Performance Engineering – Introduction

Holger Pirk

Slides as of 14/01/22 11:57:44

What is performance?

Let's look it up

Definition: Performance

According to the Cambridge Dictionary

Performance how well a person, machine, etc. does a piece of work or an activity

Okay, but how do you measure "well"?

In particular: "well" as it is perceived by the user

How do you measure "well"?

- The assumption is that the software is functional, i.e., bug-free
- The perceived quality beyond functionality is usually associated with execution speed
  - For now, let's say it is about end-to-end execution time

Now, you might be thinking:

I know this! This is High Performance Computing?

A distinction

High Performance Computing

- Focus is on a single problem that is...
  - ...of high value...
  - ...usually relatively simple (though solutions may be complex)...
  - ...sometimes supported by special-purpose hardware

Performance Engineering

- Focus is on systems, i.e., complex, flexible pieces of software

What is a System?

Examples of Systems:

- Big Data Systems: Spark, Flink, Trill, etc.
- DBMS: DB2, SQLServer, Oracle DBMS, MySQL, etc.
- AI: Tensorflow, Torch, Theano, etc.
- Operating System Kernels
- Operating System Tools: awk, grep, sed

What is a System?

Merriam Webster

- a regularly interacting or interdependent group of items forming a unified whole

Properties

- Often made up from **components**,
  - that interact
  - to achieve a **greater goal**
- Usually applicable to different problems/domains (i.e., generic)
- How is that different from a well-designed application?
  - The goal is domain-agnostic
  - Systems are designed to be flexible at runtime

What is a System?

Flexibility

- The exact conditions under which a system operates are unknown at development time
  - A Data management system does not know the data format/schema beforehand
  - An operating system does not know the number of users
  - Grep does not know what regex to search for
  - Tensorflow does not know what matrix dimensionality to expect

What is a System?

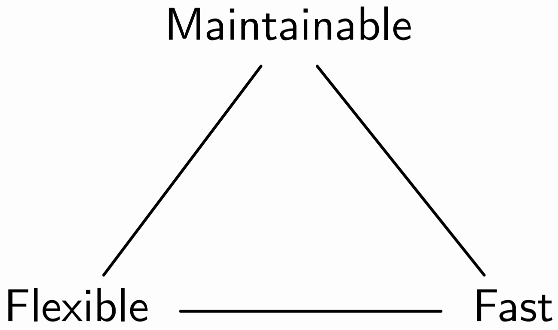
Further complications

- Software systems are complex and developed over years
- they need to be maintainable (you know how to write maintainable code)
- they need to be fast (if you have taken Advanced Architecture you know how to write fast code)

The Challenge:

The Challenge: Building a System that strikes a Balance

A classic trade-off triangle



Let's illustrate this with an example

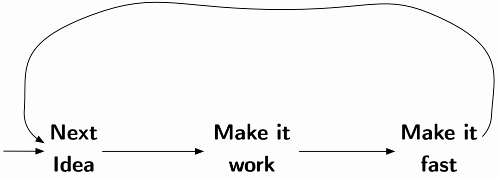
Challenge

- Develop a piece of software that counts the lines in a CSV file that containing the phrase " FAUST:"
- Apply what you have learned in your software engineering class

Interactive Coding!

The Process we just went through

Performance Engineering



What is Performance Engineering

According to Wikipedia

Performance engineering encompasses the techniques applied during a systems development life cycle to ensure the non-functional requirements for performance will be met.

Key Question: When do you stop optimizing?

Key Question: How fast is fast enough?

Step 1: Define a Target Metric

Examples for metrics

- throughput
- latency
- scalability
- memory usage
- energy consumption
- TCO
- elasticity
- efficiency
- etc.

