distributed Operating System that ensures in event of failure of one or more machines, entire operation is not halted.

The above abstractions are achieved through key design considerations that enable Distributed Operating System to be an efficient manager. Successful Distributed Operating System should have reliable way of communication amongst process, controlled mapping of events that enable synchronized working resources for the system as a whole. Key design considerations like Resource management, Communication Primitives and Synchronization that enable Distributed Operating System achieve its goal will be explored in this paper.

Development and research interest in Distributed Operating Systems peaked in the 1980's when Andrew S. Tanenbaum envisioned it's relevance and importance, but from then on it has mostly plateaued, until recently, it has once gained interest and many companies like MapR, Facebook, Apple to name a few have invested time in developing and are extensively using Distributed Operating system to manage resources to handle their distributed computing needs.

Introduction:

You know you have a distributed system when the crash of a computer you've never heard of stops you from getting any work done. — **LESLIE LAMPORT**

A distributed system is a collection of nodes that are interconnected by a LAN (Local Area Network) or WAN (Wide Area Network). LAN may be implemented by fiber/cable and WAN may be implemented by satellite communications and media access protocols that may be available for communication of these nodes on LAN/WAN could be via Ethernet, Internet. Individual nodes are independent in a sense they do not share memory, individual clocks need to be synchronized for global efficiency and communication between nodes is through message passing.

Distributed systems have desired key features that define the system as a whole. They need to be heterogeneous, adapt to difference in software and hardware, should work concurrently, be fault tolerant, open to adapt to system growth, avoid centralization to support scalability.

For a long time Network Operating Systems have managed networked computer system, which is an extension of traditional operating system to facilitate resource sharing amongst various nodes. In networked system, users are aware of the nature, locality and the specific system they are using for a required service and each computer in that system runs on its own operating system with no or little fault tolerance.

Instead today distributed programming for the analysis of large data sets, high energy-physics and the deployment of complex distributed services has become the focus and requirements for global resource management with high fault tolerance and transparency are high in demand.

Focus of our study is Distributed Operating System. Distributed Operating System is an operating system that runs on multiple interconnected resources, but presents to the user an image of a very powerful machine and all the computers within the system work in close cooperation with each other as part of this global operating system.

In other words, How to build a system that solve the problem of services needing to communicate with each other in a reliable manner and yet not every developer/user need to understand and write all the particulars of how these things talk to each other ?

Design Features:

1. Transparency:

Distributed Operating system appears to users as a single coherent system. In networked operated system access to resources or transfer of data from remote to local machines is done by File Transfer Protocol mechanism that requires logging in to the appropriate remote machine and only the remote server performs all or most of the transfer of data, resulting in user becoming aware of multiplicity of the machines. Hence, the distinguishing characteristic of Distributed Operating System is that of transparency.

A. **Access Transparency:** Access Transparency means that the user should not be concerned or need to know if the resources (software or hardware) are local or remote. That implies that the Distributed Operating system should let users access remote resource the same way it would if it were a local resource. The user interface should have understandable, easy to implement and contain a well-defined set of system calls that would be meaningful to local as well remote resources. In other words, explicit communications are hidden. Since the system communicate with each other via message passing, it may not be possible to provide complete access transparency due to need to address issues related to communication failures. Keeping syntactical consistency may prove to be challenging because remote access are more complex than local access.

B. **Location Transparency:** The details of the topology or the location of the objects should not be known to the user. The thing that distinguishes access transparency and location transparency is that fact in the former case the user is unaware whether he is accessing local or remote access, whereas in the latter case the name of the resource should not reveal any clue as to the location of the resource namely naming transparency. The name of the resource should be independent of topology of system, the current location of resource. No matter which resource the user is using, they should not be able distinguish the location of the resource by its name. User should also not be required to use different names to access same resource from different locations. Global resource naming should be implemented.

C. **Replication Transparency:** System can provides replicas (additional copies) of resource or files for performance enhancement of the system. The nature and method of replicating resource and placing them on the various components of the system should be transparent to the user. Replication control details like how many copies files/resources at which nodes, where each copy should be placed, when a copy should be created/deleted should be made entirely automatically by Distributed Operating System. Consistency of multiple instances of

files and data. Reading and updating of replicated data will conceived by the user as if it is performed on one single local copy.

