Abstract

Architecting System Performance applies and elaborates the course Architectural Reasoning Using Conceptual Modeling to architect performance of systems. We teach an architecting method based on many views and fast iteration of the views. Visual models, functional models, and mathematical models in all views are the means to communicate about the system, to discuss specification and design choices, to reason about consequences, and to make decisions.

Distribution

This article or presentation is written as part of the Gaudí project. The Gaudí project philosophy is to improve by obtaining frequent feedback. Frequent feedback is pursued by an open creation process. This document is published as intermediate or nearly mature version to get feedback. Further distribution is allowed as long as the document remains complete and unchanged.
Abstract

Course overview of the course Architecting System Performance.
# Nuggets Architecting System Performance

## Course Overview

|------------------------|-----------------------|--------------------------|

- **time-oriented performance**

---

{Architecting System Performance; Course Overview}

version: 0.3
July 3, 2023
ASPCOnuggets

Gerrit Muller
## Assignments in Face-to-Face Module

<table>
<thead>
<tr>
<th>0. elevator case</th>
</tr>
</thead>
<tbody>
<tr>
<td>supersystem</td>
</tr>
</tbody>
</table>

### 1. sketch the problem
- goal
- use case

### 2. make conceptual model of the current situation
- model dynamic behavior
- model 0-order kpp using functions (as simple as possible)
- quantify contribution to kpp using observed data

### 3. explore customer and business relevance
- develop story
- model workflow and performance
- model customer value as function of kpp

### 4. make conceptual model of potential solutions
- model the foreseen solution
- model & compare 2 alternative solutions

### 5. list questions and uncertainties, reformulate problem and goal, and formulate gaps and options

### 6. develop an elevator pitch to report your findings and recommendations to management
Abstract

Listing the course material for Architecting System Performance
The ASP™ course is partially derived from the EXARCH course developed at Philips CTT by Ton Kostelijk and Gerrit Muller.

Extensions and additional slides have been developed at ESI by Teun Hendriks, Roland Mathijssen and Gerrit Muller.
<table>
<thead>
<tr>
<th>core</th>
</tr>
</thead>
<tbody>
<tr>
<td>Physical Models of an Elevator</td>
</tr>
<tr>
<td><a href="http://www.gaudisite.nl/info/ElevatorPhysicalModel.info.html">http://www.gaudisite.nl/info/ElevatorPhysicalModel.info.html</a></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>Teaching conceptual modeling at multiple system levels using multiple views</td>
</tr>
<tr>
<td>Understanding the human factor by making understandable visualizations</td>
</tr>
<tr>
<td><a href="http://www.gaudisite.nl/info/UnderstandingHumanFactorVisualizations.info.html">http://www.gaudisite.nl/info/UnderstandingHumanFactorVisualizations.info.html</a></td>
</tr>
<tr>
<td>core</td>
</tr>
<tr>
<td>------</td>
</tr>
<tr>
<td>Architecting System Performance; Course Didactics</td>
</tr>
<tr>
<td><a href="http://www.gaudisite.nl/info/ASPcourseDidactics.info.html">http://www.gaudisite.nl/info/ASPcourseDidactics.info.html</a></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSRP: <a href="https://en.wikipedia.org/wiki/DSRP">https://en.wikipedia.org/wiki/DSRP</a></td>
</tr>
<tr>
<td>Assumptions: “Systems Engineering and Critical Reflection: The Application of Brookfield and Goffman to the Common Experiences of Systems Engineers” by Chucks Madhav; proceedings of INCOSE 2016, in Edinburgh, GB</td>
</tr>
<tr>
<td>70/20/10:</td>
</tr>
<tr>
<td><a href="http://charles-jennings.blogspot.nl/">http://charles-jennings.blogspot.nl/</a></td>
</tr>
<tr>
<td><a href="http://jarche.com/2015/11/the-bridge-from-education-to-experience/">http://jarche.com/2015/11/the-bridge-from-education-to-experience/</a></td>
</tr>
<tr>
<td>Assumptions and beliefs:</td>
</tr>
<tr>
<td><a href="https://pivotalthinking.wordpress.com/tag/ladder-of-inference/">https://pivotalthinking.wordpress.com/tag/ladder-of-inference/</a></td>
</tr>
<tr>
<td><a href="http://stwj.systemswiki.org/?p=1120">http://stwj.systemswiki.org/?p=1120</a></td>
</tr>
</tbody>
</table>
## Greedy and Lazy Patterns

<table>
<thead>
<tr>
<th>core</th>
</tr>
</thead>
<tbody>
<tr>
<td>Architecting System Performance; Greedy and Lazy Patterns</td>
</tr>
<tr>
<td><a href="http://gaudisite.nl/info/ASPgreedyAndLazy.info.html">http://gaudisite.nl/info/ASPgreedyAndLazy.info.html</a></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fundamentals of Technology</td>
</tr>
<tr>
<td><strong>core</strong></td>
</tr>
<tr>
<td>---------</td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>optional</strong></th>
<th>Performance Method Fundamentals</th>
</tr>
</thead>
<tbody>
<tr>
<td>Measurement issues; From gathering numbers to gathering knowledge by Ton Kostelijk</td>
<td><a href="http://www.gaudisite.nl/MeasurementExecArchSlides.pdf">http://www.gaudisite.nl/MeasurementExecArchSlides.pdf</a></td>
</tr>
</tbody>
</table>
Abstract

This presentation presents the ideas behind the course Architecting System Performance. A number of frameworks and mental models show the context of this course and the approach to performance advocated in this course.
Performance attributes require means for analysis, evaluation, and creation of structure (parts and interfaces) and dynamic behavior (functions) at all levels.

Hence, we need conceptual modeling at all levels.
What and Why to Model

- How well is the customer served?
- How credible becomes the solution?
- How much are time and effort reduced?
- How much is the risk reduced?
- How much is the solution improved?

Purpose and type of model depend on project life cycle
Type of model and views depend on purpose

**customer**
- Key drivers
- Risks

**business**
- Key drivers
- Risks

**business as usual**
- (No modeling)
- Obvious
- Historic data
- Competitive data

**modeling**
- Feasibility
- Communication
- Risk mitigation
- Exploration
- Validation

**decision factors:**
- Accuracy of model
- Credibility of results
- Level of abstraction
- Working range
- Calibration of model
- Robustness of model
- Time to first results and feedback
- Effort
- Evolvability (adaptation to new questions)

- How much effort is needed to create model(s)?
- How much effort is needed to use and maintain model(s)?
- How much time is needed to obtain useful result?
Modeling Evolves over Time

- understanding
- exploration
- optimization
- verification

project phase determines purpose of the model determines type of the model
The Modeler’s Mindset Evolves too

understanding exploration optimization verification

Explorative
what is needed?
what can be achieved?

Defensive
what are the risks?
will the system perform well?
how to mitigate shortcomings?
The Architect Can Be "Out of Phase"

understanding  exploration  optimization  verification

Explorative
what is needed?
what can be achieved?

