COMP90015 Distributed Systems

Distributed System Models

School of Computing and Informati

2022 Semester II

1. Modelling Overview

2. Functional Models
   - Processes and Machines
   - Co
   - Ne
   - Dis
   - Sys

3. Non-functional Models
   - Temporal Model
   - Failure Model
   - Security Model

Functional and non-functional models

In this section we will *develop a model* of a distributed system, from first principles.

- Developing the model means specifying a notation for representing a real-world distrib tical) object or conce
- Functi implements functio ection our Functi of the system that are constant over the system's lifetime.

- Non-functional models are concerned with aspects wh uch as time, reliability and security. When we model time then w r Functional models to provide time-dependent Funct

In this section we will demonstrate how the models can be used to make some basic statements of a distributed system's properties and behaviour.

Processes and machines

The *Architectural Model* is, in a broad sense, concerned with the components or entities of the system, the roles that they take, the relations ract with each othe being consider

dware/OS, i.e. the platf

- *Architectural elements* – components of the system that interact with one another
- *Architectural patterns* – the way components are m system

- *Associated middleware solutions* – existing solu

Process Roles

Each process has a high-level purpose or role that it takes in the distributed system. Here are some common examples of process roles:

- User Interface Process – a process that allows users to interact with the distributed system, either graphical or command line based. It usually acts as a client to other processes in the distributed system.
- Data Cache – a process that caches information, typically in memory, for fast access b
- Comm t to in order to under
- Job Ma ributed system processes.
- Index or Registry Server – a process that provides the locati network address, of resources in the distributed system
- Database Server – a process that provides a database serv ase server process allows other processes acting as database clients to connect to it and obtain database services.
- Authentication Server – a process that authenticates users, e.g. by passwords or other security mechanisms. Other processes in the distributed system may require users to authenticate with the Authentication Server prior to providing service.

Middleware

In most cases the roles described earlier can be undertaken by a group of processes that we sometimes refer to as a sub-system e.g.:

- Reliable Key-Value Storage – a fault tolerant general purpose storage system
- Cluster ster of machi
- Enterp erization
- High Pe mechanisms

They are usually referred to as *middleware* system architecture we may choose to use various kind are as sub-systems that solve common problems for us in a dist . We should understand the architecture of the sub-syste cts it has on our distributed system architecture as a whole. Our models help us do this.

Machine Roles

Similarly to processes, machines tends to have well defined roles in the distributed system, and here are some examples:

- Dekstop/PC – runs client processes, has a fixed location at home or at work, user may or may not be at the machine, has no power limitations, has reasonably good resource capacity, usually connected to the network via Ethernet, may be switched on for long periods of time, may or may not have a public I
- Mobile … a all kinds of public r … tion, usually limited r … lar, may be on 24/7, usually wont have a public IP address
- Front End Server – runs server processes that allow client … t, located in a machine room at a fixed location, is only remotel … ted to, has no power limitations, has significant resource capa … high speed Ethernet, on 24/7, has a public IP address
- Back End Server – runs distributed processes to support front end servers, located in a machine room at a fixed location, is not reachable from the public Internet, has no power limitations, has significant compute and storage capacity, connected via high speed Ethernet, on 24/7, has a private address
- Virtual Machine/Server – runs on cloud infrastructure, provides capabilities similar to Front/Back End Servers.

Interfaces and IPC mechanisms

- Distributed processes can not directly access each others internal variables or procedures. Passing parameters needs to be reconsidered, in particular, call by reference is not supported as address spaces are not the same between distributed processes. This leads to the notion of an *interface*. The set of functions that can be invoked by external processes is specified by one or more *interface definitions*.
  - Prog they are not a
  - Prog platform used to imp
  - So lon mpatible), the service implementation can change transparently.
- Choices of IPC mechanism include:
  - Direct IPC are the underlying primitives – relatively low-level (el) of support for communication between processes in a distribut d memory, sockets, multicast communication
  - Remote invocation – based on a two-way exchange between c a distributed system and resulting in the calling of a remote operation, procedure or method, e.g. request-reply protocols, remote procedure calls, remote method invocation
  - Indirect communication:
    - *space uncoupling* – senders do not need to know who they are sending to
    - *time uncoupling* – senders and receivers do not need to exist at the same time
    - for example: group communication, publish-subscribe systems, message queues, tuple spaces, distributed shared memory