D. **Failure Transparency:** Distributed systems are more prone to failures due to its complexities, which may lead to degradation of performance of the system. Failures such as communication link failure, node failure, storage disk failure should be gracefully handled by Distributed Operating System and the intricacies should be hidden to minimize the impact on the user. The user should not be aware of any type of failure other than probably a slow running process till the system resolves the failure.
   The extent of failure transparency in a system depends on the type of failures. Some failures like file server could be hidden because file servers work in cooperation with each other, so that the user can utilize the file service even if only one of the file server is up and running. It may be slow but the service has not halted. In case of failures like communication network failure, hiding the failure might be difficult since it will disrupt the working of the system and will be noticeable by the user. Complete failure transparency has still yet to achieved due to current state of art in Distributed Operating System. It is an ongoing research attempt to achieve complete fault tolerant system.

E. **Migration Transparency:** Migration transparency allows an object (files or process) capable of being moved from one node to another in a distributed system for better performance, reliability, accessibility and security. The goal of migration transparency is to enable the movement of object within the system in a user transparent way. Decisions regarding which objects to be moved where without renaming are handled by the Distributed Operating System. Inter process communication mechanism should ensure that the message reaches the right object even after migrating to the new location without having to resend messages or messages being lost due to object not found at their original location.

F. **Concurrency Transparency:** Like any uniprocessor distributed system face concurrency issues related accessing and sharing of resources. Since the users in the system have to perform job with the limited availability of resources, one user process may influence the action of other concurrently running user process as it competes for resource. Like any uniprocessor concurrent updates to a single shared file should be avoided. Concurrency transparency means that the user feels that they are the only user of the system and other users do not exist and hence issues related to concurrent updates, concurrent sharing are oblivious to them and are handled by the system. Concurrency Transparency can be achieved by ensuring ordered access to resources in the system, by mutual exclusion property, no starvation and no deadlock property.

G. **Performance Transparency:** Since it is economical to share resources, for better performance intelligent resource allocation and usage are required. The goal of performance transparency is to allow automatic reconfiguration of system as loads and resources allocations vary. Fairness is a key factor in better performance of a system. Redistribution of workloads is done in a user transparent way if the system finds one process is overworked while another is idle. The processing capability of the system should be uniformly distributed and redistribute the load if any unbalance in the load distributed is found. User should not be required to handle or concerned about any workload distribution.

H. **Scaling Transparency:** Every system needs to grow and expand as per the needs. Aim of Scaling transparency aims to allow the expansion of the system without interrupting the activities of the users. In order to do that the systems needs to embrace open system architecture and scalable algorithms for designing the components of the Distributed Operating system.

2. Autonomy

In network operating system, each computer in the system has their personal/local operating system and the OS of the various computers in distributed system may or may not be the same. There is centrally no coordination at all between the computers except that they use same communication protocols when two processes of different computers communicate with each other. Every computer determines by itself when to create or terminate its own processes and who its local resources are managed.

In a distributed operating system, there is a distinguished system wide operating system and every individual computer of the distributed operating system operates as a part of this global operating system. The distributed operating system closely associates all the computers of the system in the sense that they work in close cooperation with each other for the efficient and effective utilization of various resources in the system. That is, the processes and several resources are managed globally, and there is a single set of globally valid system calls available on all computers.

The set of system calls is implemented by the kernel of the OS. It is the kernel that manages and controls the hardware to arrange the facilities and resources that are given to other programs. To make the same set of system calls all over accurate, with a distributed operating system identical kernels are run on all the computers of a distributed computing system. The kernels of various computers often cooperate with each other in global decision making, such as finding the most suitable machine for executing the newly created process in the system.

In conclusion it can be said that computers in a distributed operating system have a lesser degree of autonomy and have sense of cooperation than the computers in network operating systems.

3. Fault Tolerance:

It is the ability of the system which allows it to function continuously even in the state of errors. There is a difference between fault tolerant system and Non fault tolerant system In non-fault tolerance the reliability gets improved by choosing only non-faulty devices before an execution starts. In fault tolerant system probability of faults will occur, just ensure that the system continue to function even in the presence of errors. In Classical fault tolerant system mechanisms such as logs and locking were used and has convoluted structure to make a system resilient to face the malicious attacks on these mechanisms. It deals with security in a number of ways like the failure model, the nature of resilience and defense against service denial attacks
Fault is an error which affects the working of the system. Faults are categorized into two categories that are Component faults and Distributed Systems faults.

