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.

October 20, 2017
status: preliminary
draft
version: 0.4
Abstract

Course overview of the course Architecting System Performance.
1. Course introduction
2. Managing system performance
3. Course didactics
4. Connecting breadth and depth
5. Performance Modeling
6. Level of Abstraction
7. Visualizing Dynamic Behavior
8. Emerging Behaviour
9. Budgeting
10. Modeling Paradigms
11. Applications and Variations
12. Model Analysis
13. Reasoning Approach
14. Defining Performance
15. Measuring Performance
16. Resource Management
17. Greedy and Lazy Pattern
18. Scheduling
19. Robust Performance
20. Bloating, Waste, and Value

**time-oriented performance**
### Assignments in Face-to-Face Module

<table>
<thead>
<tr>
<th>0. elevator case</th>
</tr>
</thead>
<tbody>
<tr>
<td>supersystem</td>
</tr>
<tr>
<td>1. sketch the problem goal use case key performance parameters main concepts critical technologies</td>
</tr>
<tr>
<td>2. make conceptual model of the current situation</td>
</tr>
<tr>
<td>• model dynamic behavior</td>
</tr>
<tr>
<td>• model 0-order kpp using functions (as simple as possible)</td>
</tr>
<tr>
<td>• quantify contribution to kpp using observed data</td>
</tr>
<tr>
<td>3. explore customer and business relevance</td>
</tr>
<tr>
<td>• develop story</td>
</tr>
<tr>
<td>• model workflow and performance</td>
</tr>
<tr>
<td>• model customer value as function of kpp</td>
</tr>
<tr>
<td>4. make conceptual model of potential solutions</td>
</tr>
<tr>
<td>• model the foreseen solution</td>
</tr>
<tr>
<td>• model &amp; compare 2 alternative solutions</td>
</tr>
<tr>
<td>5. list questions and uncertainties, reformulate problem and goal, and formulate gaps and options</td>
</tr>
<tr>
<td>6. develop an elevator pitch to report you findings and recommendations to management</td>
</tr>
</tbody>
</table>
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.
### core

**Physical Models of an Elevator**

[http://www.gaudisite.nl/info/ElevatorPhysicalModel.info.html](http://www.gaudisite.nl/info/ElevatorPhysicalModel.info.html)

---

### optional

**Teaching conceptual modeling at multiple system levels using multiple views**


**Understanding the human factor by making understandable visualizations**

[http://www.gaudisite.nl/info/UnderstandingHumanFactorVisualizations.info.html](http://www.gaudisite.nl/info/UnderstandingHumanFactorVisualizations.info.html)
<table>
<thead>
<tr>
<th><strong>core</strong></th>
<th>Architecting System Performance; Course Didactics</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<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><strong>optional</strong></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Assumptions</strong>:</td>
<td>“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><strong>70/20/10</strong>:</td>
<td></td>
</tr>
<tr>
<td></td>
<td><a href="http://charles-jennings.blogspot.nl/">http://charles-jennings.blogspot.nl/</a></td>
</tr>
<tr>
<td><strong>Assumptions and beliefs</strong>:</td>
<td></td>
</tr>
<tr>
<td>section</td>
<td>title</td>
</tr>
<tr>
<td>-----------------</td>
<td>----------------------------------------------------------------------</td>
</tr>
<tr>
<td>core</td>
<td>Architecting System Performance; Greedy and Lazy Patterns</td>
</tr>
<tr>
<td><strong>core</strong></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td></td>
</tr>
<tr>
<td>Architecting System Performance; Measuring</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>optional</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Performance Method Fundamentals</td>
</tr>
<tr>
<td>Measurement issues; From gathering numbers to gathering knowledge by Ton Kostelijk <a href="http://www.gaudisite.nl/MeasurementExecArchSlides.pdf">http://www.gaudisite.nl/MeasurementExecArchSlides.pdf</a></td>
</tr>
<tr>
<td>Modeling and Analysis: Measuring</td>
</tr>
<tr>
<td>Exploring an existing code base: measurements and instrumentation</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

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?

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
Modeling Evolves over Time

- Understanding
- Exploration
- Optimization
- Verification

Project phase determines purpose of the model, which determines the 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?