Placement

Placement of processes onto machines in the distributed system is a fairly central aspect of the distributed system architecture and design problem. E.g.:

- mappi … ke use of a single pr … ltiple machi …
- cachin … t or whereb …

- mobile code – transferring the code to the location that is m … nt, e.g. running a complex query on the same machine that stores t … er than pulling all data to the machine that initiated the query
- mobile agents – code and data together, e.g. used to install a … ain software on a users computer, the agent continues to check for updates in the background

Modelling processes, machines and process placement

- We can start modelling a distributed system by consider the collection of processes, or let us say n processes in order to *model* it.
- Similarly we can consider the collection of machines that these processes will be distr

- Each proc uld define a ma 's say if $i \in \mathcal{N} = \{1, 2, \ldots, n\}$ is a process, referred to as process i, then let $P_i = j$ be the machine $j \in \mathcal{M} = \{1, 2, \ldots, m\}$ th

- We could simplify the model by assuming that there are a achines as there are processes, $m = n$. Some distribut this way, e.g. peer-to-peer file sharing systems tend to have a peer running on each user's computer. There are no other processes or machines involved.

Process resource requirements and machine capacity I

Lets continue to expand the model. Let L_j be the number of processes that are placed on machine j:

$$L_j = \quad i \quad P_i = j$$

Load can b i ... e resource capacity ... sources; cpu, mem

We can model each processes' resource requirement ... e can also model each machine's capacity. A more detail ... mes harder to understand but it allows us to make more precis ... nts about the distributed system. Let's see where we can get to in this direction.

We could simply say that each machine has an integral capacity of $K \geq 1$, meaning that no more than K processes can be placed on any single

Process resource requirements and machine capacity II

machine. Our machines are said to be *homogeneous* with respect to capacity. Or we could allow our machines to be *heterogeneous* and model the capacity as $C_j \in \mathbb{Z}^{>0}$, i.e. $C_j > 0$ is an integer. The case when $C_j = K$ for all j is the (special) homogeneous case of the heterogeneous model.

If we like we c[illegible]r our model instead of t[illegible] realistic – a given machine's capacity may well be e.g. 5.5 meaning that when executing 5 processes the machine still has spare c[illegible]ot enough to support 6 processes. In engineering analysis [illegible]ation we may want to use real numbers, but for the analysis and o[illegible]ation of distributed algorithms and when analysing a distributed system's architectural properties we may prefer integers.

We could model each of the machine's primary resource capacities – cpu, memory, network and storage – with more detail. If this is desired we

Process resource requirements and machine capacity III

could write $C_{j,\text{resource}}$ to represent the capacity of a specific resource on machine j, e.g. $C_{j,\text{cpu}}$ could be the number of cpus on machine j. Or more succinctly $C_{j,r}$ could refer to the capacity of resource $r \in \mathcal{R}$, which is some set of resou eed to model ma but we can if we wish.

When mo f available upload (data going out of the machine) and download (g into the machine) bitrates. The network is an important res el because all of our IPC will make use of it. There are other cons ations as well with respect to network modelling since it is impac computer network that our machine is connected to, and as well the other machine in the communication. We will specifically revisit our model of the network later.

Process resource requirements and machine capacity IV

On the other hand we have process requirements. It is implicit in the model that each process requires to be placed on a machine. However continui … unt of resource … amount of t … nning With thes … source requirements for resource r on a given machine

$$L_{j,r} = \sum_{i \mid P_i = j}$$

Analysis: load balancing and placement problems I

We can now formulate a number of interesting statements and problems with respect to process placement on machines and resource consumption. E.g. we might want to assert that no machine is allowed to execute more processes than it has capacity for:

In other words we might assert that the placement of processes should never exceed the resource capacity of any machine. We ed formulate the problem of finding a placement of proces ves some objective (subject to the capacity restriction ab ing the maximum consumption of resource r over all machines,

$$\underset{\{P_1,P_2,\ldots,P_n\}}{\arg\min} \; \max_j \{L_{j,r}\},$$

Analysis: load balancing and placement problems II

minimizing the average percentage resource consumption over all machines:

$$\arg\min \quad \max \quad \frac{1}{} \quad \frac{L_{j,r}}{} \quad ,$$

or finding a [illegible] hat are required ([illegible] uirements:

$$\arg\min_{\{P_1, P_2, \ldots, P\}}$$

where in this case $m = \max\{P_i\}$ and again, the placement is subject to the capacity restriction given earlier. The bin packing problem is strongly **NP** complete.

