Elective: III

Distributed Operating System

Unit - I

Motivation and Goals, broad overview and advantages of DS, main characterstics: absense of global clock and state and possibility of large network delays. Issues in DS - transparency, scalability, security, resource management etc. Theoritical foundation - Lamport's clocks Chandy - Lamport Global state recording algorithm - termination detection

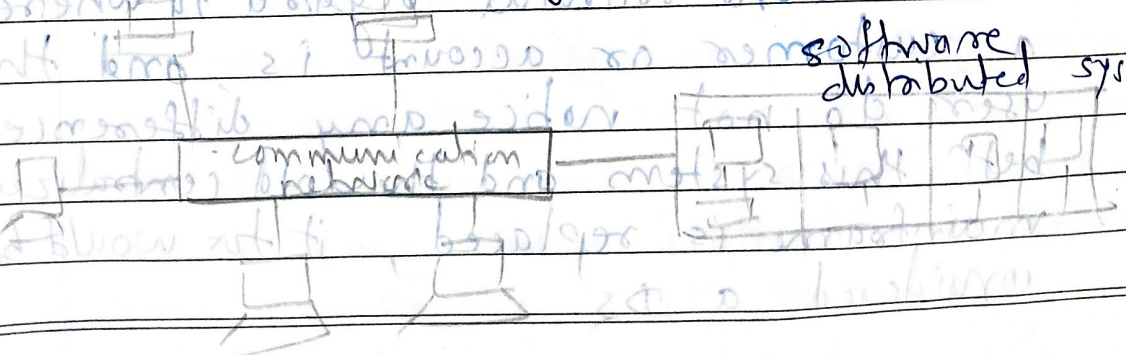
Distributed system

A distributed system is a collection of independent computers that appear to the users of the system as a single computer

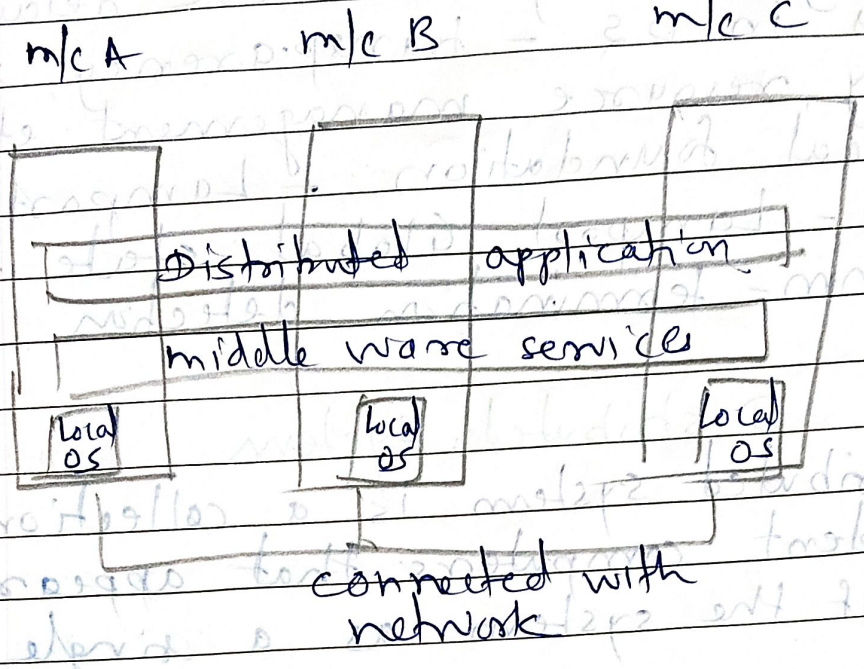
(or)

DS is a collection of independent computers linked by a computer network and equipped with distributed s/m software.

This defn has 2 aspects first one deals with hardware and second deals with software, users think of the system as a single computer.



Distributed system enables computers to coordinate their activities to share resources of system hardware, software & data.



