PERFORMANCE ENGINEERING

Lecture 9: Lost-and-found topics

May 23rd, 2022

Ana Lucia Varbanescu

a.l.varbanescu@uva.nl

To do today

- Revisit distributed systems
- Simulators as models
- Polyhedral model (also see online)
- Queuing theory

- Say “good bye” ☺

Revisit distributed systems

- … aka, large-scale systems

- Performance modelling
  - Computation – per node
    - Homogeneous or heterogeneous!
  - Communication
    - Various models
  - Synchronization
    - Often modelled in terms of latency due to barriers

Communication (cost) models [1]

- PRAM = parallel random access memory
  - Lock-step operation
  - Same access time to all memory locations
- CRCW PRAM
  - No cost for contention
- EREW PRAM
  - Atomic-like contention cost : 1 acces, P-1 wait

Communication (cost) models [2]*

The α - β cost model :

$$T(s)=\alpha+s\cdot\beta$$

- α = latency/synchronization cost per message
- β = bandwidth cost per byte
- Each processor can send and/or receive one message at a time

*More details: http://spcl.inf.ethz.ch/Teaching/2015-dphpc/lecture/lecture12-loggp-6up.pdf

L9Q1: Distributed MVM

- Sketch an implementation of a distributed memory matrix-vector multiplication
  - Assume P processors
  - Assume an N x N dense matrix of floats, N>>P
- Model the performance of the application, assuming no overlap between computation and communication.
- What is the communication cost, assuming:
  - Message unit = 4 bytes
  - α = 1 ms
  - β = 1 ms/byte
  - Fully-connected network
- What changes for a ring topology?
- What changes for a tree topology?

LastDay!

Communication (cost) models [3]*

- Network capacity, contention, number of processors, …
- **LogP** (S -> R)
  - L = **latency**
    - Time for fixed-length packet to traverse the network from S to R
  - o = **overhead**
    - Time spent at S, R to send/receive a packet
  - g = **gap**
    - minimum time interval between consecutive sends or receives on a given processor.
  - P = number of processors.

- P1-to-P2 time for 1 packet?
  - 2*o + L
- P1-to-P2 time for n packets?
  - 2*o+L + (n-1)*g

Summary from : https://people.eecs.berkeley.edu/~demmel/cs267-1995/lecture09.html

In summary

Distributed systems modelling is a combination of computation modelling (which we know), communication modelling (which has its own models and tools), and additional costs from:

- Resource management
- Scheduling
- Synchronization
- Load balancing

Metrics of interest

- Performance
- Scalability

Simulation-based modeling

What’s in a name?

- Simulation = the emulation of the (**simplified**) behavior of a real-world system over an interval of time.

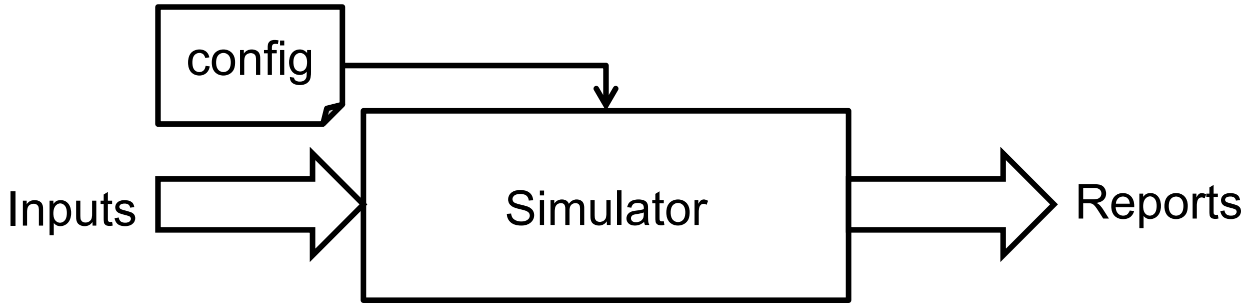
- An analytic model = a mathematical abstraction of the system
  - i.e., a representation that can be *mathematically analyzed* to deduce system behavior.

- A simulation model = an *algorithmic abstraction* of the system
  - i.e., a representation which, when *executed*, reproduces the behavior of the system

Why (not) simulation?