Step 2: Decide when the Requirements are met

Two options:

- Easier: Setting an optimization budget (usually in terms of developer time)
  - Common customer request: make it as fast as possible
- Harder: Setting an optimization target/requirement/threshold
  - You may have real-time-requirements:
    - Soft: if your software misses the requirements, it is considered "an error"
    - Hard: if your software misses the requirements, it is considered "failed"
  - These are often called QoS objectives

Quality-of-Service (QoS) objectives

Definition

- Statistical properties of a metric that shall hold for the system
- Can include pre-conditions:

Example

The framerate of the game will, on average, be higher than 60 frames per second if run on a GPU with 50 GFlops or more.

- Sometimes in conflict with functional requirements (e.g., the realism of the AI)

Service-Level Agreements (SLAs)

- Formal, legal contracts specifying QoS objectives as well as penalties for violations

Example

Trading orders shall not exceed 1ms response time. In case of violation, the user is eligible for a 10% credit towards fees.

- Key question: how do you enforce SLAs?
- As stated, these are non-functional requirements!

When defining requirements, be SMART

Specific	State exactly what is acceptable in numeric terms
Measurable	Make sure what is stated can actually be measured
Acceptable	rigorous enough to guarantee success in reality
Realizable	lenient enough to allow implementation
Thorough	ensure that all necessary aspect of the system are specified

We will primarily focus on "Measurable"

Performance Evaluation Techniques

- Measuring
  - Monitoring
  - Benchmarking
- Analytical Modeling
- Simulation
- Hybrids:
  - measure, then model
  - model & simulate
  - ...

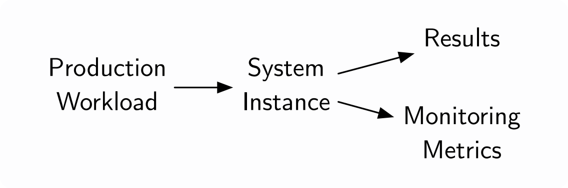
Measuring

- Performed on the actual system
- Can be on prototype or final system
- Can achieve good accuracy if done properly
- Often based on "instrumentation"
- Often costly
- Can be difficult

Monitoring, i.e., Measuring in Production

- Constant monitoring is required to enforce SLAs
  - Observe system performance,
  - Collect statistics,
  - Analyze data,
  - Report SLA violations
- Monitoring can incur costs
- Thus, often not continuous

Monitoring, i.e., Measuring in Production



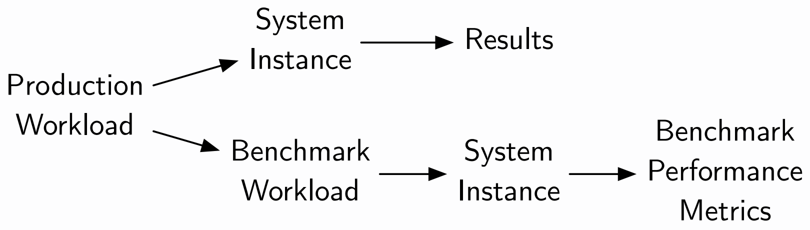
Benchmarking, i.e., measuring in the lab (diverging slightly from the book)

- Two step process:
  - Get the system into a predefined (or steady) state
  - Perform a series of operations (the workload) while measuring relevant performance metrics

Example

Database workloads usually have a data generator to make the system load a dataset (i.e., predefined state) and a query set (the series of operations).

Benchmarking, i.e., measuring in the lab (diverging slightly from the book)