Example: Think about a large bank with hundreds of branch offices all over the world. Each office has a master computer to store local accounts and handle local transactions. In addition, each computer has the ability to talk to all other branch computers and with a central computer at headquarter. If transaction can be done without regard to where a customer or account is and the user do not notice any difference betn this system and the old centralized mainframe is replaced, it too would be considered a DS.

Advantages of Distributed system over centralised system

Item	Description
① Economics	Microprocessor offer a better price or performance than mainframes.
② Speed.	A DS may have more total computing power than a mainframe
③ Inherent Distribution	Some application involve spatially seperated machines
④ Reliability/fault Tolerance	If one machine crashes, the system as a whole can still survive.
⑤ Incremental growth	Computing power can be added in small increments.

① Economics

The real driving force behind the trend toward decentralisation is economics. With up technology, ifor a few hundred dollars you can get a CPU chip that can execute more instructions per second than one of the largest mainframes.

As a result the most cost-effectiv solution is frequently to harness a large number of cheap CPUs together in a system. Thus the leading reason for the trend towards distributed system is that these syst potentially have a much better price/ performance ratio than a single large centralized system would have

② Speed

Collection of µps cannot only give a better price/performance ratio than a single mainframe, but may yeild an absolute performance that no mainframe can achieve at any price. for example with current technology it is possible to build a system from 100,00 modern CPU chips, it would have each of which runs at 50 MIPS ie millions of instructions per second, for a total performance of 5,00,000 MIPS → for a single processor (ie. CPU) to achieve this, it would have to execute an instruction in 0.002 nsec. No existing machine even comes close to this, both theoritically and in considerations make it unlikely that any will ever will. Thus, whether the goal is normal performance at low cost or extremely high performance at greater cost, DS have much to offer

⑤ Inherent Distribution

A next reason for building a distributed system is that some applications are inherently distributed.
A supermarket chain might have some stores, each of which gets goods delievered locally, makes local sales, and makes local decisions about which vegetables are so old or rotten that they must be thrown out. It therefore makes sense to keep track of inventory at each store on a local computer rather than centrally at corporate HQ. After all, most queries & updates will be done locally. Nevertheless, from time to time top management may want to find out how many rutabagas it currently owns. One way to accomplish this goal is to make the complete system look like a single comp to the application programs but implement decentrally, with one computer per store, as we have described. This would then be a commercial distributed system.

④ Reliability

Another potential advantage of DS over a centralized system is higher reliability. By distributing the workload over many machines, a single chip failure will bring

down at most one machine leaving the rest intact. Ideally if 5% of the machines are down at any moment the system should be able to continue to work with a 5% loss in performance. For critical applications such as control of nuclear reactors or aircraft, using a distributed system to achieve high reliability may be the dominant consideration.

⑤ Incremental Growth.

Finally, incremental growth is also potentially a big plus. Often a company will buy a mainframe with the intention of doing all its work on it. If the company prospers and the workload grows, at a certain point the mainframe will no longer be adequate. The only solution are either to replaced the mainframe with a larger one (if it exists) or to add a second mainframe. Both of these can causes major havoc on the company's operation.

In contrast, with a DS, it may be possible to simply add more processors to the system, thus allowing it to expand gradually as the need arises

Disadvantages of Distributed system

Item	Description
Software	Little software exists at present for distributed system
Networking	The network can saturate or cause other problems
Security	Easy access also applies to secret data.

① Software → With the current state-of-the-art, we do not have much experience in designing, implementing and using distributed software. What kinds of OS, prog. language, and applications are appropriate for these systems? How much should the users know about the distribution? How much should the system do and much should the users do? The experts differ (not that this is unusual with experts, but when it comes to DS, they are barely on speaking terms. As more research is done, this problem will diminish, but fo the moment it should not be underestim

(2) Networking

A second potential problem is due to the communication network. It can lose messages, which requires special software to be able to recover, and it can become overloaded. When the network saturates, it must either be replaced or a second one must be added. In both cases, some portion of one or more buildings may have to be rewired at great expense, or network interface boards may have to be replaced (e.g. by fiber optics). Once the system comes to depend on the network, its loss or saturation can negate most of the advantages the distributed system was built to achieve.