“look ahead”

mindset of most stakeholders
mindset of architect
10 Fundamental Recommendations

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

**objectives**
- support communication
- facilitate reasoning
- support decision making
- create understanding
- maintain 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
Iterative Performance Management during Development

- measure
- evaluate
- analyse

- determine most important and critical requirements

- simulate
- build proto

- model
- analyse constraints and design options
Managing Performance during Product Development

- measurement
- design estimate and uncertainty
- specification

- incomplete understanding
- design robustness problem
- finished product
- degrading performance

- calibration input

- worse
- better

- 1000
- 100

- time
Quantification Steps

- Back of the envelope
- Benchmark, spreadsheet calculation
- Measure, analyze, simulate
- Cycle accurate

- Order of magnitude
- Guestimates
- Calibrated estimates
- Feasibility measure, analyze, simulate
- Back of the envelope benchmark, spreadsheet calculation measure, analyze, simulate cycle accurate

Architecting System Performance; Managing System Performance
Version: 0.2
October 20, 2017
BWMquantificationSteps
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

- Remembering
- Understanding
- Applying
- Analyzing
- Evaluating
- Creating

Higher Order Thinking Skills are more difficult to teach but more valuable. They take time to develop.

Lower Order Thinking Skills, on the other hand, people can acquire fast, and they must be mastered before higher order skills can be acquired.
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**

Organizing **Systems**

Recognizing **Relationships**

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

reflection wall
Scope and Topic of Reflection

operational or life cycle context
organization
principle
process or method
procedure or technique
tool or notation

system of interest
component or function of interest
project
team
individual

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/

Architecting System Performance; Course Didactics

version: 0.1
October 20, 2017
ASPCDladderOfInference
Architecting System Performance; Connecting Breadth and Depth

by Gerrit Muller TNO-ESI, University College of South East Norway
e-mail: gaudisite@gmail.com
www.gaudisite.nl

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.

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.
Organizational Problem: Disconnect

What does Customer need in Product and Why?

How can the product be realized
What are the critical decisions

Customer objectives
Application
Functional
Conceptual
Realisation

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

Architecting System Performance; Connecting Breadth and Depth

version: 0
October 20, 2017
RATWdisconnect
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

Architecting System Performance; Connecting Breadth and Depth

version: 0
October 20, 2017
RATWbreadthAndDepth

number of details

system requirements

design decisions

parts
connections
lines of code

and growing every year...
Major Bottleneck: Mental Dynamic Range

Architecting System Performance; Connecting Breadth and Depth
Gerrit Muller
version: 0
October 20, 2017
RATWmentalDynamicRange
Breadth

- Stakeholders
  - Concerns
  - Needs
  - Interests

- Supporting systems
  - Train
  - Plan
  - Maintain

- Surrounding systems
  - Supply
  - Receive
  - Manage

- System of Interest
  - Regulations
  - Processes
  - Procedures

Architecting System Performance; Connecting Breadth and Depth

version: 0
October 20, 2017
CBADbreadth
Devilish details in design space may have large impact on performance. Many detailed design decisions determine system performance.

**Design space**
- Resource management
- Process, transport, store, in/out
- Internal logistics
  - Concurrency, processes
- Processing
  - Algorithms, machining, ...

**System performance**
Abstract

Principles and concepts of modeling performance.
Empirical versus First Principle Models

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.
**Conceptual model**: a model explaining observations and measurements using some 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\textsuperscript{th}</td>
<td>main function, main parameters</td>
<td>[ t_{\text{top floor}} = \frac{S_{\text{top floor}}}{v_{\text{max}}} ]</td>
</tr>
<tr>
<td>1\textsuperscript{st}</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}}} ]</td>
</tr>
<tr>
<td>2\textsuperscript{nd}</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

Architecting System Performance; Level of Abstraction

Gerrit Muller

version: 0
October 20, 2017
RAPpyramid
From system to Product Family or Portfolio

Architecting System Performance; Level of Abstraction
Gerrit Muller

version: 0
October 20, 2017
DRALpyramidGrowth
Product Family in Context

- Parts, connections, lines of code
- Multidisciplinary design
- Systems
- Stakeholders
- Enterprise
- Enterprise context

Number of details:

- $10^9$
- $10^6$
- $10^3$
- $10^0$
- $10^3$
- $10^6$
- $10^9$
The seemingly random exploration path

Architecting System Performance; Level of Abstraction
Gerrit Muller

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 definition, elaborated use cases, performance models, budgets and measurements, component designs, killing details.