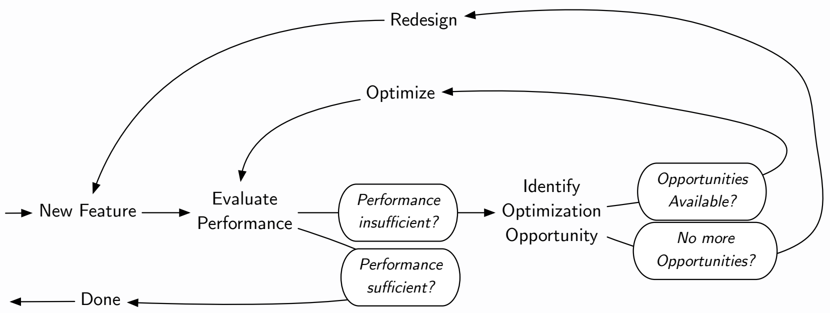
Benchmarking Workloads

- Can be batch (like a query set),
  - program usually has access to the entire batch from the start
  - useful when the metric is throughput
  - simpler because generator performance does not matter
- can be interactive (a program that generates requests)
  - generator generates work piece by piece (often randomly)
  - useful when the metric is latency
  - workload generator better be at least as fast as system under evaluation
- hybrids are quite common:
  - sample random queries from a predefined work set

Interpreting results (monitoring or benchmarking)

- A single measured datapoint is generally meaningless
- there is way too much noise in modern computer systems
- you need to aggregate multiple runs and
- report some measure of variance

The Optimization Loop



So, how do we identify optimization opportunities?

Parameters

Definition

- System and workload characteristics that affect performance

System Parameters: those that generally do not change while the system runs (caches, CPU instruction costs, . . .)

Workload Parameters: those that may change, even while the system is running (users, available memory, . . .)

- I will sometimes use the term "Resource Parameters" or just "Resources" to refer to parameters of the underlying platform

Numeric parameters

- CPU frequency
- Available memory
- Users
- Memory throughput
- ...

Nominal parameters

Example: Target device class

- Embedded devices
- phones
- laptops
- desktops
- servers
- mainframes
- clusters
- data centers
- geo-distributed

Utilization

- A service has a certain amount of resources available to perform
- examples are CPU cycles, memory capacity (in bytes), memory or network bandwidth (in bytes per second)
- the total amount/budget of available units of a resource are a parameter

Definition Utilization

the percentage of a resource that is used to perform the service

Bottleneck

Definition: Bottleneck

The resource with the highest utilization

Nomenclature

- We use the term x-bound to indicate that x is the bottleneck of an application:
  - CPU-bound
  - (disk-)bandwidth-bound
  - memory-bound (capacity)
  - ...

Bottleneck

Note

- Sometimes, factors bound the performance that are not strictly speaking resources.
  - Most important example: latency
  - latency-bound means most of the time, the systems waits for an operation to complete

Identifying bottlenecks for an entire complex software system is practically infeasible

Performance-Dominating Code Paths

- Remember: we are dealing with complex systems
- Many systems have a "performance-dominating" code path
- To limit optimization complexity, restrict your effort to such paths
- Some code paths have special names:
    - Critical path is the sequential part of the code
    - Hot path is the path that takes the most time

Good, now we know what to optimize!

But how do we do it?

The goal

- Quickly compare alternative designs (development) or
- Quickly select a close-to optimal value for a plattform parameter (tuning)

Parameter Tuning

- Workload parameters are usually not under one's control
- (Some) system parameters are

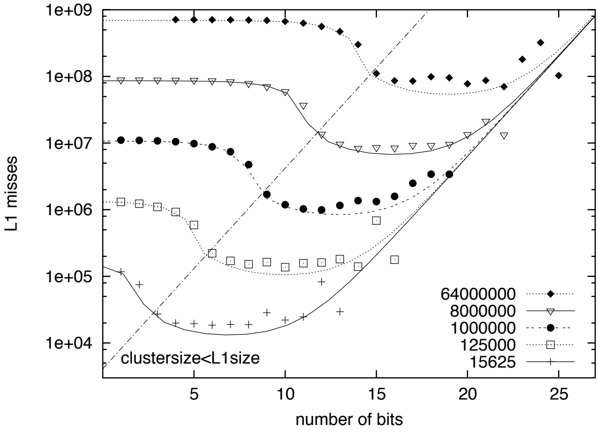
Parameter Tuning

Finding the vector in the parameter space that either

- minimizes the resource consumption or
- maximizes a performance metric

- Parameter tuning means exploring the parameter space
- Even with (non-linear) optimization, this is expensive
- Analytical models accelerate that process immensely