Connections between processes

Interprocess communication is an essential aspect of the Architectural Model. We can expand our model to include the notion of IPC by defining communication as a *directed edge* between two processes $i, i' \in \mathcal{N}$ which we can write as the tuple (i, i'), and then define the set of all connections:

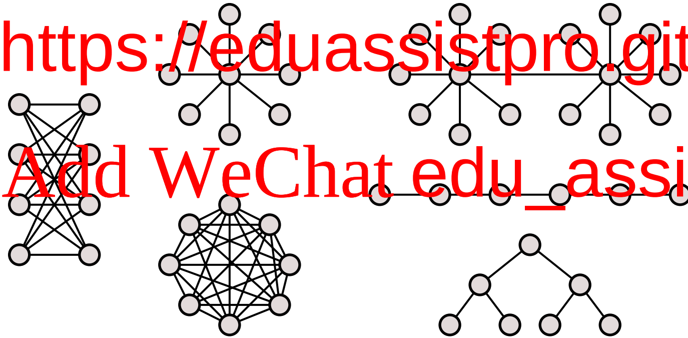
$(i, i'$ i'

To be spec n initiate commun the lifetime of the distributed system (the model is curre fying exactly when or even if it does, just that it could take place). processes $a, b \in \mathcal{N}$ those processes can never dir lifetime of the distributed system then $(a, b) \notin \mathcal{E}_n$ and also $(b, a) \notin \mathcal{E}_n$.

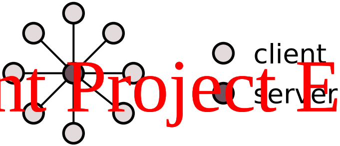
Sometimes we use undirected edges, written $\{i, i'\}$ when its not important to consider which process initiates the communication, or when either process could initiate the communication. With this model, $\{i, i'\} \in \mathcal{E}_n$ is equivalent to $(i, i'), (i', i) \in \mathcal{E}_n$.

Example process connection patterns

The connection "pattern" or *architecture* that arises between processes is an essential aspect to model and understand in a distributed system – we are usually especially interested to understand such patterns as the number of processes in the system grows larger, or to the extreme as $n \to \infty$.



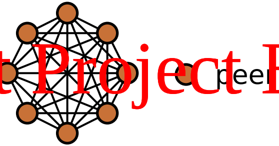
Centralized Pattern



Client pro ... er directly commun ... sign pattern fo

- It is relati
- All data is kept at the server.
- If the server fails or is attacked, the distributed system fails – t of failure.
- Since clients never directly communicate, they are not e the network – privacy/security.
- If some clients fail or are attacked, it does not have an impact on other clients.
- A single server may not be able to support a large number of clients – scalability bottleneck.
- Depending on the geographic/network location of the server to the clients, different clients may see different performance characteristics – latency may vary among clients.

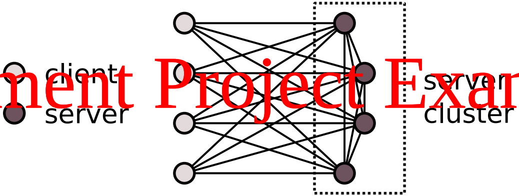
Decentralized Pattern



The decen r-to-peer file sharing sy

- Every pe r process.
- Harder to implement than a centralized pattern – proces clients and servers.
- Data is distributed evenly over the peers.
- The failure of any peer is no worse than the failure of any other e distributed system is very robust to process failure.
- Since peers are exposed to each other over the network their IP address becomes public – privacy/security.
- Resource capacity increases for the system as a whole for each new peer included in the system – scalability.

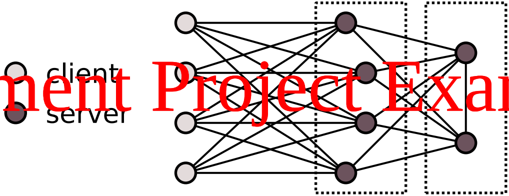
Multi-server Pattern



When a sin connecting to it, the mul pacity.

