Chapter 11
Transparency

Complexity

Distributed systems consist of many interacting components.

Given the connectivity and even the existence of many components may vary during operation. The system is complex and dynamic.

They are complex to operate
Mitigation:
transparency, common services, middleware

Transparency means that the complexity is hidden from the user.

Common services: provide utilities which provide services to many applications: leads to standardization and reduces requirements for a particular application

Middleware: software which provides a common base between heterogeneous systems and facilitate interoperability

3 T*ransparency*

Access to the services of the system without knowledge of the implementation.

Definition (R.J. Anthony)
“single system abstration presented to the user”

Interface hides the complexity of the underlying distributed processes and resources so it runs like a single entity.

Problems during running should be hidden from the user and the system should be self repairing.

Many facets:
Access
Location
Replication
Concurrency
Migration
Failure
Scaling
Distribution
Implementation

<u>Types of transparency</u>

Large distributed systems must allow access to resources in a transparent way. Otherwise interactions with the system become complex.

www.google.de
Will take you through to a machine which is “close” and not too heavily loaded. Imagine needing to specify not only what address you wanted to access, but the route for the communications to pass along.
Internet name resolution is a good example of a multi level distributed system whose complexity is entirely hidden from the user.

Check for instance DNS on wikipedia

How ?

Access transparency
Same api for local or remote access

Location transparency
No knowledge of location

Replication transparency
Multiple copies – all kept consistent and referred to as a thing, rather than an instance

Concurrency transparency
Concurrent processes share without interference, while making no special arrangements.

In some cases I think the use of the word transparency is rather forced. Although all the concepts are valid

Types of transparency

Migration transparency
Movement of process occurs with no user intervention, yet completing as if unmoved

Failure transparency
Processes complete even if part of the system fails. Probably need migration transparency.
e.g. internet – self healing – autonomic

Scaling transparency
More resources give more performance. No other change needed

Performance transparency
Graceful degradation (as load increases – or resources degrade)

Distribution transparency
Existence of the network is hidden. No requirement to know network addresses or communication protocols

Implementation transparency
Ability to mix components in different languages – importance of interface definition.

Not all applications have (or need) all these facilities

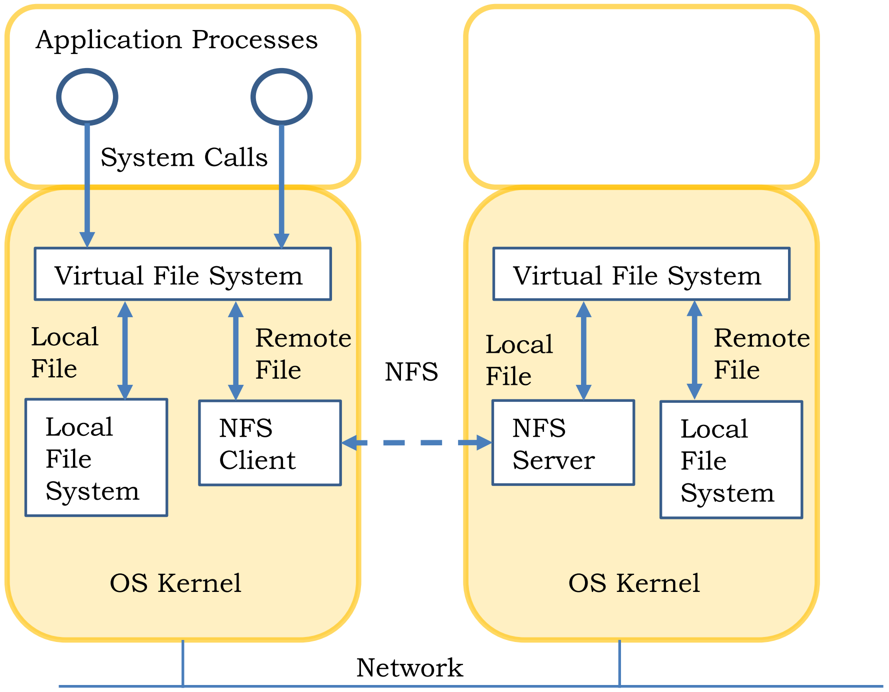
Access transparency

Same api for local or remote access

The user's interface to access a particular resource should be the same no matter where that resource resides. (Local or remote).

May be achieved via software layer: deals with access resolution; requests which can be satisfied locally to the local resource; requests which need remote access to a corresponding layer on another computer. Referred to as *resource virtualisation.*

Unix **V**irtual **F**ile **S**ystem (VFS)



Resource Virtualisation

For instance printing to a colour printer which is a virtual resource and the "system" sends the output to the nearest physical resource. Means that a user in an office in Stuttgart can print to the "same" printer as when they are in London

Location transparency

Resources and services. Distributed systems increase complexity:
Replicate resources; distribute over multiple locations; dynamically reconfigure (varying workloads or partial failure)

Access without knowledge of location.
Resource naming scheme – no location.
System maps the name to a unique identifier which can be mapped onto current location.

Mapping from name to physical object can be used to provide resilience and speed.
Request for much disk space – don't access a disk server being maintained – don't access a server whose network is congested.

Domain Name System
DNS

Mapping from name to physical realisation is called "name resolution"

Again resource virtualisation is an enabler.

Remote procedure call (RPC) and Remote message invocation (RMI) are not location independent, an address is needed; a service which provides an address for such a call can create location transparency. Object Request Broker (ORB) is an example of such a system.

