COMP28112 Lecture 2

Challenges/goals with distributed systems:

- Heterogeneity
- Openness
- Security
- **<u>Scalability</u>**
- Failure Handling
- Concurrency
- Transparency

- **A few words about parallel computing**

“Latency is not zero”

Two options are made available to a client to retrieve some information from a remote server:

- The first option allows the client to retrieve a 100MB file (containing all the relevant information) at once.
- The second option requires the client to make 10 remote requests to the server. Each remote request will return a file of 1MB (all 10 files are needed by the client to retrieve the relevant information).

If the latency of the network link between the client and the server is 1sec and the available bandwidth is 100MB/sec which option would be faster? What if the latency is 1msec? What do you conclude?

Why would you need 100 (or 1000 for that matter) networked computers?

What would you do?

Increased Performance

- There are many applications (especially in science) where good response time is needed.
- How can we increase execution time?
  - By using more computers to work in parallel
- This is not as trivial as it sounds and there are limits (we’ll come back to this)
  - Parallel computing
- Examples:
  - Weather prediction; Long-running simulations; …

How do we speed up things?

Many scientific operations are inherently parallel:

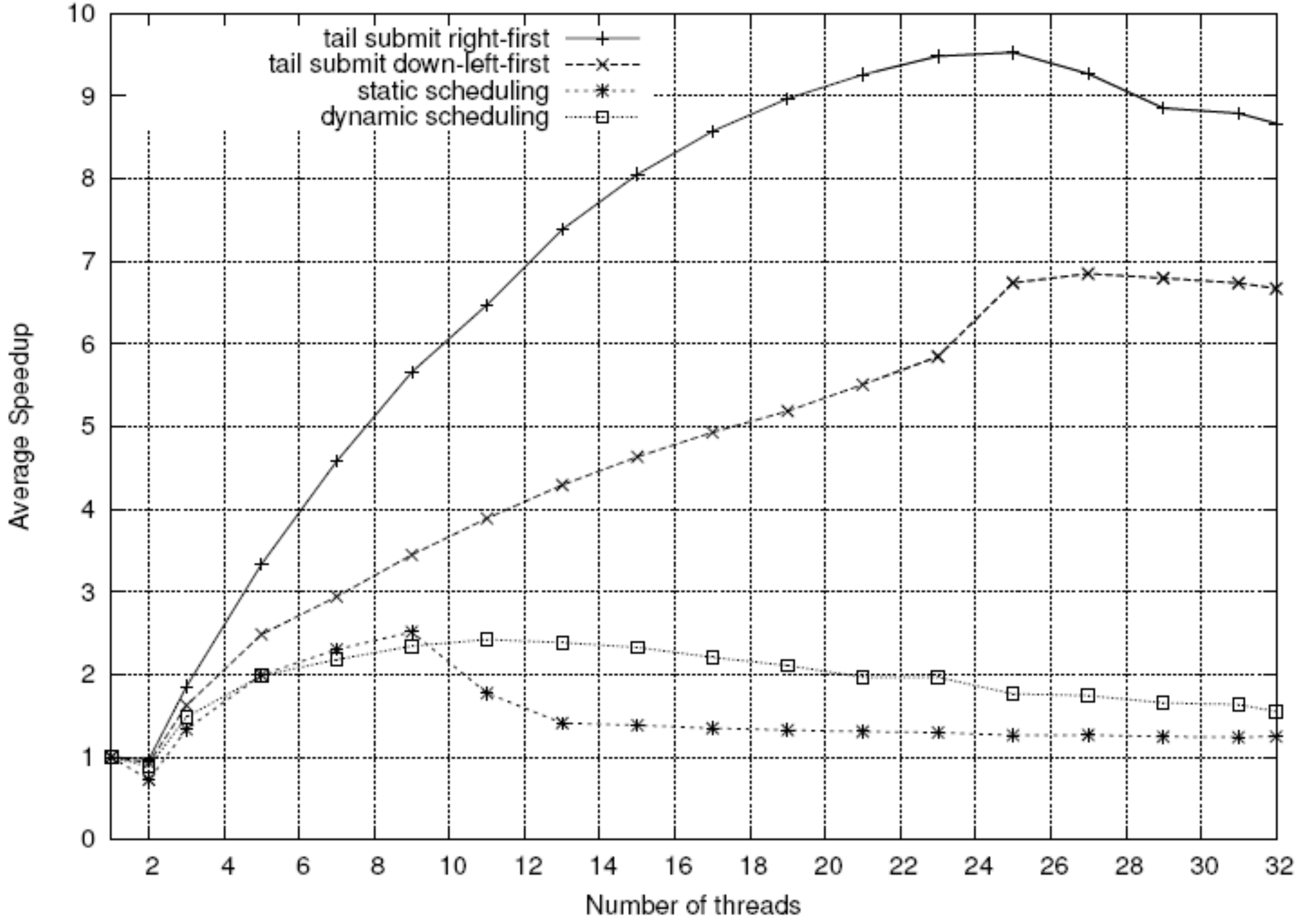
```
for(i=1;i<100;i++)
    A[i]=B[i]+C[i]

for(i=1;i<100;i++)
   for(j=1;j<100;j++)
     A[i,j]=B[i,j]+C[i,j]
```

Different machines can operate on different data.