3.1 Component Faults:

Component faults occur due to a component failure. It can be caused by any design flaw, unexpected inputs, Human error. Component faults are further categorized into several sub-categories with respect to time.

A. **Intermittent fault**: It follows a repetitive cycle of occurrence and disappearance on its own. For example – Loose connection, In this scenario due to loose connector, System will have intermittent fault. Connection will establish and then disappear in a span of time. It is difficult to diagnose and It is very annoying to the user as half of the time the system will run smoothly with no error but some other time the user will face connection errors.
B. ***Transient Fault:*** It's Occurrence is just once or seldom, then it just goes way Re-execution of process with some error can take place normally .Example – Sometimes, a network message doesn't get deliver to the specified destination. But if it gets retransmitted , It will reach the destination.
C. **Permanent Fault:** It will persist until the faulty component gets replaced or repaired. Example – If disk crashes, Data loss error can occurs, It will continue to persist until either the data gets recover using ECC or parity or the disk gets replaced with a new one. Other faults Occur due to Natural disasters like Fire, Flood, Earthquake.

3.2 Failure Models

There are several types of failure models .Few of them are listed First One is Crash failure. In this model the server stops but is working correctly until it stops. Second type is Omission failure. This is further categorized to two types, Receive omission – In this server fails to respond to incoming

requests. Second one is Send Omissions In this server fails to receive incoming messages A server fails to send message . More types are Timing and response failure models.

Different approaches followed to make a system fault tolerant

Redundancy technique: The basic approach behind redundancy technique is to avoid single points of failure by duplicating hardware and software, So that if one of them fails, another will continue to work. Additional System overhead is needed to maintain the consistent copies of the resources. It can be used with hardware, software and time.

A. **Hardware Redundancy**: Addition of more devices to tolerate the loss of some destroyed Component Duplication of critical process takes place on two or more machines, Technique is to provide each process with a backup on some other machine. If one process is being updated, the same should happen to the backup process on other machine, It will help to deal with hardware crash . Either one should work perfectly in this situation .Examples – Redundant array of independent disks – Maintain an exact copy of one disk data to another disk If crash occurs, either of the disk can work appropriately.
B. **Software Redundancy** -Program is structured to collection of modules. Each program runs on N machines in parallel. After each module is executed, Machine compares their results and vote on answer. Software bugs can be voted down.
C. **Time Redundancy** It is achieved by performing an operation several times. Example - Retransmissions and reliable Communication TCP/IP's retransmission of packets. Sender requires acknowledgement from a receiver for each packet If the acknowledgement is not received in a given time duration, the sender retransmits the packet.

3.3 Fault detection and Recovery

A. **Atomic transactions:** Either all of the operations are performed successfully or None of the changes will take place at all. Either it should commit or abort at the end of it's execution. No process can observe the intermediate results that will run simultaneously with the other process. Transaction is a collection of operation that represents a unit of consistency and recovery. There is no possibility that only part of the transaction executes. If a transaction fails or aborts prior to committing, the TP system will undo the effects of any updates (will recover). We either commit or abort the entire process. Checkpointing and Logging and recoverable objects can be used to ensure a transaction is atomic with respect to failures. It makes crash recovery more easier.
B. **Stateless Servers:** In the client server model , A server may be implemented by two ways, either it will follow stateful or stateless approach In stateful servers history of request is taken into consideration. Whereas in stateless servers there is no state that must be restored, a failed server can simply restart after a crash .Thus recovery from crash is much more easier

in case of stateless server. Other benefit is that working of the server remains simple because it does not have to implement the state recording associated with opening, closing, and locking of files.

C. **Acknowledgements and timeout based retransmission of messages:** The messages gets lost during a node crash has to be tracked for their retransmission . Technique which is used to handle lost messages require timeout strategies and return of acknowledgement messages. The receiver should send ack to the sender, In case the sender doesn't receive the ack within certain time period, it can be implied that message has been lost, Thus retransmission for that message will take place.The drawback with this approach is duplicated messages being transmitted between sender and receiver, which can be resolved by ordering of messages.