- Each client can communicate with any of the server to obta ients never communicate directly with each other, similar to t attern.
- Considering only the multi-servers, they share many ch decentralized pattern: every server is like a peer to other ser communicate directly with each other, harder to imple server, data is distributed over the servers, the failure of a single server does not lead to system failure.
- The multi-server architecture exposes more server IPs to the clients – potential security issue.
- Servers can be geographically located at places that provide more uniform access for clients.

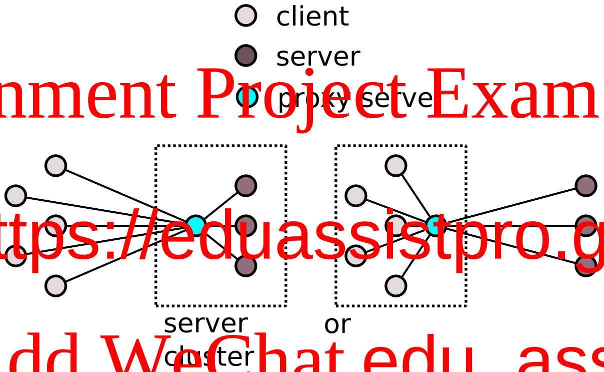
Multi-tier Pattern



The multi
functionality (as opposed to a vertical view given by laye

- The tier furtherest from the clients tends to be the data stor maintains all of the data for the distributed system direct c ion with clients never occurs which increases security.
- The tier closest to the clients typically never communica : his provides isolation and less overheads for synchronization. This tier accesses the data storage tier to obtain data for the clients.
- In a sense, each tier's processes are like client processes for the next tier inwards.
- More complicated than a multi-server and latency grows with the increase in the number of tiers.

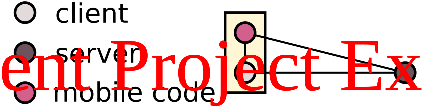
Proxy Pattern



A proxy is a process that sits between a client and a server, re[illegible]g request and responses.

- Clients never communicate directly with the server(s) – extra latency.
- All data flows through the proxy – bottleneck and single point of failure.
- A proxy used at the server side can relay requests to appropriate servers, e.g. a web proxy can relay to web servers and websocket servers.
- A proxy used at the organization side can manage client communication to the servers – security.

Mobile code and mobile agents



- In the mo cuted on the client's shown

  - For m the opposite direction, running client code on the server can avoid load at the client. Both cases avoid some amount of network communication. Mobil e as seen by Javascript in the browser as an example. It leads to "fat cli opposed to thin clients such as the browser without any mobile code). Bef became popular we used Applets as mobile code.

  - A mobile agent is mobile code that also maintains state that is copi gent's code.
  - A mobile agent can move from one server to another, running tasks and collecting data. E.g. if the client requires to undertake a complex query, it may be better to express that as a mobile agent that is executed on the servers that have the data, since undertaking the query on the client may use excessive network resources.
  - Executing code supplied by untrusted parties is a high security risk.
  - Heterogeneous resources are a challenge.

Process communication attributes

Edges in $\mathcal{E}_n$ represent *required* communication between processes. Application requirements on IPC may be described as process communication edge attributes. For each $(i,i') \in \mathcal{E}_n$ let:

- $\omega_{i,i'}$ be the *bitrate* that is required by process i for communication with process i', this may or may not equal $\omega_{i',i}$ and communication requirements are usually a
- $\tau_{i,i'}$ be t
  transm
  which m

- $\rho_{i,i'}$ be the maximum allowable *packet loss*,
  packet of data sent from process i to process
  or may not be equal to $\rho_{i',i}$.

If $r =$ network is one of our resource types, we might wri

$$P_{i,\text{network}} = \sum_{(i,i')\in\mathcal{E}_n} \omega_{i,i'} + \sum_{(i',i)\in\mathcal{E}_n} \omega_{i',i}$$

Machine communication attributes

Similarly, we can consider edges in say $\mathcal{E}_m$ representing possible communication between machines, that is that two machines are reachable over the network. In many cases, we model the underlying network by describing the overall characteristics of the communication channel between all pairs of machines. For each $(j, j') \in \mathcal{E}_m$ let:

- $\omega^m_{j,j'}$ be ... machi ... se a duplex ...
- $\tau^m_{j,j'}$ be t ... of data, sent by machine j, to be received by machi ... $'$ ... ot equal $\tau^m_{j',j}$.
- $\rho^m_{j,j'}$ be the *packet loss*, or probability that a packet ... j to machine j' is dropped or lost, which may or may no ... $^m_{j',j}$.