Defensive
what are the risks?
will the system perform well?
how to mitigate shortcomings?

mindset of most
stakeholders

mindset of architect

“look ahead”
10 Fundamental Recommendations

**principles**
- use feedback
- work incremental
- work evolutionary
- be explicit
- make issues tangible

**objectives**
- support communication
- facilitate reasoning
- support decision making
- create
- maintain
- understand
- insight
- overview

**recommendations**
- Time-box
- Iterate
- Quantify early
- Measure and validate
- Multiple levels of abstraction
- (Simple) mathematical models
- Analysis of accuracy and credibility
- Multi-view
- System and its context
- Visualize

 HELP TO ACHIEVE

translate into

ARCHITECTING SYSTEM PERFORMANCE; MANAGING SYSTEM PERFORMANCE

version: 0.2
July 3, 2023
MAO recommendations

Architecting System Performance; Managing System Performance
Gerrit Muller
Iterative Performance Management during Development

- measure
- evaluate
- analyse

- determine most important and critical requirements
- model
- analyse constraints and design options

- simulate
- build proto
- measure
- evaluate
- analyse
Managing Performance during Product Development

- Measurement:
  - Incomplete understanding
  - Calibration
  - Input

- Design:
  - Estimate and uncertainty

- Specifications:
  - Robustness problem

- Performance:
  - Measurement
  - Design
  - Estimation and uncertainty

- Finished product:
  - Degradation performance

Graph showing:
- Better vs. worse performance over time
- Incomplete understanding vs. calibration input
- Design robustness problem
Quantification Steps

- order of magnitude
- guestimates
- calibrated estimates
- feasibility, measure, analyze, simulate
- back of the envelope
- benchmark, spreadsheet calculation
- cycle accurate
- accurate
Abstract

The didactics behind a course like Architecting System Performance is a challenge, because the learning goals relate mostly to attitude and ways of thinking. At the same time, the material covers methods, techniques, tools, and concepts, which may lure participants in mechanistic approaches. Core in the didactic approach is reflection. This presentation offers some "thinking models" to assist reflection.
Competence Requires Various Learning Styles

- **what**
  - Knowledge
  - Skills
  - Ability
  - Attitude

- **how**
  - lecturing
  - exercises
  - assignments
  - practice
  - coaching
  - reflection

- **who**
  - participant
  - teacher/coach
Bloom’s Taxonomy and Higher Order Thinking Skills

Higher Order Thinking Skills
- more difficult to teach
- more valuable
- takes time to develop

Lower Order Thinking Skills
- people can acquire them fast

Remembering
Understanding
Applying
Analyzing
Evaluating
Creating

Higher Order Thinking Skills
must be mastered before, however when missing can be acquired fast

Lower Order Thinking Skills
people can acquire them fast
Course Assumption:

This course focuses on Higher Order Thinking Skills.

We assume

that you have appropriate knowledge

and

that you are able to find and absorb

required specific knowledge fast.
Problem-Based Learning Using Reflection

source: Kolb's learning cycle
http://www.infed.org/biblio/b-explrn.htm
Role of Experience in Learning

70:20:10 learning model

70: Experience

20: Exposure

10: Education

Modeling
Coaching
Scaffolding
Articulation
Reflection
Exploration

DSRP Model

- Making **Distinctions**
  - identity
  - Distinction
  - other

- Organizing **Systems**
  - System

- Recognizing **Relationships**
  - Relation

- Taking **Perspectives**
  - Perspective
mental switch from problem/system to “meta” how, what, why?

reflection wall

flips team 4

flips team 1

team 4

team 1

flips team 3

team 3

flips team 2

team 2
Scope and Topic of Reflection

- operational or life cycle context
- system of interest
- component or function of interest

- organization
- project
- team
- individual

- principle
- process or method
- procedure or technique
- tool or notation

- technical
- psychosocial
- means
The Role of Assumptions and Beliefs in Thinking

The “Ladder of Inference” originally proposed by Chris Argyris and developed by Peter Senge and his colleagues [The Fifth Discipline Fieldbook] illustrates how these biases can be built into our thinking. 
https://pivotalthinking.wordpress.com/tag/ladder-of-inference/

observe data
select data
add meaning
make assumptions
draw conclusions
adopt beliefs
take actions
reflexive loop

beliefs influence what we observe

Abstract

System Performance plays a crucial role in the customer value proposition and the business proposition. Minor details deep down into the system may have a large impact on system performance, and hence on both value propositions. Challenge in architecting system performance is to connect both worlds, which are mentally far apart.
Organizational Problem: Disconnect

What does Customer need in Product and Why?

Customer objectives
Application
Functional
Conceptual
Realisation

How can the product be realized
What are the critical decisions

system requirements
design decisions
parts connections
lines of code
and growing every year...

Architecting System Performance; Connecting Breadth and Depth
Gerrit Muller

version: 0
July 3, 2023
RATWdisconnect
Architect: Connecting Problem and Technical Solution

**Customer objectives**

**Application Functional Conceptual Realisation**

**What** does Customer need in Product and **Why**?

**How** can the product be realized

**What** are the critical decisions

---

**System requirements**

- design decisions
- parts connections
- lines of code

**and growing every year...**

---

**number of details**

- $10^0$
- $10^1$
- $10^2$
- $10^3$
- $10^4$
- $10^5$
- $10^6$
- $10^7$
- $10^8$

---

Architecting System Performance; Connecting Breadth and Depth

version: 0

July 3, 2023

Gerrit Muller

RATWbreadthAndDepth
Major Bottleneck: Mental Dynamic Range

Architecting System Performance; Connecting Breadth and Depth

version: 0
July 3, 2023
Breadth

System of Interest
- supporting systems
  - train
  - plan
  - maintain
- surrounding systems
  - supply
  - receive
  - manage
- stakeholders
  - concerns
  - needs
  - interests
- regulations
  - processes
  - procedures
Devilish details in design space may have large impact on performance. Many detailed design decisions determine system performance.
Abstract

Principles and concepts of modeling performance.
**Empirical** model: a model based on **observations** and **measurements**.

An empirical model **describes** the observations.

An empirical model provides **no understanding**.

**First principle** model: a model based on **theoretical** principles.

A first principle model **explains** the desired property from first principles from the **laws of physics**.

A first principle model **requires values** for **incoming parameters** to calculate results.
A **conceptual model** is a model explaining observations and measurements using a selection of first principles. A conceptual model is a hybrid of empirical and first principle models; simple enough to understand and to reason, realistic enough to make sense.
### From Zero to Higher Order Formulas

<table>
<thead>
<tr>
<th>Order</th>
<th>Description</th>
<th>Formula</th>
</tr>
</thead>
<tbody>
<tr>
<td>0(^{th}) order</td>
<td>Main function</td>
<td>(t_{\text{top floor}} = \frac{S_{\text{top floor}}}{v_{\text{max}}})</td>
</tr>
<tr>
<td>1(^{st}) order</td>
<td>Add most significant secondary contributions</td>
<td>(t_{\text{top floor}} = \frac{S_{\text{top floor}}}{v_{\text{max}}} - a_{\text{max}} \frac{t_{a}^2}{v_{\text{max}}} + 2 \frac{v_{\text{max}}}{a_{\text{max}}} + 2 \frac{a_{\text{max}}}{j_{\text{max}}})</td>
</tr>
<tr>
<td>2(^{nd}) order</td>
<td>Add next level of contributions</td>
<td>(t_{\text{top floor}} \approx \frac{S_{\text{top floor}}}{v_{\text{max}}} - a_{\text{max}} \frac{t_{a}^2}{v_{\text{max}}} + 2 \frac{v_{\text{max}}}{a_{\text{max}}} + 2 \frac{a_{\text{max}}}{j_{\text{max}}})</td>
</tr>
</tbody>
</table>
Abstract

A recurring question in modeling and performance analysis is when to stop digging. What level of detail is needed to achieve acceptable performance? What level of abstraction result in credible and sufficiently accurate results? How to cope with many levels of abstraction?
Level of Abstraction Single System

- Static system definition
- Monodisciplinary
- Number of details
- System requirements
- Multidisciplinary design
- Static system definition
- Monodisciplinary

Architecting System Performance; Level of Abstraction

Gerrit Muller

version: 0
July 3, 2023
RAPpyramid
From system to Product Family or Portfolio

Architecting System Performance; Level of Abstraction
Gerrit Muller

version: 0
July 3, 2023
DRALpyramidGrowth
The seemingly random exploration path

thinking path of an architect during a few minutes up to 1 day
Coverage of problem and solution space

- covered or touched by architects
- covered by engineers and experts
- level of detail