(3) Security

The easy sharing of data which is one of the advantage of DS may turn out to be a two-edged sword. If people can conveniently access data that they have no business looking at. In other words security is often a problem

Design Issues

① Transparency

Probably the single most imp issue is how to achieve the single system image. In other words, how do the system designers fool everyone into thinking that the collection of machines is simply an old-fashioned timesharing system? A system that realizes this goal is often said to be transparent

Easiest way of transparency is to hide the distribution from the users.

for example, when a UNIX user types make to recompile a large number of files in a directory, he need not be told that all the compilations are proceeding in parallel on different machines are done using a variety of file servers to do it. To him, the only thing that is unusual is that the performance of the system is halfway decent for a change. In terms of command issued from the terminal and the results displayed on the terminal, the DS can be made to look just like a single processor system

What does transparency really mean?

It is one of those slippery concepts that sounds reasonable but it is more subtle than it at first appears.

As an example imagine a DS consisting of

workstations each running some standard OS. Normally system services are obtained by issuing a system call that traps to the kernel. In such a system remote files should be accessed the same way.

A system in which remote files are accessed by explicitly setting up a network connection to a remote server and then sending messages to it is not transparent because remote services are then being accessed differently than local ones. The programmer can tell that multiple machines are involved, and this is not allowed.

The concept of transparency can be applied to several aspects of a DS

Kind	Meaning
Location transparency	The users cannot tell where resources are located

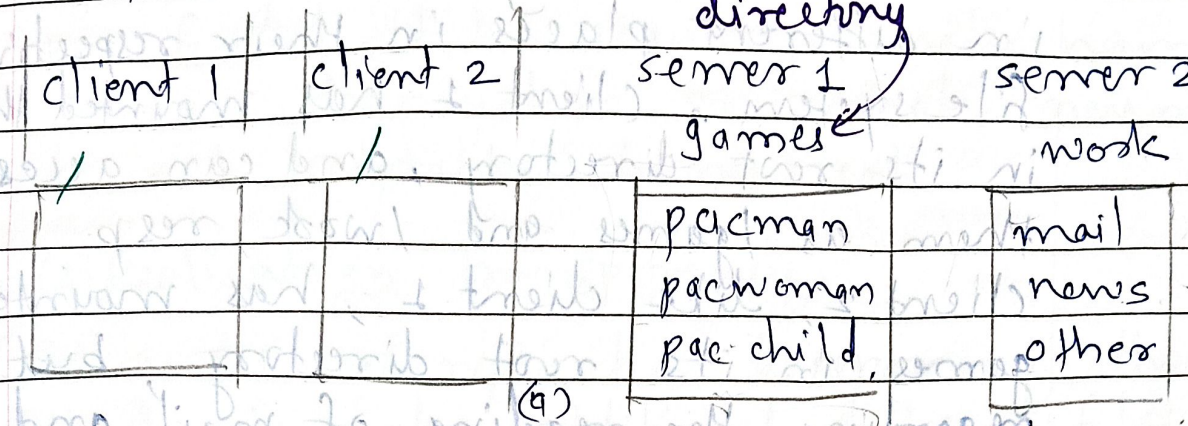
Location transparency refers to the fact that in a true DS users cannot tell where hardware and software resources such as CPUs, printers, files, and databases are located. The name of the resource must not secretly encode the location of the resource.

