Chapter 1

Introduction

Paul Kyberd

Paul.Kyberd@brunel.ac.uk

2 days

Friday Start 09:00
3.5 hours in morning
1 hour for lunch
3.5 hours in afternoon

Saturday Start 08:00
End at around 13:00 – and then lunch

Will try to make sessions them equal length

3 sessions: 70 mins
2 breaks: 15 mins

ASK questions

www.brunel.ac.uk/~eestppk/EE5616

Software Engineering

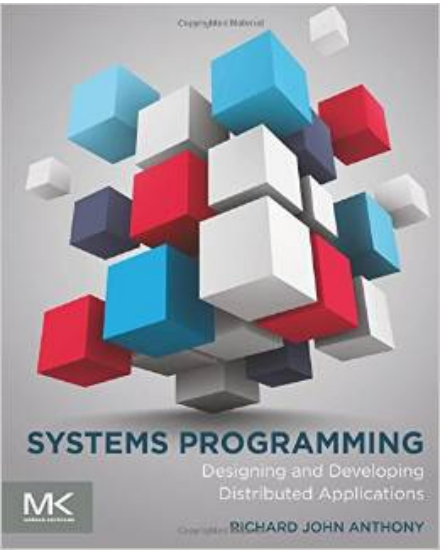
Traditional Engineering:
- How to build a bridge that will not fall down.
- Will carry the traffic
- Will not bankrupt the supplier or purchaser

Software Engineering
- How to build programmes that will not fail
- Will run efficiently – time and resourses

For modern systems that means how to build distributed systems

Multiple cores on a CPU; multiple processors in a farm or cloud.

Cloud – eg Amazon EC2



Similar to the approach of a new(ish) book (Published March 2015) … aims to give you the tools to develop distributed applications. It should be thought of as providing extra ideas and context.

I will give you an introduction to the subject by using the book and my experience.

If you are interested you can develop your skills using the book

Content

Testing – all

Principles behind design of distributed systems.

How to go about designing distributed systems.

Take examples from large distributed systems

Large scale computing (20th century) depended on *supercomputers* or large numbers of processors (often special purpose) connected by a custom network.

Large scale computing (21st century) so far depends on large numbers of commodity computers;
standard IP networks;
Standard protocols.

Workflow important, but not covered

Typically 100s to 100,000 jobs
Automatic monitoring and control vital

Compared with Prof Khan's lectures more about the techniques to realise the design

Definition

None agreed

A distributed system is an information-processing system that contains a number of independent computers that cooperate with one another over a communications network in order to achieve a specific objective.

You are using a distributed system when a computer you have never heard of, in a place you don't know can cause a programme you are running to fail.

Communication network allows coordination via message exchange – to achieve a common goal.

Visible: machines on a communication network

Structural: a set of cooperating processes

Why the move to distributed computing?

Better price-performance

Better total performance

Redundancy – can be achieved

Opportunities for parallel execution

Improved scalability – just add components

Physical Limits

Disadvantages

Increased complexity

Networking issues – failure and inconsistent operation

Security

Problem

Disadvantages

Solving a problem on a single core means creating a system which mimics the way that you might set about solving it.

Distributed implies:

- Problem division into bits performed simultaneously
- Coordinating between separate sections

Coordination on a single system can be done by local messaging and use of the system clock.
A failure will typically be obvious and system will cease.

Coordination between spatially separated computers needs a “clock”.
If failure of a single part means failure of the whole distributed systems will become much less reliable.
The computation must complete in the presence of (a reasonable amount of) failure.

Synchronous & Asynchronous

A *synchronous* system is under the control of a central clock
Synchronous communication is when both parties (processes) in an exchange wait for an answer before proceeding

An *asynchronous* system has independent local clocks
Asynchronous communication is when processes dispatch a message and perform other tasks. The reply will typically interrupt them.

Asynchronous systems typically perform *(much)* better.

Failures in a synchronous system are easy to identify.

Failures in an asynchronous system are *impossible* to identify.

Awareness of failure, but not obsession.

We shall mostly consider asynchronous systems, but we will (implicitly) assume failure rates are low.

Pro

Scalability
Reliability
High performance
Geographic distribution

Con

Complexity – difficulty of testing
Requirement for replication - consistency
Dynamic changes
Finding distributed resources
Shared resources; control and consistency

Architectural Metrics

Number of components

Number of instances of each component

Cardinality of connections

Dynamic or static connections

transparency

Types of transparency

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 ?

More details on this later

Access transparency
Same api for local or remote access

Location transparency
No knowledge of location

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

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.

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

How to organise the resources

Network and distribution

Complexity

Layered Architectures

Hierarchical Architectures

Heterogeneity

Stateful and Stateless systems

I will look at the things which I consider most important/useful.

Will refer forward on a couple of occasions to EE5531

I will keep them separate but will use them to support each other and I hope give you a better range of tools to build economical, high performance systems

And pass the exam