Subjects
Many Levels of Abstraction

- Key performance
- Performance definition
- Elaborated use cases
- Performance models
- Budgets and measurements
- Component designs
- Killing details

Architecting System Performance; Level of Abstraction

version: 0
July 3, 2023
ASFLAlevels
Fidelity Properties

low fidelity
- low effort
- fast

what fidelity is needed for:
- planning
- training
- validation
- design exploration?
what configurations do we need?
what can we afford?

high fidelity
- large effort
- slow

Enterprise context
- enterprise
- multidisciplinary
- systems
- stakeholders

number of details
10^0
10^1
10^2
10^3
10^4
10^5
10^6
10^7
10^8
10^9
Visualizing Dynamic Behavior

by Gerrit Muller    TNO-ESI, University of South-Eastern Norway]

    e-mail: gaudisite@gmail.com

www.gaudisite.nl

Abstract

Dynamic behavior manifests itself in many ways. Architects need multiple complementary visualizations to capture dynamic behavior effectively. Examples are capturing information, material, or energy flow, state, time, interaction, or communication.
Overview of Visualizations of Dynamic Behavior

Information Transformation Flow

Concrete “Cartoon” Workflow

Abstract Workflow

Timeline of Workflow

Swimming Lanes Concurrency and Interaction

State Diagram

Signal Waveforms

Flow of Light

Information Centric Processing Diagram
Visualizing Dynamic Behavior

version: 0
July 3, 2023
SSMEtypicalWorkoverOperationCartoon
Workflow as Functional Model

1. Assembly, functional test
2. Run EDP/LRP
3. Run risers
4. Hook up SFT and TF
5. Move above well
6. ROV assisted connect
7. System function and connection seal test
8. Run coil tubing and wireline BOP
9. Retrieve SFT and TF
10. Retrieve risers
11. Retrieve EDP/LRP
12. Retrieve coil tubing and wireline BOP

Perform well workover operations:
- Unhook coil tubing and wireline BOP
- Retrieve coil tubing and wireline BOP

13. Perform workover operations
14. ROV assisted disconnect
15. ROV assisted connect

Move away from well:
- Move away from well

Move above well:
- Move above well

Wellhead system function and connection:
- ROV assisted connect
- ROV assisted disconnect

Retrieval process:
- Retrieve SFT and TF
- Retrieve risers
- Retrieve EDP/LRP
- Retrieve coil tubing and wireline BOP

Visualization of dynamic behavior:
- Rig
- Vessel or platform assembly
- Functional test run
- EDP/LRP run
- Run risers
- Hook up SFT and TF
- ROV assisted connect
- ROV assisted disconnect
- System function and connection seal test
- Run coil tubing and wireline BOP
- Disassembly
Workflow as Timeline

assumptions:
running and retrieving risers: 50m/hr
running and retrieving coiled tubing/wireline: 100m/hr
depth: 300m

Visualizing Dynamic Behavior
Gerrit Muller
SSMEtypicalWorkoverOperationTimeline
version: 0
July 3, 2023

Swimming Lane Example

Visualizing Dynamic Behavior
56  Gerrit Muller
Example Signal Waveforms

$G_y = 0$

$G_y = 127$

imaging = repeating similar pattern many times

typical TE: 5..50ms

transmit receive
Example Time Line with Functional Model

- Call family doctor
- Visit family doctor
- Call neurology department
- Visit neurologist
- Call radiology department
- Examination itself
- Diagnosis by radiologist
- Report from radiologist to neurologist
- Visit neurologist

Days: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25

Functional flow:

days
Information Centric Processing Diagram

Visualizing Dynamic Behavior

Gerrit Muller
Example State Diagram

idle

start

operating

event

pre-alarm mode

acknowledge

alarm mode

reset

alarm handled
Flow of Light (Physics)

**Diagram:**
- Laser
- Illuminator
- Sensor
- Reticle
- Lens
- Wafer
- Aerial Image
- NA
- Aberrations
- Transmission

**Text:**
- Laser:
  - Pulse-freq, bw, wavelength, ...
- Illuminator:
  - Uniformity
- Sensor
- Reticle
- Lens
- Wafer
- Aerial Image
- NA
- Aberrations
- Transmission
Dynamic Behavior is Multi-Dimensional

How does the system work and operate?
Functions describe *what* rather than *how*.
Functions are *verbs*.
Input-Process-Output paradigm.
Multiple kinds of flows:
  physical (e.g. hydrocarbons, goods, energy)
  information (e.g. measurements, signals)
  control
Time, events, cause and effect
Concurrency, synchronization, communication

multi-dimensional information and dynamic behavior

Visualizing Dynamic Behavior
62    Gerrit Muller
version: 0
July 3, 2023
VDIkeyPhrases
Abstract

The essence of a system is that the parts together can do more than the separate parts. The interaction of the parts results in behavior and properties that cannot be seen as belonging to individual parts. We call this type of behavior "emerging behavior".
Emergence is Normal and Everywhere

emergent behavior and properties =

function of

**dynamic interaction** between

parts in the system and

context of the system

examples

- flying and stalling of an airplane
- Tacoma bridge resonance
Emergence, Desire, and Foreseeing

- Foreseen
- Foreseen, but underestimated
- Unforeseen

- Mitigated
- Risk
- Side-effect

- Undesired
- Desired

Goal of design
Modeling and Analysis: Budgeting

by Gerrit Muller        TNO-ESI, HSN-NISE
     e-mail: gaudisite@gmail.com
           www.gaudisite.nl

Abstract

This presentation addresses the fundamentals of budgeting: What is a budget, how to create and use a budget, what types of budgets are there. What is the relation with modeling and measuring.

Distribution

This article or presentation is written as part of the Gaudí project. The Gaudí project philosophy is to improve by obtaining frequent feedback. Frequent feedback is pursued by an open creation process. This document is published as intermediate or nearly mature version to get feedback. Further distribution is allowed as long as the document remains complete and unchanged.

July 3, 2023
status: preliminary draft
version: 1.0
content of this presentation

What and why of a budget

How to create a budget (decomposition, granularity, inputs)

How to use a budget
What is a Budget?

A budget is

a quantified instantiation of a conceptual model

A budget can

prescribe or describe the contributions

by parts of the solution

to the system quality under consideration
Why Budgets?

- to make the design explicit
- to provide a baseline to take decisions
- to specify the requirements for the detailed designs
- to have guidance during integration
- to provide a baseline for verification
- to manage the design margins explicitly
Visualization of Budget Based Design Flow

Modeling and Analysis: Budgeting

Gerrit Muller

version: 1.0
July 3, 2023
EAAbudget
## Stepwise Budget Based Design Flow

<table>
<thead>
<tr>
<th>Step</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>1A measure old systems</td>
<td>micro-benchmarks, aggregated functions, applications</td>
</tr>
<tr>
<td>1B model the performance starting with old systems</td>
<td>flow model and analytical model</td>
</tr>
<tr>
<td>1C determine requirements for new system</td>
<td>response time or throughput</td>
</tr>
<tr>
<td>2 make a design for the new system</td>
<td>explore design space, estimate and simulate</td>
</tr>
<tr>
<td>3 make a budget for the new system:</td>
<td>models provide the structure measurement and estimates provide initial numbers specification provides bottom line</td>
</tr>
<tr>
<td>4 measure prototypes and new system</td>
<td>micro-benchmarks, aggregated functions, applications profiles, traces</td>
</tr>
<tr>
<td>5 Iterate steps 1B to 4</td>
<td></td>
</tr>
</tbody>
</table>
Budgets Applied on Waferstepper Overlay