Design Considerations:

1. Resource Management:

Distributed Systems are described by resource multiplicity and system transparency. The system consists of several resources attached to one another in the network. The distributed Operating system facilitates resource sharing by transferring a local process and executing it at a remote node of network. Resources for example printers, scanners, devices, memory, files, etc. are distributed throughout the system, and at any point of time any of these nodes will have either light or idle workloads.

A resource manager of a distributed operating system schedules the processes in the system, to a pool of free resources that can optimize the resource usage, network congestion, response time, and scheduling overhead. The scheduling decisions are made on various factors such as resource requirements, availability of resources, and static / dynamic state information of the various nodes. . A variety of broadly differing techniques for scheduling these process in a distributed system have been stated. They can be mainly classified into the following three types:

- Task assignment approach is based on prior knowledge of the characteristics of both the processes to be executed and the system. Each process submitted by the user is seen as a collection of related tasks and they are scheduled to appropriate nodes. It mainly deals with the assignment of tasks to various nodes to minimize inter processes communication costs, and improve the turnaround time.
- Load balancing approach makes process assignment decisions in an attempt to equalize the average workload on all nodes. All the process submitted are distributed among the nodes of the system so that it equalizes the workload among all the nodes in the system.

- Load sharing approach attempts to keep all the nodes busy if there is sufficient processes in the system for all the nodes that is it conserves the potential of system to perform work by conforming that no node is vacant while the processes have to wait for being processed.

Desirable features of a good global Process/Resource Scheduler:

Dynamic in nature: A good process-scheduling algorithm should make sure to handle dynamically changing status of various computers of the distributed system. The process/resource assignment decisions should be made on current status of the system and not on some pre-designed policy/strategy.

Quick Decision-making Capability: The scheduler must make appropriate decisions quickly about the assignment of resources/processes to various computers. This is an extremely important aspect of algorithms and makes many potential solutions unsuitable.

Stability: A scheduling algorithm is said to be unsafe if it can enter a state where all the computers are spending all their time migrating process from one to another and not achieving any useful work in an attempt for better performance. This type of waste migration is called as *processor thrashing.*

Scalability: The scheduler should be able to handle small as well as large networks.

Fault Tolerance: A good scheduling algorithm should not be disabled by the crash of one or more computers of the system. At any instance of time, it should continue working for the computers that are in good condition.

1.1 Task Assignment Approach

The task assignment algorithms seek to assign the tasks of a process to the nodes of the distributed system in such a way that it achieves the following goals:

- Minimization of Inter-Process Communication costs.
- Fast turnaround time for the complete process.
- A high degree of parallelism
- Efficient utilization of system resources in general.

These goals would often conflict with each other, such that in order to minimize the inter-process communication we might end up assigning all the work to a single node. While efficient tries to distribute all work among the nodes. Similarly, while increasing the turnaround time and degree of parallelism encourage parallel execution of tasks but the precedence relationship among the processes limits their parallel execution.

1.2 Load Balancing Approach

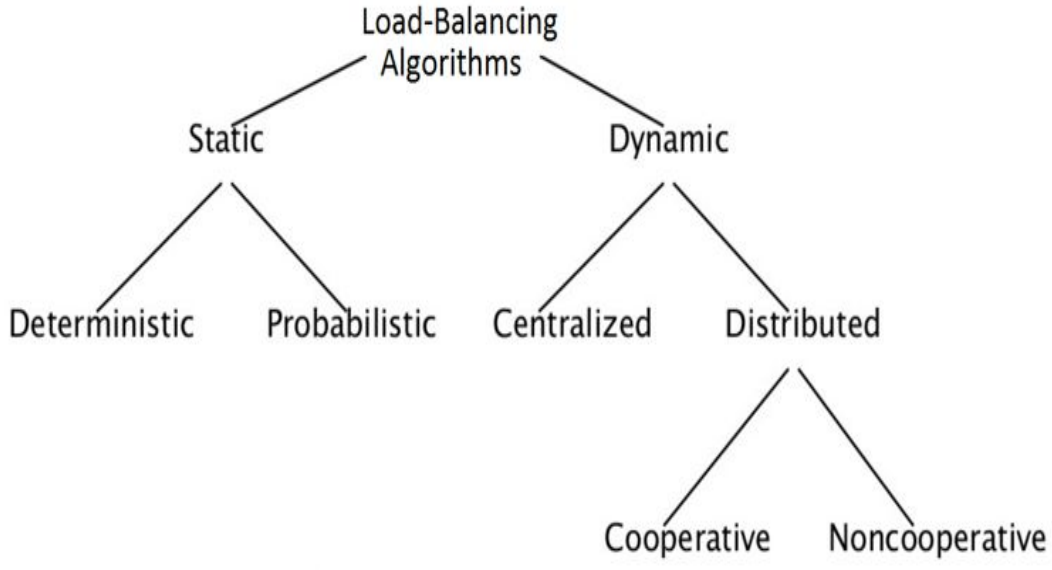
The algorithms using this approach are aimed for better resource management, and it is desirable that load in a distributed system to be distributed uniformly. Thus a load balancing algorithm tries to equitize the total system load by transparently shifting the workload from heavily loaded computers to lightly loaded computers.