Levels:
- Monodisciplinary
- Multidisciplinary
- Systems

Number of details:
- 10^0
- 10^1
- 10^2
- 10^3
- 10^4
- 10^5
- 10^6
- 10^7
- 10^8
- 10^9

Elaboration, design and engineering, simplification, abstraction.
Fidelity Properties

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

systems
stakeholders
enterprise
enterprise context
multidisciplinary
monodisciplinary

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

Architecting System Performance; Level of Abstraction
Gerrit Muller
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

Concrete “Cartoon” Workflow

Timeline of Workflow

Swimming Lanes

Concurrency and Interaction

Abstract Workflow

Timeline and Functional Flow

Signal Waveforms

Flow of Light

Information Centric Processing Diagram

Visualizing Dynamic Behavior

version: 0
October 20, 2017
VDBOverview

Visualizing Dynamic Behavior

Gerrit Muller

100
22
5
12
14
25
12
72
11
200
18
21
12
7
inter
resume
reset
move away from
22
6
pulse
Gy
Gx
Gz
wavelength
print
robot
prealign
clean
master
prefill
clean wafer
0 100b 200b
run EDP/LRP
hook up coiled tubing/wireline
function and seal test
run coiled tubing/wireline
assembly and test
run... production resume
production
deferred operation 62 hrs
move above well
move away from well
ROV assisted connect
Example Functional Model of Information Flow

1. **Get sensor data**
2. **Transform into image**
3. **Fuse sensor images**
4. **Detect objects**
5. **Classify objects**
6. **Update world model**
7. **Analyze situation**
8. **Determine next step**

- **Get GPS data**
- **Calculate GPS location**
- **Update GPS location**

- **Get v, a**
- **Estimate location**

**Objects**

**Location**

**World model**
"Cartoon" Workflow
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. hook up coil tubing and wireline BOP
8. system function and connection seal test
9. run coil tubing and wireline
10. perform workover operations
11. retrieve coil tubing and wireline BOP
12. unhook coil tubing and wireline BOP
13. ROV assisted disconnect
14. move away from well
15. retrieve SFT and TF
16. retrieve risers
17. retrieve EDP/LRP
18. disassembly
19. ROV assisted connect
20. ROV assisted disconnect
21. retrieve SFT and TF
22. retrieve risers
23. retrieve EDP/LRP
24. disassembly
25. move above well
26. move away from well
27. ROV assisted connect
28. ROV assisted disconnect
29. retrieve SFT and TF
30. retrieve risers
31. retrieve EDP/LRP
32. 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
Swimming Lane Example
Example Signal Waveforms

imaging = repeating similar pattern many times

g_y=0

Gy=127

Typical TE: 5..50ms

Transmit

Receive

Visualizing Dynamic Behavior

Gerrit Muller
Example Time Line with Functional Model

functional flow

- 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

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

Visualizing Dynamic Behavior

Gerrit Muller

version: 0
October 20, 2017
MREndToEndTimeline
Information Centric Processing Diagram

- raw image → enhance → resized image → grey-value image → view-port
- retrieve → enhance → interpolate → lookup → merge → display
- text, gfx
Example State Diagram

- **idle**
  - start → **operating**
  - event → **pre-alarm mode**
  - acknowledge → **alarm mode**
- **reset** → **alarm handled**
Flow of Light (Physics)

laser
pulse-freq, bw, wavelength, ..

illuminator
uniformity

sensor

reticle

lens

wafer

aerial image

NA
abberations
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
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”.

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.
emergent behavior and properties =

function of

**dynamic interaction** between

**parts** in the system and

**context** of the system

text in the system and context of the system

text 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

Goal of design
Side-effect
Undesired
Desired

Modeling and Analysis: Emerging Behavior
Gerrit Muller
October 20, 2017
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.
content of this presentation

What and why of a budget

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

How to use a budget
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
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

**Model**
- \( t_{proc} \)
- \( t_{over} \)
- \( t_{disp} \)
- \( t_{over} \)

**Measurements**
- Existing system

**Micro Benchmarks**
- Aggregated functions
- Applications

**Simulations**
- \( V_{4aa} \)

**Estimates**
- SRS
  - \( t_{boot} = 0.5s \)
  - \( t_{cap} = 0.2s \)

**Budget**
- \( t_{proc} = 10 \)
- \( t_{over} = 20 \)
- \( T_{proc} = 30 \)
- \( t_{disp} = 5 \)
- \( t_{over} = 20 \)
- \( T_{disp} = 25 \)
- \( T_{total} = 55 \)