- **process overlay**: 80 nm
- **matched machine**: 60 nm
- **process dependency sensor**: 5 nm
- **reticle**: 15 nm
- **single machine**: 30 nm
- **lens matching**: 25 nm
- **matching accuracy**: 5 nm
- **stage grid accuracy**: 5 nm
- **alignment repro**: 5 nm
- **metrology stability**: 5 nm
- **stage overlay**: 12 nm
- **position accuracy**: 7 nm
- **frame stability**: 2.5 nm
- **system adjustment accuracy**: 2 nm
- **interferometer stability**: 1 nm
- **blue align sensor repro**: 3 nm
- **tracking error WS**: 2 nm
- **tracking error X, Y**: 2.5 nm
- **tracking error RS**: 1 nm
- **tracking error phi**: 75 nanoradians
- **off axis Sensor repro**: 3 nm
- **off axis pos. meas. accuracy**: 4 nm
- **blue align sensor**: 3 nm
- **alignment repro**: 3 nm
- **tracking error**: 1 nm
- **tracking error phi**: 75 nanoradians
## Budgets Applied on Medical Workstation Memory Use

### Table: memory budget in Mbytes

<table>
<thead>
<tr>
<th>System</th>
<th>code</th>
<th>obj data</th>
<th>bulk data</th>
<th>total</th>
</tr>
</thead>
<tbody>
<tr>
<td>shared code</td>
<td>11.0</td>
<td></td>
<td></td>
<td>11.0</td>
</tr>
<tr>
<td>User Interface process</td>
<td>0.3</td>
<td>3.0</td>
<td>12.0</td>
<td>15.3</td>
</tr>
<tr>
<td>database server</td>
<td>0.3</td>
<td>3.2</td>
<td>3.0</td>
<td>6.5</td>
</tr>
<tr>
<td>print server</td>
<td>0.3</td>
<td>1.2</td>
<td>9.0</td>
<td>10.5</td>
</tr>
<tr>
<td>optical storage server</td>
<td>0.3</td>
<td>2.0</td>
<td>1.0</td>
<td>3.3</td>
</tr>
<tr>
<td>communication server</td>
<td>0.3</td>
<td>2.0</td>
<td>4.0</td>
<td>6.3</td>
</tr>
<tr>
<td>UNIX commands</td>
<td>0.3</td>
<td>0.2</td>
<td>0</td>
<td>0.5</td>
</tr>
<tr>
<td>compute server</td>
<td>0.3</td>
<td>0.5</td>
<td>6.0</td>
<td>6.8</td>
</tr>
<tr>
<td>system monitor</td>
<td>0.3</td>
<td>0.5</td>
<td>0</td>
<td>0.8</td>
</tr>
<tr>
<td>application SW total</td>
<td>13.4</td>
<td>12.6</td>
<td>35.0</td>
<td>61.0</td>
</tr>
<tr>
<td>UNIX Solaris 2.x</td>
<td></td>
<td></td>
<td></td>
<td>10.0</td>
</tr>
<tr>
<td>file cache</td>
<td></td>
<td></td>
<td></td>
<td>3.0</td>
</tr>
</tbody>
</table>

| total                                       |      |          |           | 74.0  |
Power Budget Visualization for Document Handler

Modeling and Analysis: Budgeting
74  Gerrit Muller

version: 1.0
July 3, 2023
MDMpowerProportions
Evolution of Budget over Time

- fact finding through details
  - aggregate to end-to-end performance
  - search for appropriate abstraction level(s)

- from coarse guesstimate to reliable prediction
- from typical case to boundaries of requirement space
- from static understanding to dynamic understanding
- from steady state to initialization, state change and shut down
- from old system to prototype to actual implementation

\[ \text{time} \]

\[ \text{start} \quad \text{later} \quad \text{only if needed} \]
Potential Applications of Budget based design

- resource use (CPU, memory, disk, bus, network)
- timing (response, latency, start up, shutdown)
- productivity (throughput, reliability)
- Image Quality parameters (contrast, SNR, deformation, overlay, DOF)
- cost, space, time
What kind of budget is required?

<table>
<thead>
<tr>
<th>Static</th>
<th>Dynamic</th>
</tr>
</thead>
<tbody>
<tr>
<td>Typical case</td>
<td>Worst case</td>
</tr>
<tr>
<td>Global</td>
<td>Detailed</td>
</tr>
<tr>
<td>Approximate</td>
<td>Accurate</td>
</tr>
</tbody>
</table>

Is the budget based on wish, empirical data, extrapolation, educated guess, or expectation?
A budget is a quantified instantiation of a model

A budget can prescribe or describe the contributions by parts of the solution to the system quality under consideration

A budget uses a decomposition in tens of elements

The numbers are based on historic data, user needs, first principles and measurements

Budgets are based on models and estimations

Budget visualization is critical for communication

Budgeting requires an incremental process

Many types of budgets can be made; start simple!
The Boderc project contributed to Budget Based Design. Especially the work of

*Hennie Freriks, Peter van den Bosch (Océ),

*Heico Sandee and Maurice Heemels (TU/e, ESI)*

has been valuable.
Abstract

The word modeling is used for a wide variety of modeling approaches. These approaches differ in purpose, level of detail, effort, stakeholders, degree of formality, and tool support.
Human Thinking and Tools

- enterprise context
- enterprise
- stakeholders
- systems
- multi-disciplinary design
- parts, connections, lines of code

Number of details:
- $10^9$
- $10^6$
- $10^3$
- $10^0$
- $10^3$
- $10^6$
- $10^9$

Human overview

Tools to manage large amounts of information

e.g. Doors Core
Formality Levels in Pyramids

- **Less formal, communication-oriented**: heterogeneous uncertainties, unknowns, variable backgrounds, concerns
- **Well defined, repeatable, reusable, machine readable**: SysML, DOORS, IDEF0
- **More formal, more rigorous**: generated/instantiated

Levels:
- $10^0$
- $10^1$
- $10^2$
- $10^3$
- $10^4$
- $10^5$
- $10^6$
- $10^7$
- $10^8$
- $10^9$

The diagram illustrates the increasing formality as you move up the pyramid, with less formal, more rigorous levels at the bottom and more formal, more rigorous levels at the top.
<table>
<thead>
<tr>
<th>paradigm</th>
<th>purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>Conceptual system modeling</td>
<td>architecting understanding, evaluating, creating reasoning, communicating, decision making</td>
</tr>
<tr>
<td>SysML</td>
<td>formal capture of structure and behavior integrating other tools simulating</td>
</tr>
<tr>
<td>Design for 6 sigma</td>
<td>quality improvement in repeatable environments black box oriented</td>
</tr>
<tr>
<td>Conceptual information modeling</td>
<td>understanding and formalizing information</td>
</tr>
<tr>
<td>Design Framework</td>
<td>capturing and tracing architecture decisions</td>
</tr>
<tr>
<td>Matlab</td>
<td>modeling and analyzing designs and algorithms simulation and code generation</td>
</tr>
<tr>
<td>CAD</td>
<td>mechanical and electrical design interoperates with dedicated analysis, e.g. thermal, structural</td>
</tr>
<tr>
<td>Formal specification and design (model checkers)</td>
<td>verification</td>
</tr>
</tbody>
</table>
Abstract

Models are used for a wide variation of purposes. Stakeholders can get confused between "reality" and the virtual counterparts. In practice, many hybrids between "real" and virtual systems exist. For example, planning and training systems using real algorithms and data, and physical systems using a world model for situation awareness.
Model Applications and Variations

- Development
  - Understanding
  - Exploration
  - Optimization

- Verification
  - Test Data
  - Comparison
  - Trouble Shooting

- Operation
  - Mission Planning
  - Training
  - Health Monitoring

Systems of Systems apply all asynchronously

Evolvability
- All phases repeat with same needs

In system
- Situation Awareness
- Planning
- Training
- Health Monitoring
Spectrum from Real to Virtual Systems