- De-facto standard in performance modeling for computer architecture
- Pro’s:
  - Good accuracy
  - Low-level details can be modeled
  - Can be cycle-accurate
  - **“Future-proof”**
- Con’s
  - Slow
  - Error-prone / difficult verification and validation
  - Mix of host and target systems

User’s perspective on simulators



- Configuration
  - Parameters for the configuration of the simulator
    - E.g.: cache sizes, processing units, clock frequency, …
- Inputs
  - Code
    - Either actual code or a model of the code
  - Data
    - Either input data or a trace
- Reports
  - Actual data of interest
    - E.g.: Cycles, CPI, IPC, overall execution time, cache behavior reports, errors, …

User's perspective on simulators

- Accuracy vs. execution time
  - Some examples:
    - Cycle-accurate simulators => the closest to the system, and slowest
    - Functional simulators => cut a few corners …
    - Behavioural simulators => more approximation
    - Timing/performance simulators
- Trace-driven vs. execution-driven
  - Trace-driven = input is a trace of an execution of a program
  - Execution-driven = input is the actual code
- Agent-based modeling
  - Uses (autonomous) agents to simulate complex system behavior
- …

Inside a simulator

- System = collection of “state variables”
  - Discrete
  - Continuous
- “Time”
  - Discrete (usually dictated by events)
  - Continuous (usually dictated by some form of a clock)

Common types of simulations

- Discrete simulation
  - Changes in the system dictated by *events*
  - State is discrete
- Continuous simulation
  - Changes in the system dictated by time
  - State is continuous
- Monte-Carlo simulation
  - Repeated simulation to collect possible outcomes of uncertain events
    - Input = known distribution of random variables
    - Computation = input to output transformation
    - Output = analysed statistically

Event-based simulation

- A system is characterized by its state and changed by events.

- State variables
  - Define the state of the system
- Events
  - Change the system state

- Example: given a queue …
  - State: length.
  - Events: add element, remove element
  - Performance metrics?

In other words, …

- A system is characterized by states :
  - State = {X(t), t in T} , where X are the variables of the system
  - Sample path = a set of states the system traverses over time
  - Run = generating a sample path for the system

- Simulation:
  - Generating and measuring representative runs in the system

- Representative … ?
  - Trace (from existing runs)
  - Special cases
  - Models of “normal”/special cases

Event-based simulators require

When built …

- An event scheduler
- Simulated time (& time management)
- System state variables
- Event routines

When run …

- Representative inputs
- Comprehensive output reports

Types of event-based simulations

- Time-driven
  - Input: time-based sequence of events
  - Time advances independently of the events.
  - Each event is scheduled on a specific time
  - Simulation is run for many time units

- Event-driven
  - Input: sequence of events
  - State changes event-by-event only
    - in between the events, nothing happens.
  - New time is calculated based on events.

- Example: queue
  - Time-driven ?
  - Event-driven ?

Challenge: representative input!

- To run representative simulations, representative input data are key!

- Getting representative input data
  - Synthetic cases
  - Pathological cases
  - Statistical distributions of events from different types of systems
  - Real traces from the system under study

- Workload characterization
  - Finding out the right workload for studying the system
    - Simulation model or not …

Traces and trace-based simulation

- Trace-based simulation uses pre-recorded traces of previous executions for prediction
- Two components:
  - Trace generation
    - Instrument code
    - Log data
    - Write data in some ordered manner
  - Trace analysis/simulation
    - Replay traces on different system / larger system / …
    - Measure/analyze outcome
- Main challenges:
  - Traces could be very large
  - Store/order traces (especially in parallel systems)

Stand-alone research field!

Workload characterization*

- Set of techniques and tools to build models of workloads

- Based on:
  - Measurement in production
    - Monitoring, logging, …
  - Analysis
    - Statistical analysis
    - Numerical fitting
    - Stochastic processes
- Workload models are used for
  - Symbolic simulation
  - Data flow models
  - Queuing theory models
  - …

Stand-alone research field!

*Maria Carla Calzarossa et al: *Workload Characterization: A Survey Revisited*

Simulators in practice

Useful and/or famous simulators

- CPU
  - Gem5 – cycle-accurate simulator for computer architecture
    - Simulates entire OS and applications
      - Slow, but very accurate
  - Sniper – behavioural simulator for application analysis
    - Simulates applications on cores, with some OS features
      - Fairly fast, fairly accurate
- Caching
  - Dinero IV – a classic
    - Input: trace
  - Cachegrind – in valgrind
    - Input: trace
  - pyCacheSim – python-based new case simulator
    - Input: assembly-like code
- GPU
  - GPGPU-Sim
    - Full GPU simulator

Queuing theory and modeling

Why do we care?