Parameter Tuning



see Boncz et al., "Database architecture optimized for the new bottleneck: Memory access"

Trading off

- Sometimes consumption of an expensive/non-scalable resource can be reduced by using more of a cheap one
- This often requires changes to the system
  - Examples? Take some time to think of some!
- There can be good, bad and ambiguous trade-offs

Analytical Performance Models

Definition

A formal characterization of the relationship between system parameters (hardware, software, data) and performance metrics.

- Sounds simple but is often very complicated
- The challenge is to model a dynamic system using a (reasonably small) static model
- Models can be stateless (e.g., characterizing equations) or stateful (e.g., markov chains)

Why analytical models are awesome

- They are fast
- they allow what-if analysis (simplifying tuning)
  - of both system as well as workload parameters
  - **this is the key to defining achievable performance requirements**
- Claim: If you can build an accurate analytical model, you have understood the system

Analytical Performance Models

Example 1

An I/O bound application, that needs to read **40MB per request** is limited to **10 requests per second** when running on a hardware platform with a **single disk** providing **400MB/s bandwidth**.

Example 2

A compute-bound application that needs **3 cycles to process one byte** is limited to **20 requests per second** if it needs to process **40MB per request** and the CPU runs at **2.4GHz**.

Simulation

Definition

A single observed run of a stateful model

Notes

- Simulations can be **extremely** expensive to calculate
  - In particular if the level of detail is high
- Personally, I rarely use simulations

Context

- We assume to be working on **systems**
- Developing fast algorithms is not the focus of this class (though we do touch upon them)

The first half of this course…

…focuses on

- Efficiency
- Scaling-up
- basically: getting the most bang for the buck

General rules

A wise man once said...

... "we should not grading get in the way of teaching"

For me this means...

- you are here to learn things first and get a good grade if you've learned what you need to know
- Please, come forward if there is something that doesn't make sense
- Finish the coursework to the best of our ability, leave the marking scheme to us

Also

- Focus is methodology

Lectures

- Lecturers Holger Pirk (me) & Lluis Villanova
  - I am a database dude that likes CPUs – I'm focusing broadly on efficiency
  - Lluis is a low-level systems dude that likes distributions – he will focus on scale-out processing
- My part of the lecture will consist of
  - One pre-recorded lecture per week (published on Monday)
  - One interactive session for Q&A and tutorials
- Please watch the pre-recorded lectures before the interactive ones
- I will use Panopto, Edstem and Zoom

Preparation (for my half of the course)

- There are resource for preparation on the webpage
  - Papers, blog posts, videos, podcasts, ...
- Consuming them isn't mandatory but
  - We will dedicate about 10 minutes before every lecture to discussing them
  - (more if necessary)

Books

- *Cover R. Jain*, "The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling," Wiley- Interscience, New York, NY, April 1991
- *David J. Lilja*, "Measuring Computer Performance: A Practitioner's Guide 1st Edition", Cambridge University Press; 1st edition (September 8, 2005)

Programming

- The language of choice is C++
- You do not need to be a wizard in template meta programming but
  - it helps :-)
- concepts are 100% transferable
- but not all other languages have C++'s versatility

Computer Architecture

- We will not be writing assembly, but
- We will be reading assembly (X86)
- You should have an idea what these things mean:
  - LRU
  - Instruction Pipeline
  - Branch Predictor
  - Compare and Swap
  - Non-temporal write

Coursework

- Two pieces in total, one with me, one with Lluis

Provide feedback, please!

https://co339.pages.doc.ic.ac.uk/feedback/introduction

Get the slides online

https://co339.pages.doc.ic.ac.uk/decks/Introduction.pdf