“real” world

- system
  - subsystem
    - consisting of
      - hardware component
      - software component
  - stakeholders
    - mutually interacting

virtual world

- system
  - subsystem
    - consisting of
      - hardware component
      - software component
  - environment
    - data
  - stakeholders
    - mutually interacting

virtual world; HIL

- system
  - subsystem
    - consisting of
      - hardware component
      - software component
  - environment
    - data
  - stakeholders
    - mutually interacting

virtual world; SIL

- system
  - subsystem
    - consisting of
      - hardware component
      - software component
  - environment
    - data
  - stakeholders
    - mutually interacting

“real” world; testing

- system
  - subsystem
    - consisting of
      - hardware component
      - software component
  - environment
    - data
  - stakeholders
    - mutually interacting

simulation in context

- system
  - subsystem
    - consisting of
      - hardware component
      - software component
  - environment
    - data
  - stakeholders
    - mutually interacting
Architecting for Variations

variation dimensions
fidelity
product/system
performance
functionality
application
model purpose
exhaustiveness

properties
time-performance
accuracy
build & update effort
build & update time
testing effort and time
credibility
applicability
usability

impact

feed
iterate

system architecture
modularity
variation design

model architecture
modularity
variation design
Abstract

Models only get value when they are actively used. We will focus in this presentation on analysis aspects: accuracy, credibility, sensitivity, efficiency, robustness, reliability and scalability.
What Comes out of a Model

- varying inputs
- varying circumstances
- varying design options
- varying realizations
- specification changes
- and ripple through

Modeling and Analysis: Model Analysis
90 Gerrit Muller

version: 1.0
July 3, 2023
MAANaspects
Applicability of the Model

input
+ ε₁
- ε₂
accuracy
credibility

model(s)
accuracy
credibility
working range

model realization
credibility
propagation

abstraction
credibility
working range

usage context
specifications
designs
realizations

measurements
assumptions
facts

version: 1.0
July 3, 2023
MAANmodelApplicability
try out models
be aware of accuracy, credibility and working range

simple and small models

1. Estimate accuracy of results
   based on most significant inaccuracies of inputs and assumed model propagation behavior

2. Identify top 3 credibility risks
   identify biggest uncertainties in inputs, abstractions and realization

3. Identify relevant working range risks
   identify required (critical) working ranges and compare with model working range

substantial models
systematic analysis and documentation of accuracy, credibility and working range
Common Pitfalls

- discrete events in continuous world
  - discretization artefacts
    - e.g. stepwise simulations
- (too) systematic input data
  - random data show different behavior
    - e.g. memory fragmentation
- fragile model
  - small model change results in large shift in results
- self fulfilling prophecy
  - price erosions + cost increase (inflation) -> bankruptcy
Worst Case Questions

<table>
<thead>
<tr>
<th>Question</th>
<th>Response</th>
</tr>
</thead>
<tbody>
<tr>
<td>Which design assumptions have a big impact on system performance?</td>
<td></td>
</tr>
<tr>
<td>What are the worst cases for these assumptions?</td>
<td></td>
</tr>
<tr>
<td>How does the system behave in the worst case?</td>
<td></td>
</tr>
<tr>
<td>a. poor performance within spec</td>
<td></td>
</tr>
<tr>
<td>b. poor performance not within spec</td>
<td></td>
</tr>
<tr>
<td>c. failure -&gt; reliability issue</td>
<td></td>
</tr>
</tbody>
</table>
FMEA-like Analysis Techniques

- (systematic) brainstorm
- analysis and assessment
  - probability
  - severity
  - propagation
- improve
  - spec, design,
  - process,
  - procedure, ...

<table>
<thead>
<tr>
<th>safety</th>
<th>potential hazards</th>
<th>damage</th>
<th>measures</th>
</tr>
</thead>
<tbody>
<tr>
<td>hazard analysis</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>reliability</td>
<td>failure modes</td>
<td>effects</td>
<td>measures</td>
</tr>
<tr>
<td>FMEA</td>
<td>exceptional cases</td>
<td></td>
<td></td>
</tr>
<tr>
<td>security</td>
<td>vulnerability risks</td>
<td>consequences</td>
<td>measures</td>
</tr>
<tr>
<td>maintainability</td>
<td>change cases</td>
<td>impact, effort, time</td>
<td>decisions</td>
</tr>
<tr>
<td>performance</td>
<td>worst cases</td>
<td>system behavior</td>
<td>decisions</td>
</tr>
</tbody>
</table>
Brainstorming Phases

wave 1: the obvious

wave 2: more of the same

wave 3: the exotic, but potentially important

don't stop too early with brainstorming!
Different Viewpoints for Analysis

**usage context**
- new product
  - e.g. WoW extension
- merger
- automated access

**system**
- new functions
- new interfaces
- new media
- new standards
- cache/memory trashing
- garbage collection
- critical sections
- local peak loads
- intermittent HW failure

power failure
- network failure
- new SW release
- roll back to old SW release

**life cycle context**
Abstract

We make models to facilitate decision making. These decisions range from business decisions, such as Service Level Agreements, to requirements, and to detailed design decisions. The space of decisions is huge and heterogeneous. The proposed modeling approach is to use multiple small and simple models. In this paper we discuss how to reason by means of multiple models.
**Purpose of Modeling**

**How to use multiple models to facilitate decisions?**
**How to get from many fragments to integral insight?**
**How many models do we need?**
**At what quality and complexity levels?**

**Modeling and Analysis: Reasoning Approach**

- facts from investigation
- measurements
- assumptions
- modeling
- analysis
- results
- decision making
- specification
- verification
- decisions
- uncertainties
- unknowns
- errors
- accuracy
- working range
- credibility
- risk
- customer satisfaction
- time, cost, effort
- profit margin
Graph of Decisions and Models

usage context

enterprise & users

black box view

design

system

life cycle context

legend

- assumption
- input e.g. measurement
- decision
- model

Modeling and Analysis: Reasoning Approach

version: 1.0
July 3, 2023
Example Graph for Web Shop

Usage Context
- Enterprise & Users
  - Customer Interest
  - Customer Behavior
  - Personnel
  - Financial
  - Service Cost
  - Salary
  - Work Flow
  - Margin

Life Cycle Context
- SLA
  - Running Cost
  - Initial Cost
  - Maintenance Effort
  - Changes

Resource Dimensionsing
- Transaction Speed
- Transaction CPU
- CPU Budget
- Memory Budget
- Overhead
- Network Load
- Storage Capacity
- Access Time
- Response Time
- Throughput
- Information
- Elapsed Time

System
- Design
- Transactions
- CPU Load
- Network Load
- Transaction
- Storage Capacity

Legend
- Assumption
- Input e.g. Measurement
- Decision
- Model
Relations: Decisions, Models, Inputs and Assumptions

Modeling and Analysis: Reasoning Approach
102 Gerrit Muller
version: 1.0
July 3, 2023
MARErelations
Reasoning Approach

1. Explore usage context, life cycle context and system

2. Determine main Threads-of-Reasoning
3. Make main Threads-of-Reasoning SMART
4. Identify "hottest" issues
5. Model hottest, non-obvious, issues

2a. "Play" with models
2b. Investigate facts
2c. Identify assumptions

3. Model significant, non-obvious, issues

6. Capture overview, results and decisions

7. Iterate and validate

_all steps time-boxed between 1 hour and a few days_

early in project: 1. Explore usage context, life cycle context and system
ten days early in project: 1a. "Play" with models, investigate facts top-down

later in project: 6. Capture overview, results and decisions bottom-up: b3. Model significant, non-obvious, issues
Frequency of Assumptions, Decisions and Modeling

![Diagram showing the frequency of assumptions, decisions, and modeling. The categories are classified as implicit (trivial?), explicit, try-outs, very simple, small, key, and substantial. Each category is represented by a bar on a logarithmic scale.]