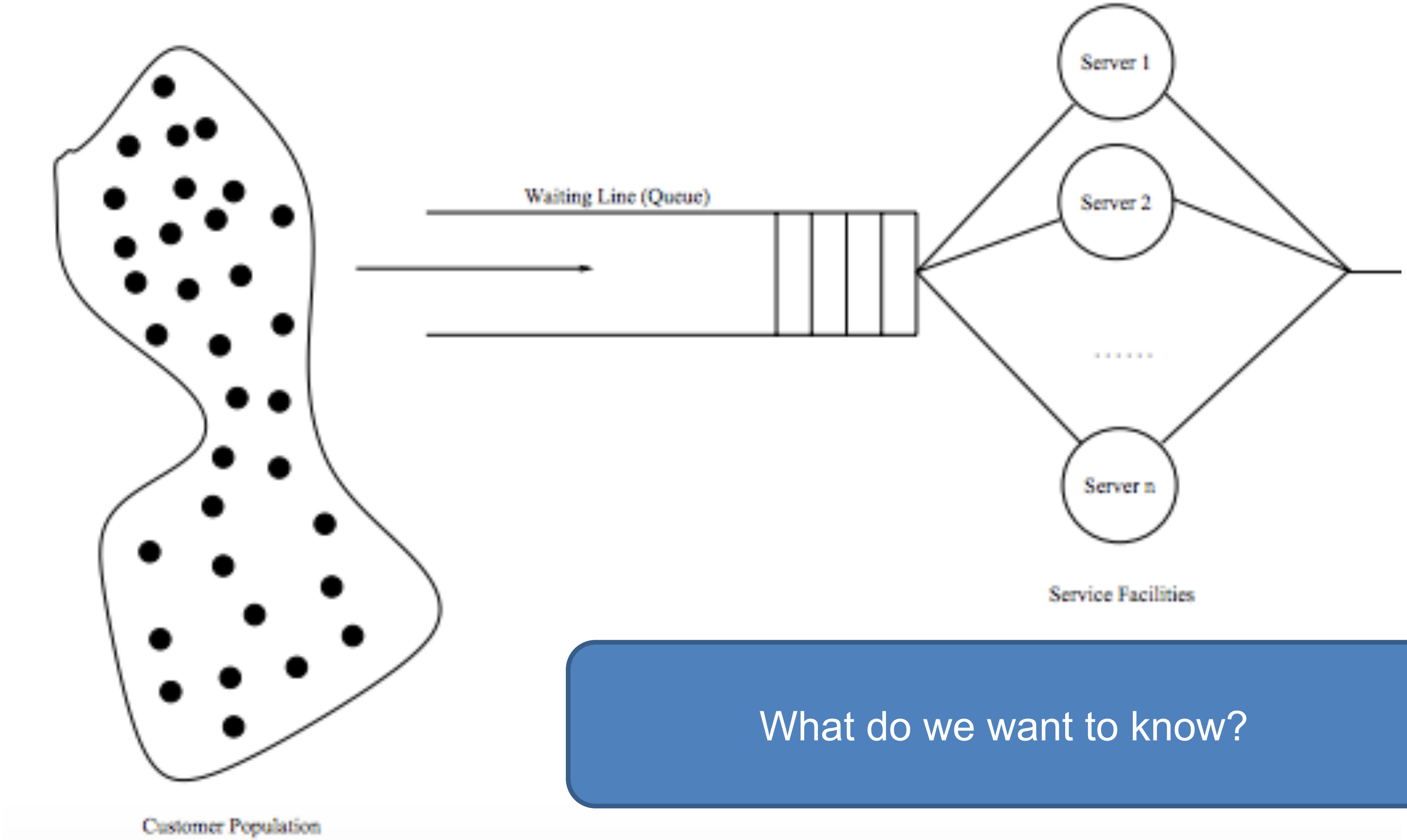
- Queuing is essential to understand the **behavior** of complex **computer** and **communication** systems
- In depth analysis of queuing systems is hard
  - … but the most important results are easy

- Useful for analyzing contention and its effects.

- Examples:
  - Service center
  - Multi-user (web-)server
  - Multi-threading on a CPU

  …

An example



What do we want to know?