**Feedback**
- Tuning

**Form**
- Measurements
- New (proto) system

**Can be more complex than additions**

**Modeling and Analysis: Budgeting**

---

**Version:** 1.0

October 20, 2017

Gerrit Muller
### 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</td>
</tr>
<tr>
<td></td>
<td>measurements and estimates provide initial numbers</td>
</tr>
<tr>
<td></td>
<td>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 Medical Workstation Memory Use

<table>
<thead>
<tr>
<th>memory budget in Mbytes</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>
<tr>
<td>total</td>
<td></td>
<td></td>
<td></td>
<td>74.0</td>
</tr>
</tbody>
</table>
Power Budget Visualization for Document Handler

Modeling and Analysis: Budgeting
Gerrit Muller

version: 1.0
October 20, 2017
MDMpowerProportions
Alternative Power Visualization

- Power supplies
- Cooling
- UI and control
- Paper path
- Paper input module
- Finisher
- Procedé
- Paper

- Electrical power

Heat
### Evolution of Budget over Time

<table>
<thead>
<tr>
<th>Event</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>fact finding through details</td>
<td>start</td>
</tr>
<tr>
<td>aggregate to end-to-end performance</td>
<td>later</td>
</tr>
<tr>
<td>search for appropriate abstraction level(s)</td>
<td>only if needed</td>
</tr>
<tr>
<td>from coarse guesstimate to reliable prediction</td>
<td></td>
</tr>
<tr>
<td>from typical case to boundaries of requirement space</td>
<td></td>
</tr>
<tr>
<td>from static understanding to dynamic understanding</td>
<td></td>
</tr>
<tr>
<td>from steady state to initialization, state change and shut down</td>
<td></td>
</tr>
<tr>
<td>from old system to prototype to actual implementation</td>
<td></td>
</tr>
</tbody>
</table>

*Modeling and Analysis: Budgeting*

76 Gerrit Muller
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

- **number of details**: 10^9, 10^6, 10^3, 10^0, 10^3, 10^6, 10^9

- **enterprise context**

- **enterprise**

- **stakeholders**

- **systems**

- **multi-disciplinary design**

- **parts, connections, lines of code**

- **tools to manage large amounts of information**

- **human overview**

  - e.g. Doors Core

Modeling and Analysis; Modeling Paradigms
82  Gerrit Muller
version: 0  
October 20, 2017  
KDAWToolsDiabolo
Formality Levels in Pyramids

- More formal, more rigorous
- Well defined, repeatable, reusable
- Communication-oriented
- Less formal, uncertainties, unknowns, variable backgrounds, concerns

Number of details:

0
10
100
101
102
103
104
105
106
107

System
Multi-disciplinary
Mono-disciplinary
Generated/instantiated

Modeling and Analysis; Modeling Paradigms
83  Gerrit Muller
### Modeling Paradigms

<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

- sales
- acquisition
- capability analysis
- development
- verification
- validation
- operation
- understanding
- exploration
- optimization
- test data
- comparison
- trouble shooting
- 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

stakeholders

system

mutually interacting

system

subsystem

consisting of

hardware component

software component

data

environment

virtual world

stakeholders

agents

mutually interacting

system

subsystem

consisting of

hardware component

software component

virtual world; HIL

stakeholders

agents

mutually interacting

system

subsystem

consisting of

hardware component

software component

virtual world; SIL

stakeholders

agents

mutually interacting

system

subsystem

consisting of

hardware component

software component

“real” world; testing

stakeholders

mutually interacting

system

subsystem

consisting of

hardware component

software component

simulation in context

stakeholders

mutually interacting

system

subsystem

consisting of

hardware component

software component
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

system architecture
- modularity
- variation design

model architecture
- modularity
- variation design

iterate
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

- working range
- worst case behavior
- exceptional behavior

- accuracy
- credibility
- working range

- sensitivity
- robustness
- efficiency

- performance
- reliability
- scalability
- other system qualities

specification feasibility

design quality

model applicability

life cycle

- design
- understanding
- exploration
- optimization
- verification

use cases
- worst case
- exceptions

varying inputs
- varying circumstances

varying design options
- varying realizations

change cases

specification changes
- and ripple through
Applicability of the Model

Modeling and Analysis: Model Analysis
91    Gerrit Muller

version: 1.0
October 20, 2017
MAANmodelApplicability

+ε₁
-ε₂

input
accuracy
credibility

model(s)

accuracy
credibility
working range

model realization
credibility
propagation

abstraction
credibility
working range

usage context
specifications
designs
realizations

measurements
assumptions
facts
How to Determine Applicability

**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
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

Which design assumptions have a big impact on system performance?