Legend:
- a: assumption
- i: input e.g.
- d: decision
- m: model

Modeling and Analysis: Reasoning Approach

version: 1.0
July 3, 2023
MAREfrequency
Life Cycle of Models

- **Understanding**
  - Try out models
  - Most try out models never leave the desk or computer of the architect!
  - Many small and simple models are used only once;
    - Some are re-used in next projects

- **Exploration**
  - Simple and small models
    - Re-used
    - Archived
    - Not maintained
  - Substantial models
    - Re-used
    - Archived
    - Not maintained

- **Optimization**
  - Archived
  - Re-used in next project

- **Verification**
  - Re-used in next project

Substantial models capture core domain know how; they evolve often from project to project.

Creation and evolution of intellectual property assets.
Examples of Life Cycle of Models

- **understanding**
- **exploration**
- **optimization**
- **verification**

- try out models
  - load/cost
  - function mix
  - load/cost peak impact

- simple and small models
  - customer global distribution

- substantial models (IP assets)
  - integral load model
  - global customer demographics

- web server performance
  - webshop benchmark suite
  - load/stress test suite
Abstract

Performance is a broad term. Each domain has its own key performance parameters. Performance can be used to indicate time-oriented performance, such as response time, throughput, or productivity. However, more broadly, it may be used for aspects like image quality, spatial performance (f.i. positioning accuracy), energy or power properties, sensitivity and specificity of algorithms, or reliability and availability.
Performance Attributes

- **time-oriented**
  - response time
  - latency
  - throughput
  - productivity

- **spatial**
  - positioning accuracy
  - working envelope
  - range
  - turning cycle

- **reliability**
  - MTBF
  - MTTR
  - uptime
  - unscheduled breaks

- **energy/power**
  - energy consumption
  - range
  - standby time
  - maximum power
  - heat release
  - cooling capacity

- **algorithmic**
  - sensitivity
  - specificity
  - accuracy
  - coverage

- **image quality**
  - sharpness
  - contrast
  - color consistency
  - color rendition
  - streakiness
  - uniformity
performance is a function of:

context

perception depends on individual human characteristics

circumstances \{ scenario

operation of interest use case \}

system of interest

specification generic, valid for the class of systems

design normal and special cases

configuration (worst case, degraded, exceptions, …)

version instance specific

history

\(^1\) a use case in this context is rich (includes quantifications) and broad (covers the operation of interest, not a single function)
Example EV Range Definition

Electric Vehicle Driving Range

Range = f(v(t), Circumstances, Driving style, Car load, Charging state, Battery age)

A quantified Use Case defines under what circumstances the EV will achieve the specified range.

New European Drive Cycle

Published under GFDL, thanks to Orzetto
The end-to-end performance is the relevant performance as the stakeholder experiences it: from initial trigger to final result.

\[ t_{\text{end-to-end}} = t_{\text{human activities}} + t_{\text{wait}} + t_{\text{elevator handling 1}} + t_{\text{move}} + t_{\text{elevator handling 2}} + t_{\text{human activities out}} \]
Abstract

Measuring is an essential part of architecting performance. Measurements provide quantified insight in actual behavior and performance. In this presentation, we discuss measuring, benchmarking, and instrumentation.
Performance Attributes in the Benchmark Stack

- Typical values
- Interference
- Variation
- Boundaries

- CPU
- Cache
- Memory
- Bus
  - (Computing) Hardware
  - Typical values
  - Interference
  - Variation
  - Boundaries

- Network transfer
- Database access
- Database query
- Services/Functions

- Duration
- Footprint
- Cache
- Interrupts
- Task switches
- OS services

- Duration
- Services
- Interrupts
- Task switches
- OS services

- End-to-end function
- Applications

- Operating system

- Operating system

- Locality
- Density
- Efficiency
- Overhead

- Tools

Architecting System Performance; Measuring

Gerrit Muller

version: 0
July 3, 2023
EBM/1benchmarkStack
Performance as Function of the Layers

system performance = f(hardware, operating system, services, applications, tools)

what is used?
how often?
how much does it cost?
Example $\mu$Benchmarks for Software

<table>
<thead>
<tr>
<th>Category</th>
<th>infrequent operations, often time-intensive</th>
<th>often repeated operations</th>
</tr>
</thead>
<tbody>
<tr>
<td>database</td>
<td>start session</td>
<td>perform transaction</td>
</tr>
<tr>
<td></td>
<td>finish session</td>
<td>query</td>
</tr>
<tr>
<td>network, I/O</td>
<td>open connection</td>
<td>transfer data</td>
</tr>
<tr>
<td></td>
<td>close connection</td>
<td></td>
</tr>
<tr>
<td>high level construction</td>
<td>component creation</td>
<td>method invocation</td>
</tr>
<tr>
<td></td>
<td>component destruction</td>
<td>same scope</td>
</tr>
<tr>
<td>low level construction</td>
<td>object creation</td>
<td>other context</td>
</tr>
<tr>
<td></td>
<td>object destruction</td>
<td></td>
</tr>
<tr>
<td>basic programming</td>
<td>memory allocation</td>
<td>method invocation</td>
</tr>
<tr>
<td></td>
<td>memory free</td>
<td></td>
</tr>
<tr>
<td>OS</td>
<td>task, thread creation</td>
<td>task switch</td>
</tr>
<tr>
<td></td>
<td></td>
<td>interrupt response</td>
</tr>
<tr>
<td>HW</td>
<td>power up, power down</td>
<td>cache flush</td>
</tr>
<tr>
<td></td>
<td>boot</td>
<td>low level data transfer</td>
</tr>
</tbody>
</table>
**Measurement Errors and Accuracy**

**Diagram:**
- **System under study** → measured signal → **Measurement instrument** → value
- **Noise** → **Resolution**
- **Offset** → **Calibration characteristics** → **Measurement error**

**Text:**

Measurements have stochastic variations and systematic deviations resulting in a range rather than a single value.
Be Aware of Error Propagation

\[ t_{\text{duration}} = t_{\text{end}} - t_{\text{start}} \]

\[ t_{\text{start}} = 10 \pm 2 \mu s \]

\[ t_{\text{end}} = 14 \pm 2 \mu s \]

\[ t_{\text{duration}} = 4 \pm ? \mu s \]

**systematic errors:** add linear

**stochastic errors:** add quadratic
Measurements have stochastic variations and systematic deviations resulting in a range rather than a single value.

The inputs of modeling, "facts", assumptions, and measurement results, also have stochastic variations and systematic deviations.

Stochastic variations and systematic deviations propagate (add, amplify or cancel) through the model resulting in an output range.
typical small testprogram

create steady state
t_s = timestamp()  
for(i=0;i<1M;i++) do something  
t_e = timestamp()  
duration = t_s - t_e
Abstract

The management of the resources largely determines system performance. This document discusses concepts related to resource management, such as caching, concurrency, and scheduling.
Generic Resource Model

- **Virtual**
  - Present
  - Process or compute
  - Communicate
  - Store
  - Acquire
  - Input

- **Physical**
  - Output
  - Process
  - Transport
  - Store
  - Raw material

Flow:
- Raw material
- Fetch
- Transport
- Process
- Transport
- Process
- Transport
- Deliver
- Product

Architecting System Performance; Resource Management

version: 0.1
July 3, 2023
ASPRMgenericResources
Performance depends on resource utilization and management.

The design of the logistics, how does EMI\(^1\) flow through the resources, is critical.