Intuition & metrics

- Response time of a system
  - From request to result
- Utilization
  - How busy the system is (<=100%) on **average** over **time**
- Service time of a system
  - Time to process the request
- Arrival rate
  - The rate at which tasks (aka, customers) arrive (at service center)
- Throughput
  - The rate at which tasks leave (the service center)
- Wait time = response time – service time
  - Wait interval BEFORE the service starts

Little's Law

Given:

- λ = arrival rate (= mean throughput at stable state)
- L = average number of jobs/customers in the queue/system
- W = average time a job/customer spends in the system

$$L = \lambda * W$$

An example:

L = 5 jobs in queue

λ = 2 jobs/s are processed => W = L/λ = 2.5 s

Another example:

What is the throughput of a system with 100 jobs in the queue, and an average wait time 1s?

W=1s, L=100 => λ = L/W = 100 jobs/s

Model and notation [1]

- Customers (aka, “Jobs”, “Requests”)
  - Infinite number
  - Arrival time : τ_n
  - **Inter-arrival time**: $t_n = \tau_n - \tau_{n-1}$
    - Assumed independent and drawn from the same distribution A(t)
- Service
  - **Service time** : x_n
    - Assumed independent and drawn from a distribution B(t)

Model and notation [2]

- Size of waiting line
  - Finite
  - Infinite
- Number of servers
- Service discipline
  - FIFO
  - LIFO
  - Random
  - Round-robin
  - Priority-based

Kendall notation

A/B/m/N – S

- A = distribution of inter-arrival times
- B = distribution of service times
- m = number of servers
- N = maximum size of waiting queue
  - If N = ∞, it is omitted
- S = service discipline
  - If S is omitted => FIFO

A and B common abbreviations

- M (Markov) - exponential distribution, memoryless
  - “the probability distribution of the time between events in a Poisson point process, i.e., a process in which events occur **continuously** and **independently** at a **constant average rate**” [Wikipedia*]
    - CDF: $1 - e^{-\lambda x}$; PDF: $\lambda e^{-\lambda x}$ (mean: $1/\lambda$, variance: $1/\lambda^2$)
    - $\lambda > 0$ = rate

- D (deterministic) - all values from a deterministic “distribution” are constant (i.e., same value)

- G (general) – a general distribution
  - At least the mean and variance known

*https://en.wikipedia.org/wiki/Exponential_distribution

Simple systems

- M/M/1
  - FIFO service
  - single server
  - an infinite waiting line allowed
  - customer *inter-arrival times* are independent and exponentially distributed with some parameter λ
  - customer *service times* are also independent and exponentially distributed with some parameter μ

- For M/M/* systems => many stable state results are "easy" to compute
- For G/G/* systems => no analytical results available

M/M/1: Basics

- All distributions are memory-less

- P_k = probability that the system is in state k, i.e., has k customers

- Empty system:
  - $P_0 = 1 – \lambda/\mu = 1 - \rho$
- System with k customers:
  - $P_k = (\lambda/\mu)^k \times P_0$

A simulator: http://www.bluffton.edu/homepages/facstaff/nesterd/java/queueingsimulator.html

M/M/1: Derived metrics

- Utilization (ρ)
  - The fraction of time that the server is busy.
  - For M/M/1: the complementary event to when the system is empty.