Location (i)

Examples:

Accessing google.de will take you to an ip address. The address is a front end to a farm of machines and your request is routed to the least heavily used server.

JQuery: dynamic websites often use javascript libraries. You may host them on the same server as the web pages. You may also point them to JQuery libraries on dedicated servers; this reduces the load on your servers.

However JQuery is hosted on a CDN – a content delivery network. This means when a user accesses a page using JQuery if downloads it from a third party web site. But the site it actually accesses depends where the user is (and not where the page is hosted). The library points to a CDN and the actual machine used to deliver the content is transparent to both the developer and the user.

(An example of the sort of services which aid building distributed systems)

Cloud services: web hosting which can dynamically expand to deal with peaks in demand.

Replication

Prevent performance bottlenecks.

Resilience against failure of resource or network link.

Provide high speed links to processing sites.

Static data sources

Requires multiple copies of objects can be created without any effect of the replication being seen by the applications.
Not possible to see how many replicas exist or see the identity of a specific replica.

The system must be able to distinguish and this creates naming problems.
When we look at data storage I will describe **a** way to solve this problem

Also means that all replicas must remain in step and always return the same answer to a question at the "same" time.

This often means that access to a resource is via a service which ensures the enforcement of these conditions.

Replication

Databases more complicated – need to ensure consistency, in the presence of multiple copies and multiple simultaneous users, in the presence of failures – a failure of a bank database may involve many millions of pounds.

Starting point for replication transparency is unsurprisingly database architectures … a course by itself

Possible architectures
Primary – Backup
Master – Slave … may be many slaves
Peer to Peer

Consistency:
Consistent … hard to do in a distributed system … ensuring consistency can become a bottleneck

New concept
Eventually consistent … easier where possible

The contents of your facebook page are held in a database. Consider posts from a George, which you and Jerome are interested in. You should both see the same posts eventually, but if you are in Germany while George and Jerome are in New Zealand, if Jerome sees Georges posts 5 minutes before they are available to you, this does not cause a problem

Concurrency

How can more than 1 process access and update data without leading to inconsistent results.

Share objects, data, resources, without creating problems.

Single access can be achieved by queuing requests to a resource.

Joint access is a complex problem and is much studied in databases.
The basic idea is of a transaction: a complex series of events which can be seen as “atomic” as far as the database is concerned. Either it all occurs or none of it occurs.

The idea of taking complex actions and providing mechanisms to ensure that all or none of them happen is of great utility in distributed systems.

A user may operate in the belief they are the only users of the system.

Concurrency

Transactions

Atomicity – the system sees the steps which make up a transaction as a single operation which cannot be divided

Consistency the stored data must be in a consistent legal state at the end of all transactions … made difficult by replication

Isolation all transactions must complete as if there was no other transaction occurring at the same time. May need locking, mutex, etc. Handled by the system not the user

Durability results of a transaction should be permanent, in the presence of hardware and or software failures.

Migration

Move processes and services from one physical machine to another without the user being aware of any break in the continuity of service.

Many virtual machine providers eg VMware – allow a virtual machine to be migrated from one physical server to another without any break in service.

Condor a job distribution service which goes back to the early 1990's.
A master distributed “batch jobs” to empty computers, (including personal desktop machines, for instance over night). If the “owner” of the machine started to use it, the job would be moved from the machine to another empty one, or suspended until a machine was free.

Failure

A user is unaware of the failure of part of the system.

Your computer keeps time using a *network time server.* If it is removed from the network – network failure, machine failure, machine maintenance, … an alternate server will be contacted.

Duplication – stateless service.

If a central or critical component failed and there are a number of possible replacements, then a replacement must be chosen.

Need to make sure only one replacement is chosen, and that one is chosen.
How …..
Co-ordinator, but this becomes a single point of failure.

Election of a new machine to provide the critical service (may be the coordinator).

Scaling

As the number of resources increases we hope to see an improvement in performance … either time for a single job or the complexity of a job which runs in a given time. (Amdahl or Gustaferson).

All systems will start to show some level of slow down after a particular point is passed, and that can be very severe.

Centralisation means a single machine may become a bottleneck.
Distribution can mean inter machine communication becomes a limiting factor.

There is normally less bandwidth then computational power.
Keeping interactions to a minimum is a good way to improve scalability.
pp computing: communication to start the job with perhaps a small number of parameters; communication at the end to return results; **nothing during execution**.
Weather forecasting: same pattern.

Perfect scaling means performance is proportional to resources. Very rarely achieved, but the ideal

Distribution

All network and location information is hidden from the user and the applications run as if the were all local.

Aim of grid computing was said to be “the death of distance”

Implementation

Heterogeneous machines and operating systems respond to a common API

The aim of middleware

Implementation

Automatic means of locating services and resources

Clock synchronisation

Coordinator process elections

Distributed transaction processes

Access to data sets (and replicas)

Communication support for components which form a group

Support for indirect and loose coupling to support scalability

By making these centralised services the underlying system can be altered and the user notices no difference to the operation of the application

Mechanism

This is a layer which runs between the underlying hardware and OS and even implementation language and means that a user defined application can be written in a way which allows communication with system services and other applications independent of the hardware, OS and language in which they are written.

CORBA: Common Object Request Broker Architecture

XML: eXtensible Markup Language

JSON: Javascript Object Notation

SOAP: Simple Object Access Protocol

The common services also may be considered as part of the middleware.

I have referred to some concepts under more than one heading.

This is because you will see them referred to in more than one way.

It is also because it is useful to think of them in different ways depending on context.