② Mitigation transparency

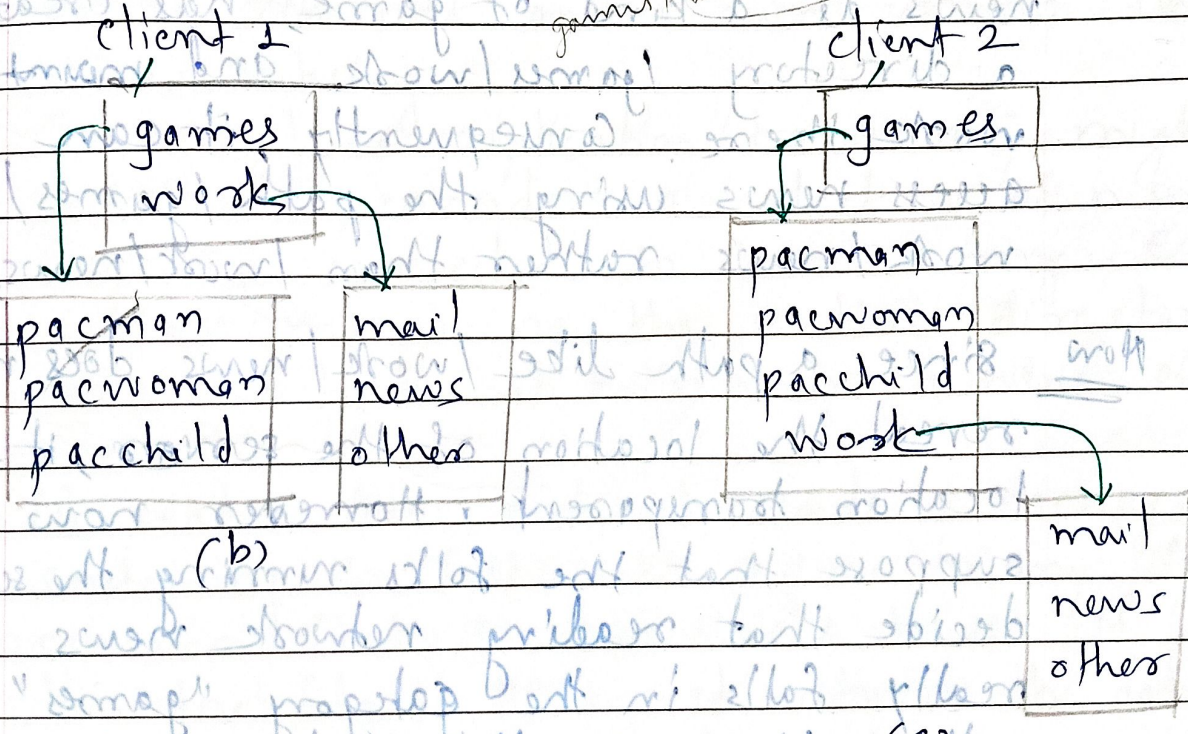
Resources can move at will without changing their names.

Mitigation transparency means that resources must be free to move from one location to another without having their name change.

ex:



(a)



(b) (c)

File servers generally maintain hierarchical file systems, each with a root directory containing subdirectories and files.

Here, two file servers are shown. One has a directory called games, while the other has directory called work. These directories each contain several files. Both of the clients shown have mounted both of the servers, but they have mounted them in different places in their respective file system. Client 1 has mounted them in its root directory, and can access them as /games and /work, resp. Client 2, like client 1, has mounted games in its root directory but regarding the reading of mail and news as a kind of game, has created a directory /games/work and mounted work there. Consequently, it can access news using the path /games/work/news rather than /work/news.

Now since a path like /work/news does not reveal the location of the servers, it is location transparent. However now suppose that the folks running the server decide that reading network news really falls in the category "games" rather than in the category "work". Accordingly they move news from server 2 to server 1.

The next time client 1 boots and mounts the servers in his customary way, he will notice that /work/ newes no longer exists.
Instead there is a new entry /games/news
Thus the mere fact that a file or directory has migrated from one server to another has forced it to acquire a new name because the system of remote mounts is not migration transparent

Ring ③ Replication transparency
If a DS has replication transparency the OS is free to make additional copies of files and other resources on its own without the users noticing.
To see how replication transparency might be achievable, consider a collection of n servers logically connected to form a ring. Each server maintains the entire directory tree structure but holds only a subset of the file themselves.
To read a file, a client sends a message containing the full path name to any of the servers. That server checks to see if it has the file. If so it returns the data requested. If not, it forwards the request to the next server in the ring, which then repeats the algorithm.