$$1 - p_0 = \lambda/\mu = \rho$$

- Mean number of customers in the system

$$N = \rho / (1-\rho)$$

- Mean response time
  - the mean time a customer spends in the system

$$T = 1 / (\mu - \lambda) = 1/\mu * 1/(1-\rho)$$

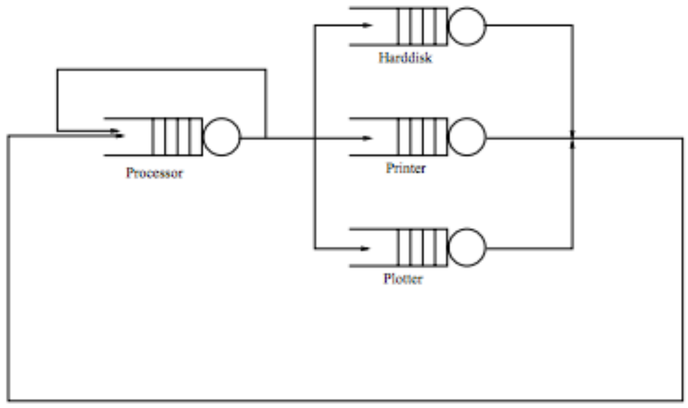
$$RT = serviceTime/(1-Utilization)$$

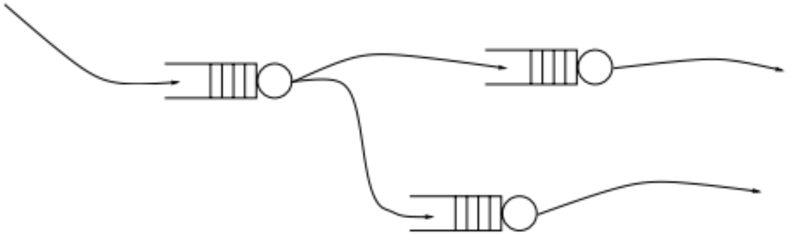
Capacity planning and loss probability

- Restrict customers to a fixed number
  - … whoever arrives at a full system is lost
- Capacity planning
  - How large should the queue be such that the probability to reject customers is “small” (given)?
- Loss probability
  - What is the loss probability for a system with a queue of capacity K?

Queuing networks

- … a network or queuing systems
- Closed = the number of customers is **fixed** and no customer enters or leaves the system
- Open = new customers **may arrive** from **outside** the system (conceptually infinite population) and **leave** the system





Key challenges

- Express your system (hardware and/or software) as a queuing system
  - Determine interarrival distribution
  - Determine service time distribution
  - Approximate λ, μ
- Map the questions/model outcomes to “easy” queuing theory results
  - What is utilization, capacity, …
- Understand the assumptions and restrictions
- Validation and re-design
  - When the model doesn’t match the measurements, it is highly likely that you have over-simplified the model

Use of queuing systems in PE

- Relevant modeling strategy for throughput and capacity related questions.
- Used to model complex systems in simple simulators
- Used to evaluate the scheduling of tasks on resources
  - Popular for large-scale systems
  - Applied on multi-core level, too
- Used to determine/reduce the load of systems
  - For datacenter utilization and price models
  - For resource provisioning

Optimizations & the polyhedral model

Motivation

Matrix Multiplication 3000x3000

```c
#include<stdio.h>

#define N 3000
#define M 3000

void initMatrices(int A[N][M],int B[N][M],int C[N][M]);
void printMatrix(int C[N][M]);

int main(){
static int A[N][M], B[N][M],C[N][M];

initMatrices(A,B,C);
for (int i = 0; i < N; i++)
    for (int j = 0; j < M; j++) {
        C[i][j] = 0;
        for (int k = 0; k < M; k++)
            C[i][j] += A[i][k] + B[k][j];
        }
printMatrix(C);
}
```

clang 9.0.0

clang -O3 gemm.c -o gemm.clang
time ./gemm.clang > clang.out
real 0m30.169s

clang 9.0.0 + Polly (Polyhedral Opts)

*clang -O3 gemm.c -o gemm.polly **-mllvm –polly***
time ./gemm.polly > polly.out
real 0m5.134s

clang 9.0.0 + Polly Parallel (Polyhedral Opts)

clang -O3 gemm.c -o gemm.polly.par
-mllvm –polly-mllvm
-polly-pattern-matching-based-opts=0
-mllvm -polly-parallel
***-mllvm -polly-num-threads=40** –lgomp*

time ./gemm.polly.par > polly.par.out
real 0m1.780s

Polyhedral : 5.87x speedup
Polly Parallel : 16.94x speedup
(ok… 40 threads, but still ☺)

The Polyhedral Model

- Mathematical model able to represent **loops**, **instructions**, and **memory locations**
  - + Able to model and apply arbitrary composition of program transformations
  - + Enable **exact dependence analysis** for **some** programs
    - + Multi-objective optimizations (SIMD, cache, parallelism, etc.)
  - - Not trivial to understand and use (i.e. to generate custom transformations).

- Used currently in compilers and code analysis
  - Automated parallelization & code optimization
  - Input for analytical models

End-of-course

Final thoughts

- The course covered
  - Performance measuring and profiling, performance analysis
  - Modeling
  - Performance optimization (a bit)

- Platforms
  - CPUs
  - (less) GPUs
  - (tiny bit) Larger scale systems

- Applications
  - HPC applications & close-to-metal models

- We skipped:
  - Software engineering aspects

Anything else you wanted to learn about?