What are the worst cases for these assumptions?

How does the system behave in the worst case?

a. poor performance within spec

b. poor performance not within spec

c. failure -> reliability issue
# FMEA-like Analysis Techniques

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

<table>
<thead>
<tr>
<th>Category</th>
<th>Analysis</th>
<th>Effect</th>
<th>Improvement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Safety hazard</td>
<td>potential hazards</td>
<td>damage</td>
<td>measures</td>
</tr>
<tr>
<td>Reliability</td>
<td>failure modes</td>
<td>effects</td>
<td>measures</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

life cycle context
- power failure
- network failure
- new SW release
- roll back to old SW release
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.
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?
Graph of Decisions and Models

usage context
enterprise & users

black box view

system
design

life cycle context

legend
- assumption
- input e.g. measurement
- decision
- model
Relations: Decisions, Models, Inputs and Assumptions

- **Legend**
  - ▲ assumption
  - 📊 input e.g. measurement
  - 🗒️ decision
  - 📀 model

The diagram illustrates the relationships between decisions (d), models (m), inputs (i), and assumptions (a).

- **Decision (d)** feeds to **Model (m)**, which in turn triggers **Input (i)**.
- **Model (m)** also feeds to **Input (i)**.
- **Assumption (a)** facilitates **Decision (d)**.
- **Input (i)** triggers **Assumption (a)**.
- **Decision (d)** calibrates **Model (m)**.
- **Model (m)** influences **Decision (d)**.

The diagram captures the flow of information and influence between these key elements.
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

6. Capture overview, results and decisions

7. Iterate and validate

all steps time-boxed between 1 hour and a few days

early in project

later in project

MAREmethod

version: 1.0
October 20, 2017

Modeling and Analysis: Reasoning Approach
103  Gerrit Muller
Frequency of Assumptions, Decisions and Modeling

- Implicit (trivial?)
- Explicit
- Try-outs
- Very simple
- Small
- Key
- Substantial

Legend:
- Assumption (a)
- Input e.g. measurement (i)
- Decision (d)
- Model (m)
Life Cycle of Models

- **Understanding**
- **Exploration**
- **Optimization**
- **Verification**

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.

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

Creation and evolution of intellectual property assets.

Simple and small models:
- archived
- not maintained
- re-use

Substantial models:
- archived
- not maintained
- re-used in next project
- re-use

Try out models:
- abandoned

Version: 1.0
October 20, 2017
MAREmodelLifeCycle
Examples of Life Cycle of Models

- Understanding
- Exploration
- Optimization
- Verification

Try out models

Load/cost

Function mix

Peak impact

Load/stress test suite

Load/cost

Customer global distribution

Integral load model

Customer demographics

Global customer demographics

Web server performance

Webshop benchmark suite

Simple and small models

Substantial models (IP assets)
Architecting System Performance; Defining Performance

by Gerrit Muller
TNO-ESI, University College of South East Norway
e-mail: gaudisite@gmail.com
www.gaudisite.nl

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

<table>
<thead>
<tr>
<th>time-oriented</th>
<th>spatial</th>
<th>reliability</th>
</tr>
</thead>
<tbody>
<tr>
<td>response time</td>
<td>positioning accuracy</td>
<td>MTBF</td>
</tr>
<tr>
<td>latency</td>
<td>working envelope</td>
<td>MTTR</td>
</tr>
<tr>
<td>throughput</td>
<td>range</td>
<td>uptime</td>
</tr>
<tr>
<td>productivity</td>
<td>turning cycle</td>
<td>unscheduled breaks</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>energy/power</th>
<th>algorithmic</th>
<th>image quality</th>
</tr>
</thead>
<tbody>
<tr>
<td>energy consumption</td>
<td>sensitivity</td>
<td>sharpness</td>
</tr>
<tr>
<td>range</td>
<td>specificity</td>
<td>contrast</td>
</tr>
<tr>
<td>standby time</td>
<td>accuracy</td>
<td>color consistency</td>
</tr>
<tr>
<td>maximum power</td>
<td>coverage</td>
<td>color rendition</td>
</tr>
<tr>
<td>heat release</td>
<td></td>
<td>streakiness</td>
</tr>
<tr>
<td>cooling capacity</td>
<td></td>
<td>uniformity</td>
</tr>
</tbody>
</table>
Defining Performance

performance is a function of:

- context
- perception
- circumstances
- operation of interest
- system of interest
  - specification
  - design
  - configuration
  - version
  - history

scenario
use case¹

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.

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

- CPU
- cache
- memory
- bus
- (computing) hardware
- typical values
- interference
- variation
- boundaries
- network transfer
- database access
- database query
- services/functions
- duration
- footprint
- interrupt
- task switch
- OS services
- services
- operating system
- tools
- latency
- bandwidth
- efficiency
- overhead
- end-to-end function
- duration
- services
- interrupts
- task switches
- OS services
- CPU time
- footprint
- cache
- applications
- locality
- density
- efficiency
- version: 0
- October 20, 2017
- EBMIbenchmarkStack
system performance = f(operating system, services, applications, hardware, tools)
## Example μBenchmarks 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><strong>database</strong></td>
<td>start session</td>
<td>perform transaction query</td>
</tr>
<tr>
<td></td>
<td>finish session</td>
<td></td>
</tr>
<tr>
<td><strong>network, I/O</strong></td>
<td>open connection</td>
<td>transfer data</td>
</tr>
<tr>
<td></td>
<td>close connection</td>
<td></td>
</tr>
<tr>
<td><strong>high level construction</strong></td>
<td>component creation</td>
<td>method invocation</td>
</tr>
<tr>
<td></td>
<td>component destruction</td>
<td>same scope</td>
</tr>
<tr>
<td></td>
<td></td>
<td>other context</td>
</tr>
<tr>
<td><strong>low level construction</strong></td>
<td>object creation</td>
<td>method invocation</td>
</tr>
<tr>
<td></td>
<td>object destruction</td>
<td></td>
</tr>
<tr>
<td><strong>basic programming</strong></td>
<td>memory allocation</td>
<td>function call</td>
</tr>
<tr>
<td></td>
<td>memory free</td>
<td>loop overhead</td>
</tr>
<tr>
<td></td>
<td></td>
<td>basic operations (add, mul, load, store)</td>
</tr>
<tr>
<td><strong>OS</strong></td>
<td>task, thread creation</td>
<td>task switch</td>
</tr>
<tr>
<td></td>
<td></td>
<td>interrupt response</td>
</tr>
<tr>
<td><strong>HW</strong></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>
Measurements have stochastic variations and systematic deviations resulting in a range rather than a single value.
Be Aware of Error Propagation

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

- \( t_{\text{start}} = 10 +/-. 2 \, \mu s \)
- \( t_{\text{end}} = 14 +/-. 2 \, \mu s \)
- \( t_{\text{duration}} = 4 +/-. ? \, \mu s \)

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.
Tools and Instruments in the Benchmark Stack

typical small testprogram

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

Architecting System Performance; Measuring

version: 0
October 20, 2017
EBMibenchmarkPositions
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

Flowchart:
- raw material
- fetch
- transport
- process
- transport
- process
- transport
- deliver
- product
- store
- store
- store
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

<table>
<thead>
<tr>
<th>Unit of Buffering</th>
<th>== or &lt;&gt;</th>
<th>Unit of Synchronization</th>
<th>== or &lt;&gt;</th>
<th>Unit of Processing</th>
<th>== or &lt;&gt;</th>
<th>Unit of I/O</th>
</tr>
</thead>
</table>

- **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>
<th>large capacity</th>
</tr>
</thead>
<tbody>
<tr>
<td>fast technology</td>
<td>slow technology</td>
</tr>
<tr>
<td>small</td>
<td>large</td>
</tr>
<tr>
<td>expensive</td>
<td>low cost</td>
</tr>
</tbody>
</table>