Critical design aspects are:

- concurrency (parallelism, pipelining)
- granularity of EMI
- scheduling (allocation of resources)

\(^1\)Energy Material Information
Granularity as Key Design Choice

- **unit of buffering**
  - ==
  - or
  - <=

- **unit of synchronization**
  - ==
  - or
  - <=

- **unit of processing**
  - ==
  - or
  - <=

- **unit of I/O**
  - ==
  - or
  - <=

- **Video frame**
  - **Video line**
  - **Pixel**

- **fine grain**: flexible, high overhead
- **coarse grain**: rigid, low overhead
Size versus Performance Trade off

<table>
<thead>
<tr>
<th>small capacity</th>
</tr>
</thead>
<tbody>
<tr>
<td>fast technology</td>
</tr>
<tr>
<td>small</td>
</tr>
<tr>
<td>expensive</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>large capacity</th>
</tr>
</thead>
<tbody>
<tr>
<td>slow technology</td>
</tr>
<tr>
<td>large</td>
</tr>
<tr>
<td>low cost</td>
</tr>
</tbody>
</table>

staircase effect: performance and size are non-linear with thresholds

![Graph showing performance and data set size]

random data processing performance in ops/s

data set size in bytes

L1 cache
L3 cache
main memory
hard disk
disk farm
robotized media

example data storage technology

Architecting System Performance; Resource Management

version: 0.1
July 3, 2023

Gerrit Muller
Pipeline pattern

production line = pipeline

lean uses the notion of tact
f.i. every 10 minutes the products
move to the next workspot

throughput = products/time

$ t_{production\ 1\ car} = t_{in} - t_{out} $
Y-chart Pattern

- Structure and topology of resources
- Design of dynamic behavior
- System performance

Feedback

Mapping
Performance Pitfalls and Resource Management

- Overhead (control, handling)
- Starvation (underrun)
- Saturation/stagnation (overrun)
- Variation (duration, quality)
- Serialization
- Interference with other work
- Unnecessary conversions or adaptations
Abstract

Greedy and lazy are two opposite patterns in performance design. An extreme application of both patterns is start-up, where greedy starts as much as possible, and lazy as little as possible.
<table>
<thead>
<tr>
<th>lazy (on demand, pull)</th>
<th>greedy (push, forecast)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>what</strong></td>
<td></td>
</tr>
<tr>
<td>do nothing until someone needs it</td>
<td>prepare time consuming operations, when resources are idle</td>
</tr>
<tr>
<td><strong>benefits</strong></td>
<td></td>
</tr>
<tr>
<td>no resource usage unless needed</td>
<td>results are available immediately</td>
</tr>
<tr>
<td><strong>disadvantages</strong></td>
<td></td>
</tr>
<tr>
<td>time to result depends on execution time</td>
<td>some resource use is wasted</td>
</tr>
<tr>
<td><strong>when</strong></td>
<td></td>
</tr>
<tr>
<td>default</td>
<td>to achieve required performance (explore other concepts too!)</td>
</tr>
</tbody>
</table>

this pattern applies to all domains (IT, goods flow, energy)
Start up of Systems as Example

- initial
- start system
- running
- start operation
- operating

How much time does it take to start a laptop with Windows?

How much time does it take to start an application (e.g. Word)?
Example from Cloud Applications
<table>
<thead>
<tr>
<th>performance issues</th>
<th>solution patterns</th>
<th>design parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>long latency (mass) storage</td>
<td>frequently used subset in fast local storage</td>
<td>caching algorithm</td>
</tr>
<tr>
<td>long latency communication</td>
<td>low latency</td>
<td>storage location</td>
</tr>
<tr>
<td>overhead communication</td>
<td>less communication</td>
<td>cache size</td>
</tr>
<tr>
<td>resource intensive processing</td>
<td>large chunks (less overhead)</td>
<td>chunk size</td>
</tr>
<tr>
<td></td>
<td>processing once (keep results)</td>
<td>format</td>
</tr>
</tbody>
</table>
### Many Layers of Caching

<table>
<thead>
<tr>
<th>Cache Type</th>
<th>Cache Miss Penalty</th>
<th>Cache Hit Performance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application Cache</td>
<td>1 s</td>
<td>10 ms</td>
</tr>
<tr>
<td>Network Layer Cache</td>
<td>100 ms</td>
<td>1 ms</td>
</tr>
<tr>
<td>File Cache</td>
<td>10 ms</td>
<td>10 µs</td>
</tr>
<tr>
<td>Virtual Memory</td>
<td>1 ms</td>
<td>100 ns</td>
</tr>
<tr>
<td>Memory Caches L1, L2, L3</td>
<td>100 ns</td>
<td>1 ns</td>
</tr>
</tbody>
</table>

**Typical Cache 2 Orders of Magnitude Faster**

**Diagram:**
- **Client** connected to **Screen**
- **Network** layer cache connected to **File Cache** and **Application Cache**
- **Virtual Memory** connected to **Memory Caches L1, L2, L3**
- **Back Office Server** connected to **Network**
- **Mid Office Server** connected to **Network**

**Network Layer Cache**
- Connected to **File Cache**
- Connected to **Application Cache**

**File Cache**
- Connected to **Application Cache**

**Application Cache**
- Connected to **Virtual Memory**
- Connected to **Memory Caches L1, L2, L3**

**Virtual Memory**
- Connected to **Memory Caches L1, L2, L3**

**Memory Caches L1, L2, L3**
- Connected to **Virtual Memory**
- Connected to **Back Office Server**
- Connected to **Mid Office Server**

**Back Office Server**
- Connected to **Network**
- Connected to **Mid Office Server**
Disadvantages of Caching Pattern

- robustness for application changes
- ability to benefit from technology improvements
- robustness for changing context (e.g. scalability)
- robustness for concurrent applications
- failure modes in exceptional user space

These patterns increase **complexity** and **coupling**.

Use only when necessary for performance.
Abstract

Scheduling plays a crucial role in resource allocation to get desired system performance. This document discusses local and global scheduling.
Scheduling of time critical operations on a single resource:

- Earliest Deadline First
  - optimal
  - complex to realize

- Rate Monotonic Scheduling
  - no full utilization
  - simple to realize
### Earliest Deadline First

<table>
<thead>
<tr>
<th>Constraints</th>
<th>Determine deadlines in <strong>Absolute time</strong> (CPU cycles or msec, etc.)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Assign priorities Process that has the earliest deadline gets the highest priority (no need to look at other processes)</td>
</tr>
<tr>
<td></td>
<td>Constraints Smart mechanism needed for Real-Time determination of deadlines Pre-emptive scheduling needed</td>
</tr>
</tbody>
</table>

**EDF = Earliest Deadline First**

Earliest Deadline based scheduling for (a-)periodic Processing

The theoretical limit for any number of processes is 100% and so the system is schedulable.
Exercise Earliest Deadline First (EDF)

Calculate loads and determine thread activity (EDF)

<table>
<thead>
<tr>
<th>Thread</th>
<th>Period = deadline</th>
<th>Processing</th>
<th>Load</th>
</tr>
</thead>
<tbody>
<tr>
<td>Thread 1</td>
<td>9</td>
<td>3</td>
<td>33.3%</td>
</tr>
<tr>
<td>Thread 2</td>
<td>15</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>Thread 3</td>
<td>23</td>
<td>5</td>
<td></td>
</tr>
</tbody>
</table>

Suppose at \( t=0 \), all threads are ready to process the arrived trigger.

Source: Ton Kostelijk - EXARCH course
Rate Monotonic Scheduling

- Determine deadlines (period) in terms of Frequency or Period (1/F)
- Assign priorities
  - Highest frequency (shortest period) ==> Highest priority
- Constraints
  - Independent activities
  - Periodic
  - Constant CPU cycle consumption
  - Assumes Pre-emptive scheduling

RMS = Rate Monotonic Scheduling

Priority based scheduling for Periodic Processing of tasks with a guaranteed CPU - load
Exercise Rate Monotonic Scheduling (RMS)

Calculate loads and determine thread activity (RMS)

<table>
<thead>
<tr>
<th>Thread</th>
<th>Period = deadline</th>
<th>Processing</th>
<th>Load</th>
</tr>
</thead>
<tbody>
<tr>
<td>Thread 1</td>
<td>9</td>
<td>3</td>
<td>33.3%</td>
</tr>
<tr>
<td>Thread 2</td>
<td>15</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>Thread 3</td>
<td>23</td>
<td>5</td>
<td></td>
</tr>
</tbody>
</table>

Suppose at t=0, all threads are ready to process the arrived trigger.