Static algorithm use only information about the average behavior of the system, ignoring the current state of the system. While the dynamic algorithms react to the system state that changes dynamically.

Deterministic algorithms use the information about the properties of the nodes and the characteristics of the processes to be scheduled, to allocate process to the nodes. A probabilistic load-balancing algorithm uses information regarding static attributes of the system such as manner of nodes, the processing capability of each node, the network topology, and so on, to formulate a simple process placement rule.

In Centralized Dynamic scheduling algorithm, the responsibility of the scheduling physically resides on a single node. On the other hand, in a Distributed Dynamic scheduling algorithm, the work involves in making the process assignment decisions is physically distributed among the various nodes of the system.

In non-cooperative algorithms, individual entries act as autonomous entities and make scheduling decisions independently of the actions of other entities. While in Cooperative algorithm, distributes entities cooperate with each other to make scheduling decisions.



1.3 Load sharing Approach

Several researchers believe in load balancing, with its implication of attempting to equalize workload on all the nodes of the system, is not an appropriate objective. Its because of the overhead involved in gathering all the information about the states of all the nodes, especially in a very large distributed system having very large number of nodes. It is necessary and sufficient to prevent the nodes from being idle while some other nodes have more than two process. Therefore its rectification is often called *dynamic load sharing* rather than *dynamic load balancing*.

There are several Issues in designing a Load Sharing Algorithm:

Load Estimation Policies: Since this approach attempts to ensure that no node is idle while processes wait for service, it is sufficient to know whether a node is idle or busy. But the simple policy of counting the number of process on a node is not suitable as there are several permanent processes running on a node. Therefore, CPU utilization is the method to be used when we measure load estimation of various systems.
State Information Exchange policies: Since load-sharing algorithms do not aim at equalizing the average load on all nodes, it is not necessary for the nodes to periodically exchange the state information with each other. Rather, a node needs to know the state of other nodes only when it is either under loaded or overloaded.

2. Synchronization:

A distributed operating system has a collection of distinct processes that are structurally separated and run concurrently. It is economical to share resources among concurrently executing processes. That is since number of available resource in a computing system is restricted, one process must necessarily influence the action of other concurrently executing processes as it competes for resources.
Sharing system resources among multiple concurrent processes may be cooperative or non-cooperative in nature. Both cooperative and competitive sharing require adherence to certain rules of behavior that guarantee that correct interaction occurs. The rules for enforcing correct interaction are implemented in the form of synchronization mechanism.
For correct functioning of several distributed applications, the clocks of different nodes of a distributed system must be mutually synchronized as well as with the external world (physical clock).
Clock synchronized algorithms used in distributed systems are broadly categorized into two types – centralized and distributed. In the centralized approach, there is a time server node

and the goal of the algorithm is to keep the clocks of all the other nodes synchronized with the clock time of the time server node. In distributed approach, clock synchronization is done either by globally averaging or localized averaging of the clocks of various nodes of the system.

There are several resources in a system for which exclusive access by a process must be ensured. This exclusiveness of access is called mutual exclusion between processes, and the sections of a program that need exclusive access to shared resources, which are called critical sections.

Three basic approaches used by different algorithm for implementing mutual exclusion in distributed systems are *centralized, distributed* and *token passing.* In the centralized approach, one of the process in the system is elected as the coordinator, which coordinates the entry to the critical sections.

In the distributed approach, all processes that want to enter the same critical section cooperate with each other before reaching a decision on which process will enter the critical section next.

In the token passing approach, mutual exclusion is achieved by using a single token that is circulated among the processes in the system.

