COMP28112 – Lecture 14

The Quest for Performance (lab exercise 3)

Performance Issues

- It's not a good idea to build a distributed system and then realise that it fails to meet demand!
  - Trial and error is not a good idea!
- One issue with distributed systems is that some of the solutions trade performance for something else: for example, XML might be good for interoperability, however, it increases the size of communication messages!

- **Performance Modelling**:
  - A model is a representation of a system that captures the most relevant characteristics of the system under study.
- The model is used to analyse the behaviour of a system under different circumstances

What is a performance model? Why?

- An abstraction of the reality that allows us to study the performance-related characteristics of the system.
  - The model is not the reality: this introduces some errors!
- How many servers to cope with demand?
- For a specific type of workload, what is the most effective scheme for load balancing?
- **Performance tuning** is a huge territory. What are the sources of performance problems? (bottlenecks)

Performance models are used to answer what-if questions as opposed to direct changes in the production environment!

Performance Modelling Approaches

Two key approaches:

- **Analytical solution**:
  - Derive formulas: Some times this is simple, but most of the times it is not!
  - There is a branch of mathematics dealing with the properties of systems with queues: it requires more advanced Maths than the Maths we teach here (unfortunately ☹)
  - Queuing theory (http://en.wikipedia.org/wiki/Queueing_theory) could have helped with lab exercise 3!
- **Simulation**:
  - Monte-Carlo Simulations (random numbers are used)
  - Discrete-Event Simulation (& events over time are considered)

Analytical Models: examples

- Remember Amdahl's law from Lecture 2?
  - This is a basic model to estimate the running time, *tp*, of a programme running on *p* CPUs, which when running on one machine runs in time *ts*: *tp=to+ts*(1-x+x/p)*
- Minimum Possible http transaction time:

  $t = RTT + t_{request} + t_{siteprocessing} + t_{reply}$, where RTT is round-trip-time, $t_{request}$ is *requestsize/bandwidth*, $t_{siteprocessing}$ is the time for processing and t_{reply} is *replysize/bandwidth*. (Heidemann *et al*, Transactions on Networking,1998)

Some problems are simple

We have 1000 transactions to process.

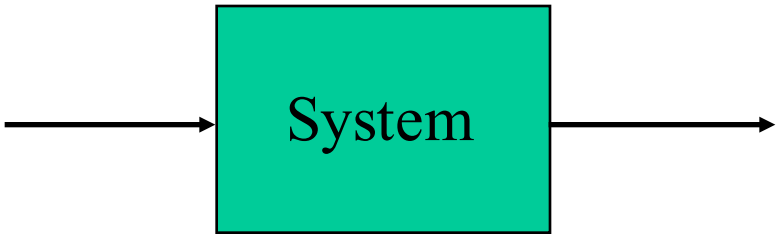
We have 3 servers:

- Atlas can process 12 transactions per second
- Bamboo can process 29 transactions per second
- Caesar can process 59 transactions per second

- How shall we allocate the transactions for load balancing?
- What if each transaction takes a different amount of time?

Queueing Theory

- Based on applied probability theory
- It studies the behaviour of systems with service counters to which customers arrive and join a waiting line (queue). Parameters of the problem:
  - Arrival rate
  - Number of servers
  - Processing time
  - …



- Key question: what is the average waiting time?
- Interested in steady-state solutions.

A useful result: Little's Law

- In the steady-state:
  - $Avg\_no\_of\_customers\_in = arrival\_rate \times service\_time$

- A web server was monitored for 30 minutes and the number of requests submitted was found to be 9000. The average number of active requests in the system was 7. What was the average service time for each request?
- In the Duchy of Grand Fenwick, every year, 100 trainee dentists pass their qualification exam and start practicing the profession. If a dentist works for 20 years on average, how many dentists will be working in the Duchy in 2050?

Simulation

- Monte-Carlo methods:
  - Calculate results based on repeated random sampling.
  1. Generate random input
  2. Simulate what follows and check the result

- Useful for computing a result after a large number of samples.
- Example: verify the winning strategy in the Monty Hall problem.

Discrete Event Simulation

- Events are randomly generated.
- The simulation proceeds as a chronological sequence of events.
- Implies some notion of time in the simulation

- Many computer games are discrete event simulations: e.g., football manager

An example of discrete event simulation

```
// football manager, main loop
Time=0;
Set_initial_conditions;
Do {
  time++;
  retrieve_events_for_time; //eg, game against X
  process_event_for_time; //eg, pick team, play game
  update_data; //eg, update league table
  generate_new_events; // player injured until time t
} while simulation_is_on;
```

Lab exercise 3

- In this system, requests keep arriving from the clients; these requests involve some computation at the server (we don't care what is computed). The average time between the arrival of two successive requests is an input parameter (arrival rate, expressed in time units, which is the unit of time used in the simulation – see below).
- When these requests arrive, they are initially stored in a queue. The maximum queue size is an input parameter. Any requests that arrive when the queue is full are rejected.
- A number of servers (or CPUs, if you like) may fetch requests from the queue for execution. Only one server can fetch requests at a time. Clearly, a single server cannot execute multiple requests at the same time.
- The time needed by the server to complete every request is also an input parameter (expressed again in time units).

Taking as an input the four parameters described above, you are going to write a program to simulate the system and show, over a period of 1,000,000 time units:

- The percentage of requests that was rejected.
- The average queue size (that is, how many requests were in the queue on average) during the simulated period of time.
- The average response time of the system to each request that was processed during this period of time (the response time is computed as the difference between the time the request has completed its execution on one of the servers and the time the request arrived to the queue).

Skeleton

```
Do {
   time++;
   run_events_for_time; //eg, new request
   update_state_of_system;
   update_statistics_you_need;
} while simulation_time_less_than_1000000
```

Arrival Rates

- A large computer cluster has, on average, 48 faults per day. 36 faults occurred in the first 6 hours of a certain day. How many faults are expected in the next 18 hours?

(To go back to Little's law, if every fault needs 30 minutes to repair, how many faults is the system expected to have at any point in time?)

You don't need to know about Little's law for lab exercise 3! All you need to do is to simulate the requests that arrive according to the arrival rate. E.g., at any point in time:

Generate random number between 0 and 1;

There is an arrival if this random number is less than 1/arrival_rate;

Random Numbers

- Every modern language has some library to generate random numbers (for example, in java, check java.util.Random; in C check rand(); etc…)
- You'll need to use some function to generate Random Numbers within a certain range
- You don't need to know this for the lab exercise, but these numbers are not truly random; they are pseudo-random numbers generated by carefully chosen mathematical formulae. If the initial condition is the same, the same numbers are generated!
- (e.g., RND=(8*RND+9)mod 10: 0,9,1,7,5,9 - there is a period, after which numbers are repeated)

This is an one session lab!