In this system, the servers can decide by themselves to replicate any file on any or all servers, without the users having to know about it. Such a scheme is replication transparent because it allows the system to make copies of heavily used files without the users even being aware that this is happening.

④ Concurrency transparent

DS usually have the multiple independent users. What should the system do when two or more users try to access the same resource at the same time?

for ex what happens if two users try to update the same file at the same time?

If the system is concurrency transparent the users will not notice the existence of other users.

One mechanism for achieving this form of transparency would be for the system to lock a resource automatically once someone had started to use it, unlocking it only when the access was finished. In this manner, all resources would only be accessed sequentially, never concurrently.

⑤ Parallelism transparency

In principle, a DS is supposed to appear to the users as a traditional, uniprocessor timesharing system

What happens if a programmer knows that his DS has 1000 CPUs and he wants to use a substantial fraction of them for a chess prog that evaluates boards in parallel? The theoretical answer is that together the compiler, runtime system, and OS should be able to figure out how to take advantage of this potential parallelism without the programmer even knowing it.

Unfortunately the current state-of-the-art is nowhere near allowing this to happen. Programmers who actually want to use multiple CPUs for a single problem will have to program this explicitly, at least for the foreseeable future. Parallelism transparency can be regarded as the holy grail for DS designers. When that has been achieved, the work will have been completed and it will be time to move on to new fields.

All this not withstanding, there are times when users do not want complete transparency for ex - when a user asks to print a document, he often prefers to have the output appear on the local printer, not one 1000 km away, even if the distant printer is fast, inexpensive, can handle colour & smell and is currently idle

Issue (2) Flexibility

The second key design issue is flexibility. It is important that the system be flexible because we are just beginning to learn about how to build DS.

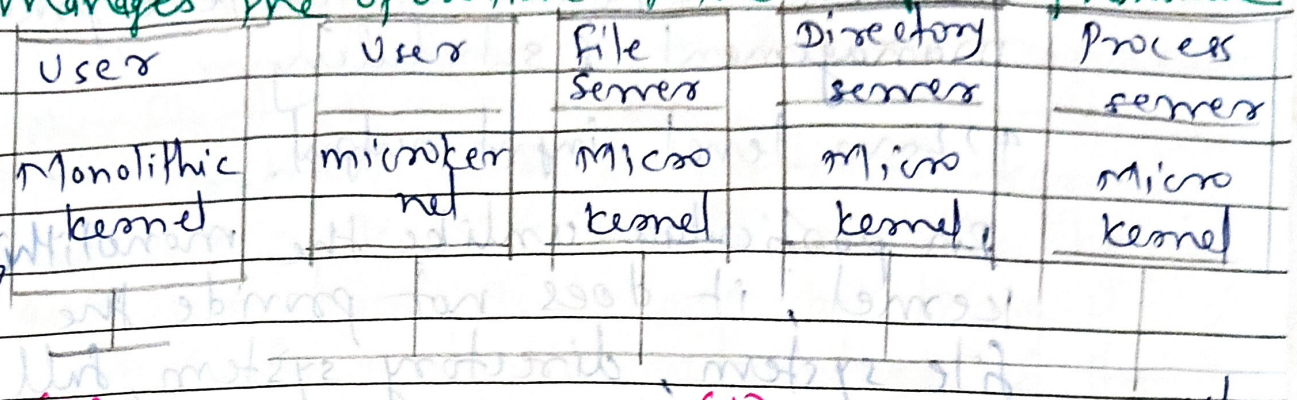
flexibility along with transparency, is like parenthood and apple pie: who could possibly be against them? It is hard to imagine anyone arguing in favour of an inflexible system. However things are not as simple as they seem.

There are two schools of thought concerning the structure of DS.

One school maintains that each machine should run a traditional kernel that provides most services itself. The other maintains that the kernel should provide as little as possible with the bulk of the operating system services available from user level servers.