These three parameters are typically the most useful. We can see later how other aspects of the communication channel, such as *jitter* can be modelled as time varying latency – basically the bitrate and latency can change for each packet that is sent, which causes jitter.

Applying graph theory I

The set of edges over processes is effectively a graph, $G = (V, E)$, with vertex set $V = \mathcal{N}$ and edge set $E = \mathcal{E}_n$ and we can therefore apply graph theory to define properties of our distributed system architecture. The topology o roperties of its arch

E.g. the nu

$$d_i^+ = \big|\{a \mid (i, a)$$

and similarly the number of *incoming* edg

$$d_i^- = \big|\{a \mid (a, i) \in \mathcal{E}_n\}\big|.$$

Applying graph theory II

While the sum $d_i^+ + d_i^-$ is sometimes interesting, mostly we want to consider the number of unique processes that each process is connected to and so we are interested in:

which in gr *number of neighbou* i degree of the graph, or the maximum number of neighbouring pr ny process has in the distributed system.

We can go further and write d *as a function of t* , $d(n)$. Now we have a topological function that is very useful for understanding our distributed system at any *scale*, i.e. at any size or value of n. E.g. if $d(n) = n - 1$ then there are some processes that communicate with every other process in the system, whereas e.g. if

Applying graph theory III

$d(n) = \sqrt{n}$, or $d(n) = \log n$, or say $d(n) = 2$ then we have a very different kind of distributed system topology.

Similarly we can examine a range of interest graph theoretic properties, e.g. the ... *rtest path* through ... path, then t ... s between all pairs of processes: $\max_{i,i' \in \mathcal{N}} |$ ... $'$

This kind of analysis provides a means of defining some i ... concepts such as *scalability* that we discuss sh ... explore a rich space of concepts that apply to distribute ... d therefore help us develop distributed systems with well understood (read predictable) behaviour.

Process and machine state

Each process maintains data which is the state of the process, and includes any data stored in the processes address space, let's say $\mathcal{S}_i^p$ is the state of process i:

- application data structures and objects,
- other da resses of other pr ate.

Each mac hine, let's say $\mathcal{S}_j^m$ i machine will include the state of the OS, and all of the perm torage on the machine:

- data stored in files on behalf of processes,
- OS state including the state of processes running on the m twork information specific to the machine.

Combined, $\mathcal{S}_i^p$ and $\mathcal{S}_j^m$ for all i and j represents the total state of the distributed system.

Distributed system overheads

Processes are required to store information concerning the distributed system architecture, that is not information related to the actual application. Some well studied overheads related to the architectural model include:

- If a process i is going to act as a client and communicate with say d_i^+ other processes over the lifetime of the distributed system then presumably the proces ... rocesses. We som ... *ble*. The table co ... n

- If a proce ... ions from other processes over the lifetime of the distributed syste ... ably the process requires to store up to d_i^- connecti ... constant size with respect to n. This is often com ... neighbour table state as above, as a single measure.

- Sometimes we are only interested in the neighbour table induced by considering undirected edges, d_i.
- If a process i can never communicate directly with process i' over the lifetime of the distributed system then presumably for information to be exchanged between process i and process i' that information will need to pass through at least $|\Phi_{i,i'}| - 1$ intermediate processes.

Scalability I

Earlier we loosely defined scalability as the ability for a system to increase its capacity linearly with an increase in resources. This would be trivial to achieve if the distributed system did not have overheads. Our topological model defined earlier allows us to consider properties of the distributed system as a f [illegible] number of [illegible] the distribut [illegible] ocesses and/or th [illegible] If the overheads grow too large, too quickly, then an increase in the number of resources will not lead to a linear increase in the capacity.

- Consider a communication pattern where every proce [illegible] communicate a message with every other process. For [illegible] eans n^2 messages. As we double the size of the distributed syst [illegible] ow the capacity of the network geometrically. This growth of network capacity is not scalable. Peer-to-peer systems avoid this by using communication patterns with a lower complexity, e.g. $n \log n$.

Scalability II

- We defined the neighbour table as the (maximum) amount of state that a proces $_i^+$. Consider $d(n)$ (i.e e distrib ourself how much st comes very large. Ge er process in the distributed system we say that its not scalable. We wo .g. the overhead to be logarithmic to the size of the system, e.g. .
- Similar considerations apply to properties like the diam ral pattern.