Deadlock is the state of permanent blocking of a set of processes each of which is waiting for an event that only other process in the set can cause. In principle, deadlocks in the distributed system are similar to the deadlocks in the centralized system. However, handling of deadlocks in distributed systems is more complex than in centralized systems because the resources, the processes, and other relevant information are scattered on different nodes of the system.

Three commonly used strategies to handle deadlocks are Avoidance, Prevention, and Detection and recovery.

3. Communication Primitives:

For a long time primary communication between networked computers was achieved through communication channels which needed to opened to establish communication by sending/receiving packets of date and closed down when communication was done.

The network itself carried these packets of data between stations and if the data needed to be transfers through WAN, they would pass multiple stations before finally reaching their destinations.

For many years this was primary means of communication. However with the advent of distributed computing the model of communication has changed. The paradigm has shifted towards communicating between computers via messaging called Inter process

communication. It provides a set of message-based protocols and does so by shielding the details of complex network protocols from the programmer.

Desirable features of the message passing system:

Simplicity: Message passing should be simple and easy use. Easy to construct new application and integrate it with the existing one by using the primitives provided by message passing system
Uniform Semantics: Communication should not matter if the process is local or remote
Efficiency: Avoid the costs of establishing and terminating connections between process Minimize the costs of maintaining connections and reducing number of messages exchanges
Reliability: Cope with message transmission failure. Acknowledge message loss, retransmit message on basis of timeouts. Duplicate messages may be sent in event of failures or timeouts
Correctness: Atomicity ensures that message intended for group of receivers will be delivered to either all of them or none of them. Ordered delivery makes sure that message reaches the receiver in an acceptable order. Survivability makes sure that messages reaches the receiver despite link failure, machines or communication failures
Security:Authentication of receiver of message by the sender. Authentication of sender of a message by the receiver and authentication of message before sending and receiving over the network

Two major concerns in message passing are Addressing (or naming) and Failure handling. An important goal in process addressing is to provide location transparency which is achieved by two-level naming scheme for processes. In order to handle failure, the two most commonly used methods are the use of explicit retransmissions based on timeouts and the use of explicit acknowledgment packets.
As distributed computing aims to provide parallelism it is also prone to failure, such as communication crash and loss of messages

Types of message failures are as follows:

Loss of request message: This type of failure happens due to loss of messages or communication link failure between receiver and sender due to the node being down at receiver's end.
Loss of response message: This type of failure happens due to loss of messages or communication link failure between receiver and sender due to the node being down at sender's end.

Unsuccessful execution of message:This happens due to receivers node crashing at the time request was being processed.

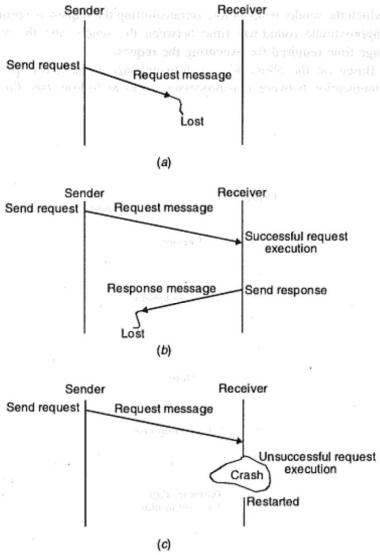


Fig. 3.6 Possible problems in IPC due to different types of system failures. (*a*) Request message is lost. (*b*) Response message is lost. (*c*) Receiver's computer crashed

