60017

System Performance Engineering

Imperial College London

Contents

Chapter 1

Introduction

1.1 Logistics

Dr Holger Pirk

Dr Luis Vilanova

First Half

- Hardware efficiency in complex systems
- Scaling up
- *"getting the most bang for buck"*

Second Half

- Scale out Processing

1.1.1 Extra Resources

UNFINISHED!!!

1.2 What is System Performance Engineering

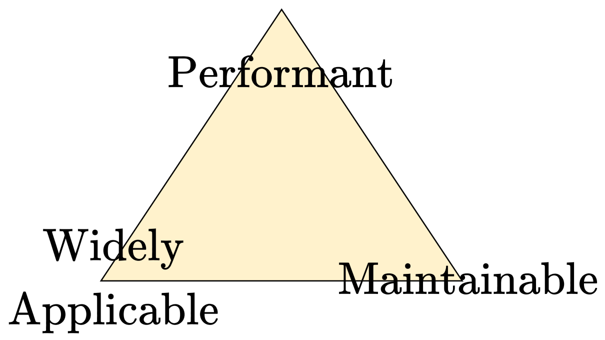
System **Definition 1.2.1**

A collection of components interacting to achieve a greater goal.

- Usually applicable to many domains (e.g a database, operating system, webserver). The goal is domain-agnostic
- Designed to be flexible at runtime (deal with other interacting systems, real conditions) (e.g OS with user input, database with varying query volume and type)
- Operating conditions are unknown at development time (Database does not know schema prior, OS does not know number of users prior, Tensorflow does not know matrix dimensionality prior)

Large & complex systems are typically developed over years by multiple teams.

The challenge with *system performance engineering* is to make systems maintainable, widely applicable and fast.



System Performance Engineering **Definition 1.2.2**

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

- Functional requirements (correctness, features) are assumed to be met.
-

High Performance Computing **Definition 1.2.3**

High performance programming uses highly distributed & parallel computer systems (e.g supercomputers, clusters) to solve advanced problems.

- Focuses on solving a single computationally difficult problem.
- Workloads are well defined and known at development time.
- Sometimes supported by custom hardware (e.g FPGAs, ASICs, custom CPU extensions)

1.3 Performance Engineering Process

1.3.1 Metrics

A *target metric* is used to quantitatively measure any improvement in *performance* (e.g for use in a *SLA*). The metric needs to be wel defined:

- When measuring starts (e.g when to measure latency from)
- Where measuring is done (is server response time measured on server, on a client, under what conditions?)

Imperials **Example Question 1.3.1**

Provide some example of metrics regarding a database.

Latency	Measuring time to query, planning time, the whole systems response time over a network.
Throughput	Measure the maximum request/second possible (often used to compare web-servers)
Memory Usage	measurable, but must be careful (e.g os interaction)
scalability	Can define a metric regarding how quickly some metric (e.g throughput) increases with scale (e.g instances of a distributed system)

It is also important to define when a requirement is satisfied.

- Setting an optimisation budget (e.g in developer hours)
- Setting a target or threshold (e.g $x\%$ over baseline implementation)
- Combination of both

1.3.2 Quality of Service (QoS) Objectives

Quality of Service Objectives **Definition 1.3.1**

A set of statistical properties of a metric that must hold for a system.

- Can include preconditions (e.g to define the environment/setup)
- Can be in conflict with functional requirements (e.g framerate vs realism in graphics)

Game On **Example Question 1.3.2**

Give an example of a basic QoS Objective for a game's framerate.

The game's framerate will be on average (over *...preconditions...*) $60fps$ if run on a GPU rated at $50GFlops$ or higher.

1.3.3 Service Level Agreements

Service Level Agreements (SLAs) **Definition 1.3.2**

Legal contracts specifying *QoS objectives* and penalties for violation.

- Non-functional requirements (not about system correctness)
- Can be legally enforced

Amazon ***Extra Fun!* 1.3.1**

Amazon Web Services (AWS) provides a set of *service level agreements* relating to performance and availability. Violations are resolved by providing customers with service credits. Amazon SLAs

When defining requirements for an SLA:

Specific	State exact acceptance criteria (numerical terms).
Measurable	Ensure the metrics used can actually be measured.
Acceptable	Requirements should be rigorous such that meeting them is a meaningful success.
Realisable	Counter to **Acceptable** - need to be lenient enough to allow implementation.
Thorough	All necessary aspects of the system are specified.

1.4 Performance Evaluation Techniques

1.4.1 Measuring

- Performed on the actual system (can be prototype or production/final).
- Can be difficult and costly (need to mitigate any impact of the measuring system on the system itself).
- As it is on the actual system, it can (if done properly) yield accurate results.

The two main types of measurement are:

Monitoring **Definition 1.4.1**

Measuring in production to get real usage performance metrics.

- Observe the system in its production environment
- Collect usage statistics and analyse data (e.g user's preferred query types/structure, schema designs for databases)
- Can monitor for and report SLA violations.

Benchmarking **Definition 1.4.2**

Measuring system performance in a controlled setting (e.g lab).

- The system to set into a predefined (or steady/hot) state
- Perform some workload while measuring performance metrics.