These two models known as the monolithic kernel and microkernel resp.

Kernel → is a comp program at the core of a computer OS and has complete control over everything in the system
- it manages the operation of the computer & hardware



The monolithic kernel is basically today's centralized operating system augmented with networking facilities and the integration of remote services.
most system calls are made by trapping to the kernel, having the work performed there, and having the kernel return the desired result to the user process
If the monolithic kernel is the reigning champion, the microkernel is the up and coming challenger. Most distributed system that have been designed from scratch use this method.
The microkernel is more flexible because it does almost nothing. It basically provides just four minimal services.
1) An interprocess communication mechanism
2) Some memory management

3) A small amount of low level process management & scheduling
4) Low level input output.

In particular unlike the monolithic kernel, it does not provide the file system, directory system, full process management or much system call handling.
The services that the microkernel does provide are included because they are difficult or expensive to provide.
All the other OS services are generally implemented as user level servers - To look up a name, read a file or obtain some other service, the user sends a message to the appropriate server which then does the work and return the result.
The advantage of this method is that it is highly modular: there is a well defined interface to each service and every service is equally accessible to every client, independent of location.
In addition it is easy to implement, install and debug new services, since adding or changing a service does not require stopping the system and booting a new kernel, as in the case with a monolithic kernel.

It is precisely this ability to add, delete and modify services that gives the microkernel its flexibility.

In contrast with a monolithic kernel the file system is built into the kernel and users have no choice but to use it.

The only potential advantage of the monolithic kernel is performance. Trapping to the kernel and doing everything there may well be faster than sending message to remote servers. However the practices shows that this advantage is nonexistent.

(3) Reliability

One of the original goals of building DS was to make them more reliable than single processor system. The idea is that if a machine goes down, some other machine takes over the job.

It is imp to distinguish various aspects of reliability

Availability → refers to the fraction of the time that the system is usable

A highly reliable system must be highly available but that is not enough. Data entrusted to the system must not be lost in any way and if files are stored redundantly on multiple servers, all the copies must be kept consistent

Another aspect of overall reliability is security. Files and other resources must be protected from unauthorized usage. In a single processor system, the user logs in and is authenticated. From then on, the system knows who the user is. In a DS, when a message comes in to a server asking for something, the server has no simple way of determining who it is from. No name or identification field in the message can be trusted since the sender may be lying. At the very least, considerable care is required here.

(4) Performance

When running a particular application on a DS, it should not be appreciably worse than running the same application on a single processor. Various performance parameters can be used. Response time is one, but so are throughput (no. of jobs per hour), system utilisation, and amount of n/w capacity consumed.

The performance problem is compounded by the fact that

communication, which is essential in a DS is typically quite slow. Sending a message and getting a reply takes time. Thus to optimize performance one often has to minimize the number of messages. The difficulty with this strategy is that the best way to gain performance is to have many activities running in parallel on different processors, but doing so requires sending many messages.

(5) Scalability

Most current DS are designed to work with a few hundred CPUs. It is possible in the future that systems will be orders of larger magnitude and soln that work well for 200 m/c will fail for 2,00,00,00,000 examples, the french PTT (post, telephone and telegraph) administration is in the process of installing a terminal in every household and business in france. The terminal known as minitel, will allow online access to a database containing all the telephone no in a france. Thus eliminating the need for printing and distributing expensive telephone books.

Once all the terminals are in place, the possibility of also using them for electronic mail is readily present.

(2) next comes Interactive access to all kinds of data bases & services for such huge DS one guiding principle is to avoid centralized components tables and algorithms.
centralized tables are almost bad & centralized components..

(3) In a large DS an enormous no of messages have to be routed over many lines. The trouble is that collecting and transporting all the inputs and outputs information. Only decentralized algorithm should be used. These algorithm generally have the following char
1. No machine has complete information about the system state
2. Machine make decision based only on local information
3. failure of one machine does not ruin the algorithm
4. There is no assumption that global clock exists