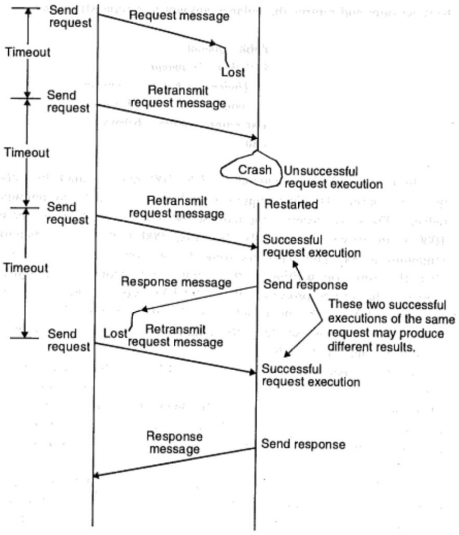
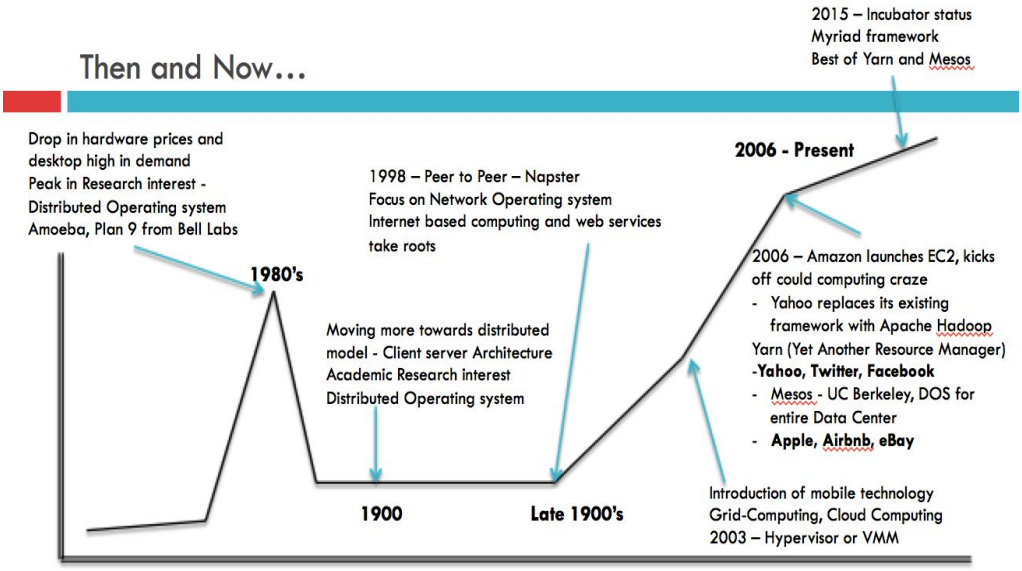


Fig. 3.10 An example of fault-tolerant communication between a client and a server.

The first image shows possible problems in IPC due to different types of failure and the second image shows example of fault-tolerant communication between process

To cope with this problem reliable IPC protocols should be established to ensure that the messages are not loss. The protocol involves retransmission of messages within a specified time if the sender does not get acknowledgement from the receiver about successful message transmission.

The client sends request to the server. On receiving the message the kernels sender starts a timer and the sender finishes processing the request before the timer is expires and the reply sent serves as an acknowledgement. If the message is not received in specified time the kernel of the clients machine retransmits the message. If the client receives the reply it sends the acknowledgement to the kernel of the server if the server machine does not receive acknowledgement it retransmits the message.



Timeline:

Distributed processing though seems like a recent development has been in focus of research institutions from the late 1980's when changes in hardware, rapid price drop and increase in affordability of computers proliferated the markets. There were many assumptions made about the incremental growth of Distributed Operating Systems at that time. We attempt to trace the development of Distributed Operating Systems, where it was and where it is now.

1980': Interest in Distributed Operating systems peaked around mid- 19080's and was focus of many research institutions at that time. Tanenbaum along with many other predicted that with drop in hardware prices, increase in computing powers, distributed computing will become more widespread, both in high-energy physics and in other applications and centralized operating systems will gradually give way to distributed ones. Based on that belief, operating system was developed to handle distributed system for networks of computers that would essentially present the network to the user as if it were a single machine. Examples : Amoeba, Mach, Plan 9 by Bell labs, Chorus

Early 1990's: However towards 1990s it was further away from truth [4]. The concentration was more towards development of traditional single machine operating systems aiming towards their roots in time-shared mainframe systems (UNIX, BSDs, Linux) and systems based on newly developed home microcomputer OSes (such as DOS or MacOS).
Interest in distributed computing remained but the focus was more on Client - Server Architecture, where many clients or remote processor would request and receive service from a centralized host server.
Architecture of a computer network in which many client (remote processors) request and receive service from a centralized server (host computer).

Late 1990's: 1998's saw the shift in the way distributed computing was performed. Client server Architecture gave way to Networked Operating systems. The preference moved from relying on one main centralized servers to a set-up where every computer as part of the network behaved like a client and a host. No central server was needed.
Example - Peer-to-Peer computing implemented by Napster

Early 2003's: Internet of things [Kevin Ashton in 2009] became an indispensable part of daily life. With the increase in wireless network coverage and wireless communication, mobile technology takes roots and rapid developments take place in that area. Example: Cloud computing, Grid computing, Cluster computing, HyperVisor or VMM.