Ideally, this may speedup execution by a factor equal to the number of processors

(speedup = ts / tp = sequential_time / parallel_time)

Mesa et al, "Scalability of Macroblock-level Parallelism for H.264 Decoding"



Parallel Computing vs Distributed Computing

- Parallel Computing:
  - Multiple CPUs in the same computer
- Distributed Computing:
  - Networked Computers connected by a network.

- But don’t use the distinction above as a universal rule: in practice, it is not clear where exactly we draw the line between the two.
- Some of the problems are common, but with a different flavour…

An example: IBM Deep Blue

- A 30 node parallel computer.
- In May 1997 the machine won a 6-game match of chess against the then world champion.
- The machine could evaluate about 200 million positions per second or up to 40 turns in advance.

- In parallel computing, we need to deal with:
  - **Communication** (parallel computers may need to send data to each other)
  - **Synchronisation** (think about the problems with process synchronisation from COMP25111: Operating Systems).

Not all applications can be parallelised!

```
x=3                    x=3
y=4                    y=x+4
z=180*224              z=y+x
w=x+y+z                w=x+y+z
```

What can you parallelise?

If the proportion of an application that you can parallelise is x ($0<x<1$), then the maximum speed up you expect from parallel execution is: $1/(1-x)$

The running time, *tp*, of a programme running on *p* CPUs, which when running on one machine runs in time *ts*, will be equal to: $tp=to+ts*(1-x+x/p)$, where *to* is the overhead added due to parallelisation (e.g., communication, synchronisation, etc).

(Read about **Amdahl's law**)

Example

x=0.2; Then, *tp*=*to*+*ts**(0.8+0.2/p)

Assume that *to*=1, *ts*=1000, then:*tp*=801+200/*p*

p=1, *tp*=1001; *p*=2, *tp*=901; *p*=4, *tp*=851 …

In practice, *to* is a function of the number of processors, *p*. Assume that *tp*=*to***p*+*ts**(0.8+0.2/*p*):

Then, *tp*=800+*p*+200/*p*:

p=1, *tp*=1001; *p*=2, *tp*=902; *p*=4, *tp*=854;

p=10, *tp*=830; *p*=100, *tp*=854; *p*=200, *tp*=1001, …

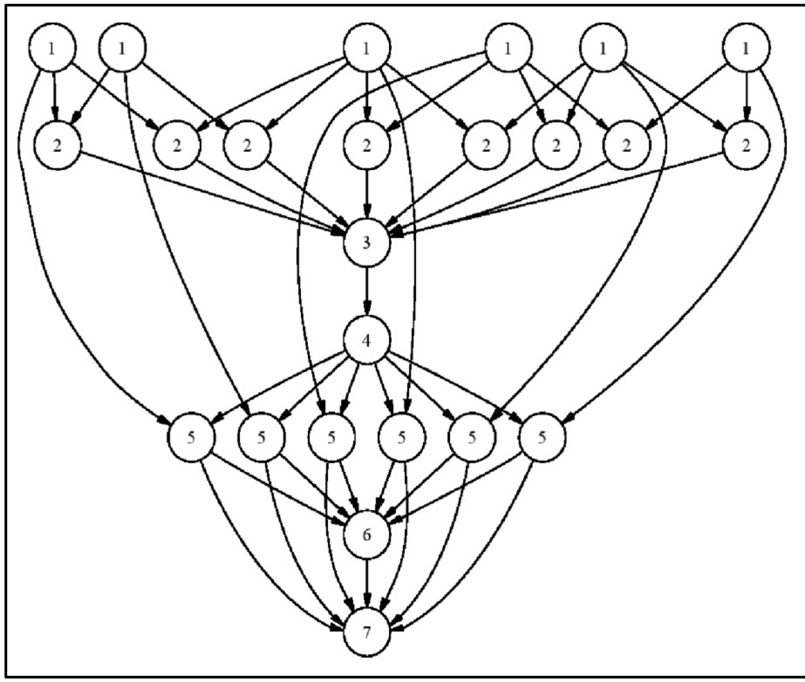
Further reading: http://www.sciencedirect.com/science/article/pii/S0743731513002177 ("Amdahl, the story of a law that gets broken all the time" – "The total amount of work divided by the length of the critical path is a much better way to get an upper bound on useful parallelism for a given task")

How to parallelise an application

1. Try to identify instructions that can be executed in parallel. These instructions should be independent of each other (the result of one should not depend on the other)
2. Try to identify instructions that are executed on multiple data; different machines operate on different data.

(Loops are the biggest source of parallelism)

Dependencies can be schematically represented – see graph!



Summary...

- Several challenges / goals:
  - Heterogeneity, openness, security, scalability, failure handling, concurrency, transparency.
- Parallel computing: not really a topic for this module; some of the problems are common, but the priorities are different.
- Reading:
  - Coulouris4, Coulouris5, skim through Chapter 1.
  - Tanenbaum, Section 1.2 (all of Chapter 1 covered)
  - Read about Amdahl's law!
- Next time: Architectures and Models.