Benchmarking requires representative workloads in order to get metrics likely to be representative of a production environment.

Batch Workload — **Definition 1.4.3**

Program has access to entire batch at start of the benchmark.

- Useful when a throughput metric is being measured
- Simple to generate, and can even be recorded from a production environment.

Interactive Workload — **Definition 1.4.4**

A program generates requests to pass to the system being benchmarked.

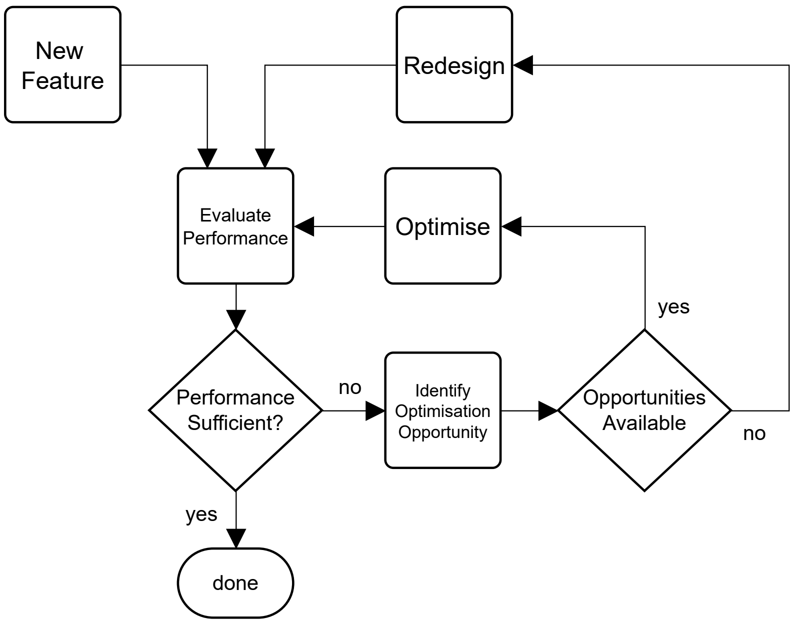
- Useful when a latency metric is being measured.
- Workload generator needs to fast enough to saturate system being benchmarked.
- Often more representative of a production environment (e.g an operating system receives a workload over time)

Hybrid — **Definition 1.4.5**

A common setup combining batch and interactive workload strategies (e.g sample random queries from a predefined work set).

In order to get useful results from which

1.5 Optimisation Loop



Performance Parameters — **Definition 1.5.1**

System and workload characteristics that affect performance.

System Parameters	Do not change as the system runs (instruction costs, caches)
Workload Parameters	Change as he system runs (available memory, users)
Numeric Parameters	Quantitative (e.g CPU frequency, available memory, number of user)
Nominal Parameters	Qualitative parameters (Runs on battery, has a GPU, runs in a VM)

The term *resource Parameters/Resources* refers to the parameters of the underlying platform (e.g CPU, memory).

Utilisation **Definition 1.5.2**

The proportion of a resource used by a to perform a service by a system.

- A service has limited resources available (e.g CPU time, memory capacity, network bandwidth etc)
- Total available resources/resource budget available to a service is a parameter

Bottleneck **Definition 1.5.3**

The resource with the highest utilisation.

- The limiting factor in performance of a system
- given some resource x is the bottleneck, the system is x-bound (e.g CPU-bound).
- Not always a resource, and performance may be bottlenecked by some other factor (e.g latency-bound $\rightarrow$ the system is dominated by waiting for some operation)

It is typically infeasible to identify all bottlenecks for an entire complex software system.

To limit optimisation complexity efforts should be restricted to optimising code paths that have particularly large effect on performance.

Critical Path **Definition 1.5.4**

The sequence of activities which contribute the larges overall duration.

Hot Path **Definition 1.5.5**

A code path where most of the execution time is spent (e.g very commonly executed subroutine)

In order to optimise we require:

- Ability to quickly compare alternative designs
- Ability to select a near optimal value for platform parameters

While workload parameters are not typically controllable at this stage, some system parameters are.

Parameter Tuning **Definition 1.5.6**

Finding the vector within the parameter space that minimises resource usage, or maximises performance.

- Exploring the parameter space is expensive (even with non-linear optimisation)
- Analytical models can be used to accelerate search.
- Tuning needs to consider tradeoffs (e.g much of a cheap resource versus little of an expensive one)

Analytical Performance Model **Definition 1.5.7**

A model describing the relationship between system parameters and performance metrics.

- Having an accurate analytical model for the system is analogous to *understanding* the performance of the system.
- Models can be stateless (e.g an equation) or stateful (e.g using markov chains).
- Need to model dynamic systems with (ideally small) static models.
- Very fast (faster than search parameter space)
- Allow for *what-if analysis* of system and workload parameters.

Simulation **Definition 1.5.8**

A single observed run of a stateful model

- Can see all interactions within the system in perfect detail
- Extremely expensive to run (limiting number of simulations and the speed of the optimisation workflow)
- Much more rarely used than other techniques described

Chapter 2

Credit

Content

Based on the Type Systems course taught by Dr Holger Pirk and Dr Luis Vilanova.

These notes were written by Oliver Killane.