2006 - Present: Amazon launches EC2, kicks of craze in cloud computing. [11]Scalable deployment of of applications by providing web service where user can create, launch and terminate server-instances. After almost a quarter of century interest in Distributed Operating systems kindles again. Around same time Yahoo unsatisfied with its existing framework Apache Hadoop-1.0. Hadoop 1.0 had scalability and utilization issues, some clusters were heavily loaded while some idle. In 2007 Yahoo replaces it with Hadoop-2.0 renamed as YARN (Yet Another Resource Manager). YARN was developed out of need to scale Hadoop. Becomes a resource manager, evaluates resources decides where the jobs go. Highly scalable and fault tolerant. Yahoo, Twitter, Facebook all use YARN for Hadoop related Work.
At same time Apache Mesos a Distributed operating system was developed at UC Berkeley - A complete package for Data Center. Mesos builds gap between the hardware and the application and was built to be a global resource manager for the entire data center.Evaluates resources, increases scalability, fault tolerant and isolation between tasks. Officially given the name of Distributed Operating Systems. Yahoo, Twitter, Facebook, eBay, Airbnb, Apple are some of the major companies that use above system for their distributed computing needs.

Conclusion:

Distributed systems are characterized by scalability, resource multiplicity and system transparency provided by a true Distributed Operating System. Distributed Operating systems looks to the user like a centralized Operating system, but runs on multiple computers, with each computer having its own memory capable of communicating and cooperating with each other through LAN/WAN. Layers of abstractions make it easier for User/Programmer, however increases the complexity of design and implementation of a true Distributed Operating Systems.
Complete transparency as envisioned by Tanenbaum may be difficult to implement and probably expensive for companies to adopt. Developing a system that focuses more on efficient resource management, higher security, higher fault tolerance and efficient Inter -Process Communications

with scalable capabilities is preferred at the cost not having complete transparency. We believe truest form of Distributed Operating System exists only in Research circles. Instead, today we have Distributed Operating System developed out of need to address the specific of growing industry.

- Yarn (Hadoop 2.0) was developed out of need to scale Hadoop-1.0, to be a better and efficient resource manager with better scalable algorithms.
- Mesos was developed to provide complete resource management solution for data center
- The Myriad framework (2015) takes the best of both the Apache Yarn management Layer and Apache Mesos distributed system kernel to develop a system managing both Hadoop –driven work loads and non hadoop driven workloads in the same platform.

Although predicting future is difficult, after almost more than a quarter of century, we believe Distributed Operating system are here to stay and is the future of distributed computing.

References:

1. RESEARCH ISSUES IN DISTRIBUTED OPERATING SYSTEMS, Andrew S. Tanenbaum Robbert van Renesse Dept. of Mathematics and Computer Science Vrije Universiteit Amsterdam, The Netherlands [1985]
2. DISTRIBUTED OPERATING SYSTEMS, CONCEPTS AND DESIGN, by Pradeep K Sinha
3. Distributed Operating Systems Suresh Sridharan CS 739: Distributed Systems University of Wisconsin, Madison 25th January, 2006 Spring 2006
4. New wine in old skins: the case for distributed operating systems in the data center Malte Schwarzkopf, Matthew P. Grosvenor, Steven Hand, University of Cambridge Computer Laboratory, Microsoft Research Silicon Valley
5. Operating system support for distributed applications in real space-time, Matthias Werner, Dirk Müller, Martin Däumler, Jan Richling, Gero Mühl [2008]
6. Distributed Operating Systems SAPE J. MULLENDER University of Twente, Enschede, The Netherlands [1990]
7. Computer Notes - Definition of Distributed Operating System by Dinesh Thakur.
8. [1] http://spark.apache.org/ a fast and general engine for large scale data processing
9. [2] http://mesos.apache.org/ Open source datacenter computing with Apache Mesos, Posted 15 Sep 2014 by Sachin P Bappalige
10. Security Engineering: A Guide to Building Dependable Distributed Systems, Ross J. Anderson
11. https://en.wikipedia.org/wiki/Amazon_Elastic_Compute_Cloud