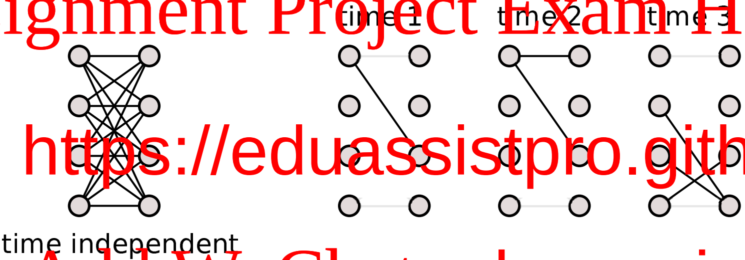
Adding time to the model

The model so far is *static* or *time-independent*. We can allow all of the model parameters to vary with time t, making the model *dynamic* or *time-dependent*:

- $n(t)$, $\mathcal{N}(t)$, – the number of and ids of processes in the distributed system at time t
- $m(t)$, time t
- $P_i(t)$, resource r at time t
- $C_j(t)$, $C_{j,r}(t)$ – the resource r capacity for ma t
- $\mathcal{E}_n(t)$, $\mathcal{E}_m(t)$ – the IPC that occurs at time d network characteristics at time t

Some aspects may remain constant such as the set of all resources $\mathcal{R}$ and other aspects are computed from the parameters, such as resource load $L_{j,r}$.

Time dependent properties



For example $\mathcal{E}_n(t)$ may vary over time as above. We ca
properties of the distributed system with respect to a given point in time, or over a time interval.

Discrete time

It is sometimes very convenient to model time as progressing in *discrete steps*. In this case we assume that t takes values $t_0, t_1, t_2, \ldots$ and that $t_{a+1} - t_a = \Delta$ where Δ is some constant; typically $\Delta = 1$.

When considering discrete time it is usually also assumed that all machines, and therefore all processes, progress *synchronously* in time, as if all of the ma [illegible] s a synchron [illegible] make assumpti [illegible]

- a messa [illegible] e step
- all messages are constant size
- in each time step a machine can send at most [illegible] st k messages (i.e. $2k$ messages in total per time step)

The synchronous model is useful for analysing distrib [illegible] s in a formal sense, where the complexity of the distributed algorithm (number of time steps to complete) is expressed in terms of communication complexity (required information that must be communicated as part of the algorithm).

Let's assume that each process can send 1 message and receive up to 2 messages in each round. Send and receive times are synchronized.

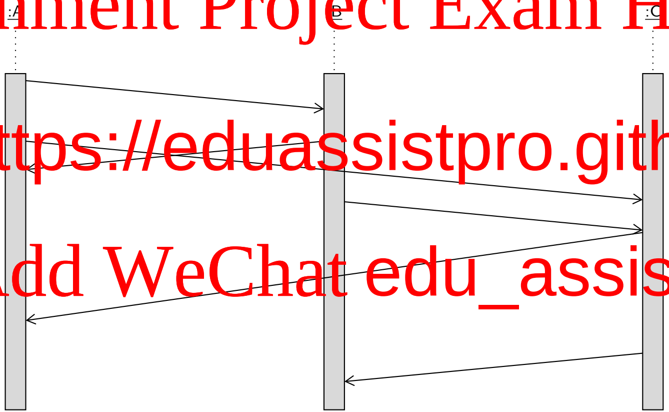
Continuous time

We may wish to consider time progressing *continuously* or *asynchronously* where t is a real number. We may still consider points in time $t_0, t_1, t_2, \ldots$ however a … using an asynchro…

- a messa…
- the time t … e sending machine to transmit the message, which is a function of th … message and the available outgoing bitrate of the machi … or the receiving machine to receive the message, which ever is th…

- only one message can be being transmitted or received at a … e

Processes send messages as needed and messages take some time to be received. Send and receive times are not synchronized.



Defining failure

All distributed systems are subject to a real-world phenomenon called *failure* which arises from:

- *faults* – unintended and usually unknown defects in hardware manufacturing and/or software programming,
- *exceed* ... itions, such as m ... apacity such as ru ... eds the range specifi ... beyond its speci

A *Failure Model* states the types of failure that are b ... what types of failure will the distributed system be subje ... therefore exclude all remaining types of failure. Perha ... g types of failure are assumed to be handled by middleware or the OS/hardware, or else they are ignored and the system's behaviour is undefined when such failures happen.