Source: Ton Kostelijk - EXARCH course
**RMS-RMA Theory**

<table>
<thead>
<tr>
<th>Assumptions Rate Monotonic Analysis (RMA):</th>
<th>RMA theory: Schedule is possible when:</th>
</tr>
</thead>
<tbody>
<tr>
<td>periodic tasks with period $T_i$</td>
<td>Load $= \sum_i U_i \leq n(2^{1/n} - 1)$ for $n = 1, 2, 3, \infty$</td>
</tr>
<tr>
<td>process time $P_i$</td>
<td>max utilization is:</td>
</tr>
<tr>
<td>load $U_i = P_i/T_i$</td>
<td>1.00, 0.83, 0.78, … log(2)</td>
</tr>
<tr>
<td>tasks are independent</td>
<td>$\sim= 0.69$</td>
</tr>
</tbody>
</table>

Rate Monotonic Scheduling (RMS) uses fixed priorities  
RMS guarantees that all processes meet their deadlines  
Fixed priority -> low overhead

Source: Ton Kostelijk - EXARCH course
Answer EDF Exercise

Answers: loads and thread activity (EDF)

<table>
<thead>
<tr>
<th>Thread</th>
<th>Period = deadline</th>
<th>Processing</th>
<th>Load</th>
</tr>
</thead>
<tbody>
<tr>
<td>Thread 1</td>
<td>9</td>
<td>3</td>
<td>33.3%</td>
</tr>
<tr>
<td>Thread 2</td>
<td>15</td>
<td>5</td>
<td>33.3%</td>
</tr>
<tr>
<td>Thread 3</td>
<td>23</td>
<td>5</td>
<td>21.7%</td>
</tr>
</tbody>
</table>

88.3%

Source: Ton Kostelijk - EXARCH course
Answer RMS Exercise

Answers: loads and thread activity (RMS)

<table>
<thead>
<tr>
<th>Thread</th>
<th>Period = deadline</th>
<th>Processing</th>
<th>Load</th>
</tr>
</thead>
<tbody>
<tr>
<td>Thread 1</td>
<td>9</td>
<td>3</td>
<td>33.3%</td>
</tr>
<tr>
<td>Thread 2</td>
<td>15</td>
<td>5</td>
<td>33.3%</td>
</tr>
<tr>
<td>Thread 3</td>
<td>23</td>
<td>5</td>
<td>21.7%</td>
</tr>
</tbody>
</table>

88.3%

Source: Ton Kostelijk - EXARCH course

Architecting System Performance; Scheduling
Gerrit Muller
version: 0
July 3, 2023
PHRTexerciseRMSanswer
A **perspective** on dynamic behavior is to view the system as a set of **periodic behaviors**.

**Periodic behavior** is easier to **model** and **analyze**, e.g., using RMS and RMA.

Modern systems and Systems of Systems consist of complex **networks of concurrent resources**.

Typically, a combination of more advanced **global** scheduling is combined with simple **local** scheduling.
Abstract

Performance should be robust. The performance should be reproducible and it should be well-behaving in extreme conditions.
Variations are Suspect

Poorly understood variations require analysis.

- Performance measurements in standardized conditions
- What causes these outlyers?
- What causes this variation?
- Smaller is better

Architecting System Performance; Robust Performance
146 Gerrit Muller

version: 0.2
July 3, 2023
ASPRPvariation
Coping with Disturbances

How does the system respond to disturbances?
How quickly does it recover?
How far does performance degrade?

steady state performance
degradation
recovery time
disturbance

Architecting System Performance; Robust Performance
147 Gerrit Muller
A system design assumption is often:
the performance of this function
{ is constant | is linear | doesn't exceed x | ...}

The working range is the interval where this assumption holds
## Validate Understanding of System Performance

| Characterize the system | use the system in varying conditions  
|                        | measure performance  
|                        | as function of the conditions |
| Stress testing         | where does the design fail?  
|                        | (go beyond specified limits) |
| Load testing           | keep the system in heavy load condition  
|                        | observe how it keeps performing  
|                        | measure variations |
| (Accelerated) Lifetime testing | age the system  
|                        | observe how it keeps performing |
Abstract

A threat to performance is the combination of feature creep and technical debt. This combination causes bloating of the design. In Lean terms, the combination causes waste. A crucial question is where is the value, and is the value in balance with the potential degradation of performance.
From Feature Creep to Performance Problems

- maturing of systems
- increasing number of features (feature creep)
- technical debt
- time effort gain by taking shortcuts
- bloating of design
- increase in resource usage
- design complexity
- lack of overview insight understanding
- performance problems

loss of knowledge
Technical Debt is a metaphor used within the software industry to communicate the consequences of pragmatic design decisions deviating from the intended design of a system.

from: http://gaudisite.nl/INCOSE2016_Callister_Andersson_SMARTtechnicalDebt.pdf
Based on Cunningham http://c2.com/doc/oopsla92.html
Are benefits (value) in balance with the costs (such as performance degradation)?

Increasing number of features (feature creep)

Time effort gain by taking shortcuts

Technical debt

Increase in resource usage

Performance problems

Bloating of design

Design complexity

Lack of overview insight understanding

Value versus Performance Degradation
Exploring bloating: main causes

<table>
<thead>
<tr>
<th>poor specification (&quot;what&quot;)</th>
<th>poor design (&quot;how&quot;)</th>
<th>genericity, configurability, provisions for future</th>
<th>dogmatic rules (for instance fine grain COM interfaces)</th>
</tr>
</thead>
<tbody>
<tr>
<td>support for unused legacy code</td>
<td>core function</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Legend:**
- **Overhead**
- **Value**
Necessary functionality \(\gg\) the intended regular function
The danger of being generic: bloating

If-then-else lots of configuration options lots of stubs lots of best guess defaults over-generic class

client side
generic design from scratch

lots of config over-rides
lots of config over-rides
lots of config over-rides

toolbox side

lots of if-then-else

in retrospect common (duplicated) code

specific implementations without a priori re-use

after refactoring

"Real-life" example: redesigned Tool super-class and descendants, ca 1994
Shit propagation via copy paste

needed code

bad code

copy paste modify

needed code

code not relevant for new function

repair code

bad code

new needed code

new bad code
Example of shit propagation

Class Old:
    capacity = startCapacity
    values = int(capacity)
    size = 0

def insert(val):
    values[size]=val
    size+=1
    if size>capacity:
        capacity*=2
        relocate(values, capacity)

Class New:
    capacity = 1
    values = int(capacity)
    size = 0

def insert(val):
    values[size]=val
    size+=1
    capacity+=1
    relocate(values, capacity)

def insertBlock(v,len):
    for i=1 to len:
        insert(v[i])

Class DoubleNew:
    capacity = 1
    values = int(capacity)
    size = 0

def insert(val):
    values[size]=val
    size+=1
    capacity+=1
    relocate(values, capacity)

    def insertBlock(v,len):
        for i=1 to len:
            insert(v[i])
Bloating causes more bloating

- Poor specification ("what")
  - Support for unused legacy code
- Poor design ("how")
  - Support for unused legacy code
  - Genericity
  - Configurability
  - Provisions for future

- Dogmatic rules
  - Support for unused legacy code

Legend:
- Overhead
- Value
Causes even more bloating...

Bloating causes performance and resource problems. Solution: special measures: memory pools, shortcuts, ...

Bloating, Waste, and Value
160 Gerrit Muller

EASRTbloatingCausesBloatingMore