Types of failure I

Failure manifests in a number of ways, listed here in a loose order of *severity*:

- *error indicators* – error codes returned from methods including returning `null` values, cases where a method call has failed, and error messages returned from other processes to indicate some kind of failure
  - can ar ... ine resou ... rrupted data read fr ... file that is not a
  - proc

  - if the process continues obliviously to the error condition then further, more severe failure may arise – the process is now in an *undefined sta*
- *exceptions* – may terminate threads if they are not hand ... e the process if the main thread terminates

  - if a method does not return error codes and the expected operatio ... d can not be achieved then an exception may be raised
  - users may interrupt the process (user is signalling an error condition)
  - out of memory, out of disk space, division by zero – the execution of code can't continue, returning is not an option
  - threads can catch exceptions and handle them, if not the thread may be terminated and this may terminate the process – if a thread is terminated or "dies" without being handled then the process may continue to fail

Types of failure II

- *process termination* – OS or user terminates the process
  - the process may be attempting to access memory or other machine resources in an invalid way and/or in way that is potentially harmful to the OS and the machine
  - the user wants to immediately prevent the process from executing
  - the OS may be trying to avoid OS failure, e.g. when swap space is exhausted then the OS may terminate the process that is using the largest amount of memory
  - proc ates with
- *OS/ha*
  - OS fai achine may be required
  - storage devices fail entirely – none of the data is available
  - network devices fail entirely – the network is not usable
  - the machine fails – may or may not be able to be restarted, machine s ding permanent storage may or may not be available
- *power supply failure* – many machines connecte may fail at the same time
  - this is an example of correlated failure
  - individual machines may or may not loose permanent storage
- *network failure* – computer network equipment can fail as well
  - packets are dropped – largely due to queueing becoming full under high load,

Types of failure III

- physi /or
  brea
- netw
- netw ot
  communicate with machines on another network

Failure detection

- Failure may be reported to the process with error codes and exceptions. However not all types of failure are reported, many types of failure need to be *detected*.
- *Silent failure* – there is no failure reporting mechanism such as error codes or exception, processes must explicitly detect such failure:
  - dropped packets – the network does not report that a packet was dropped
  - proc ... process enter ... hread dies e
  - distri
  - OS fai ... nsive
  - device failure – can cause the OS to block indefinitely, which in turn can cause processes to block indefinitely

- *Failure of a remote process* in a distributed syste ... reported to other processes in the distributed system – it m ... cted
  - unless there is another distributed system process on the same ... ote process, it is technically impossible to be able to tell the difference between some kinds of network failure and process failure – no response to communication requests could be either
  - sometimes communication requests are explicitly “refused” by the remote machine, which is indicative of remote process failure
  - if a remote process cannot be communicated with for some time, the distributed system has no alternative but to label that as failure – being unresponsive is considered failure

Failure handling I

When a failure has been detected then it should be handled. If distributed system can continue operating correctly and at full capacity under the presence of failures we say that it is *fault tolerant*. This should not be confused with the notion of "tolerating" faults as discussed below.

- *fail-stop* – the failure (termination or otherwise) of a process can be detected certain
  - Failu out that failur failed, when it has n
  - With by the remaining processes to have failed (e.g. if the process is unresponsive), the process is deliberately terminated to avoid any further potential for erro will consider that processes terminated, and even if it "wakes up" la sponding, it will be told that it should terminate since it is no longer considered distributed system.

- *graceful failure* – the distributed system can contin degraded performance, in the presence of faults. A distributed system with enough reliability can continue to operate even though some machines may have failed, but the system may not provide the same response times that it did before the failure. Another example is that packet loss may cause only a minor degradation in video stream quality.

Failure handling II

- *tolerating faults* – in this case the users of the system are required to tolerate a certain amount of failure: “can't provide service, please try again later”. Most users of commercial operating systems tolerate failure in the OS: they restart their computer. They don't take it back to the store and say it crashed and doesn' f video stream deo quality when fa

- *failure* retrying the operation transparently. This is a transparency challenge. Some failures cannot be masked since e.g. if the network is down t aming video can't be delivered and the user will notice this (we coul pre-cached commercials in the hope that the network wi oon to distract the user from noticing the fault).

- *failure recovery* – in some cases, failure needs to be recovered from, because the state of the distributed system has been affected by the failure, e.g. data has been lost, or else the state of the system is no longer valid.

Failure handling III

- If the system has redundancy, e.g. using multiple copies or erasure codes to store state, or having multiple processes on independent machines that are undertaking the same com ... ncy, but as well, r ... ailure redu ... em to be at full op
- Chec ... egular basis is a tech ... ocess fails, then the process can be restarted using the last check-point, which m ... rocess having to redo some calculations again in order to "catch up" wit ... es in the distributed system. Distributed check-pointing is a well stud ... uted systems. Complex check-pointing will check point state at diff ... rocess state and machine storage state. In case of machine storage fail ... p redundant copies over multiple machines.

Modelling and analysing failure I

The exact time and type of the next failure to occur in a specific system is difficult to accurately predict, for if it were easy to predict then engineers would likely already have removed its possibility. Therefore it is widely accepted to model failure by estimating the *mean time to failure* (MTTF) which give

The MTTF is a measure of *reliability*. High reliability.

We usually assume that the probability of the system fai he next t seconds is given by the cumulative distribution function of the Exponential distribution:

$$\mathbb{P}[\text{failure} < t] = 1 - e^{-\lambda t}$$

Modelling and analysing failure II

The Exponential distribution, $f(t;\lambda) = \lambda\, e^{-\lambda\, t}$, assumes that we have no information about when the next failure will occur.

If we are using a discrete-time model, we may want to estimate the number of f

equivale

$$\mathbb{P}[\text{failures} = x; \lambda, \Delta] = \frac{\quad}{\quad}$$

Consider n processes, where each process can fail in dom according to our model above. The probability that no esses fail within the next t seconds is:

$$\left(1 - \mathbb{P}[\text{failure} < t]\right)^n = \left(e^{-\lambda\, t}\right)^n = e^{-n\,\lambda\, t}$$

Modelling and analysing failure III

The probability that at least one of the processes fails within the next t seconds is therefore:

$$1 - e^{-n\lambda t}$$

For example, if MTTF for each process is 1 week, i.e. each process fails independ [...] n, then for a distribut [...] t least one of the pr [...]

$$1 - e^{-\frac{10}{7}} \approx 0$$

If we had 100 processes in our distributed system then th[...]ity of at least one process failing in the first day becomes practically certain. Put another way, with 100 processes, the effective failure rate is now 100λ which gives an effective MTTF of $\frac{7}{100}$ days or 1.68 hours: the system will fail on average after 1.68 hours or about 100 minutes of operation.

Modelling and analysing failure IV

As the num s chance of failing inc to invest in m to operate for sufficient periods of time to be useful.

Availability Model

While we are talking in terms of probability, if we let $X = 1$ if the distributed system is *available for use* when we try to use it and $X = 0$ otherwise, and we model X as a random variable, then the *availability* can be defined as:

i.e. the prob hen we try to use it. If it ailable, and if the availability is 1 then it is always available. The avai lso be seen as the fraction of total time for which the system is av le, e.g. 6.9 days of every 7 days the system is available and 0.1 days it t. Thus $\mathbb{P}[X = 1] \Rightarrow 6.9/7 \approx 0.99$.

Of course, this is just a model. We may know that the system is down regularly on Mondays at 1am, while our model assumes that downtime could happen at any time. We can always refine the model to take additional knowledge into account.

Goals, threats and mechanisms

- Security of a distributed system is achieved by securing processes, communication channels and protecting objects they encapsulate against unauthorized access:
  - *Access rights* specify who is allowed to perform operations on data in the distributed system.
  - Each o *incipal* – the user o
- Possibl
  - Thre f the message. Sour
  - Threats to communication channels: Enemy can copy, alter or inject messages.
  - Denial of service attacks: overloading the server or otherwise t delays to the service.
  - Mobile code: performs operations that corrupt the server or ser y way.
- Addressing security threats:
  - Cryptography and shared secrets: encryption is the process of scrambling messages.
  - Authentication: providing identities of users.
  - Secure Channel: Encryption and authentication are used to build secure channels as a service layer on top of an existing communication channel. A secure channel is a communication channel connecting a pair of processes on behalf of its principles.