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

*random data processing performance in ops/s*

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

*data set size in bytes*

*example data storage technology*
Pipeline pattern

production line = pipeline

car n+3 → car n+2 → car n+1 → car n

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

Feedback

Mapping

System performance
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.
## Greedy and Lazy Patterns

<table>
<thead>
<tr>
<th></th>
<th><strong>lazy</strong></th>
<th><strong>greedy</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(on demand, pull)</td>
<td>(push, forecast)</td>
</tr>
<tr>
<td><strong>what</strong></td>
<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>no resource usage unless needed</td>
<td>results are available immediately</td>
</tr>
<tr>
<td><strong>disadvantages</strong></td>
<td>time to result depends on execution time</td>
<td>some resource use is wasted</td>
</tr>
<tr>
<td><strong>when</strong></td>
<td>default</td>
<td>to achieve required performance</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(explore other concepts too!)</td>
</tr>
<tr>
<td><strong>this pattern applies to all domains</strong></td>
<td>(IT, goods flow, energy)</td>
<td></td>
</tr>
</tbody>
</table>
Start up of Systems as Example

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
## Caching Pattern (Physical Grab Stock)

<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>

Architecting System Performance; Greedy and Lazy Patterns

version: 0.1
October 20, 2017
ASPGLwhyCaching
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
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.
Architecting System Performance; Scheduling
by Gerrit Muller TNO-ESI, University College of South East Norway
e-mail: gaudisite@gmail.com
www.gaudisite.nl

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

<table>
<thead>
<tr>
<th>• Determine deadlines (period)</th>
<th>in terms of Frequency or Period (1/F)</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Assign priorities</td>
<td>Highest frequency (shortest period)</td>
</tr>
<tr>
<td></td>
<td>==&gt; Highest priority</td>
</tr>
<tr>
<td>• Constraints</td>
<td>Independent activities</td>
</tr>
<tr>
<td></td>
<td>Periodic</td>
</tr>
<tr>
<td></td>
<td>Constant CPU cycle consumption</td>
</tr>
<tr>
<td></td>
<td>Assumes Pre-emptive scheduling</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>0</th>
<th>9</th>
<th>15</th>
<th>18</th>
<th>23</th>
</tr>
</thead>
<tbody>
<tr>
<td>Thread 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Thread 2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Thread 3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

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

Source: Ton Kostelijk - EXARCH course

![Diagram of thread activity]

Source: Ton Kostelijk - EXARCH course
### 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>

**Source:** Ton Kostelijk - EXARCH course

---

**Diagram:**

- Thread 1: Period = 9, Processing = 3, Load = 33.3%
- Thread 2: Period = 15, Processing = 5, Load = 33.3%
- Thread 3: Period = 23, Processing = 5, Load = 21.7%

**Source:** Ton Kostelijk - EXARCH course
A perspective on dynamic behavior is to view the system as set of periodic behaviors.

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

Modern systems and Systems of Systems consists 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?

performance smaller is better

Architecting System Performance; Robust Performance
Gerrit Muller

version: 0.2
October 20, 2017
ASPRPvariation
Coping with Disturbances

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

steady state performance

disturbance

disturbance

degradation

degradation

recovery time
recovery time

performance
smaller is better

How does the system respond to disturbances? How quickly does it recover? How far does performance degrade?
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
<table>
<thead>
<tr>
<th>Characterize the system</th>
<th>use the system in varying conditions measure performance as function of the conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stress testing</td>
<td>where does the design fail? (go beyond specified limits)</td>
</tr>
<tr>
<td>Load testing</td>
<td>keep the system in heavy load condition observe how it keeps performing measure variations</td>
</tr>
<tr>
<td>(Accelerated) Lifetime testing</td>
<td>age the system observe how it keeps performing</td>
</tr>
</tbody>
</table>
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)
- Bloating of design
- Technical debt
- Increase in resource usage
- Design complexity
- Lack of overview insight understanding
- Performance problems

Loss of knowledge

Time effort gain by taking shortcuts

- Maturing of systems
- Increasing number of features (feature creep)
- Bloating of design
- Technical debt
- 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](http://gaudisite.nl/INCOSE2016_Callister_Andersson_SMARTtechnicalDebt.pdf)  
Value versus Performance Degradation

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
- bloating of design
- design complexity
- performance problems
- lack of overview
- insight understanding

Are benefits in balance with the costs?
Exploring bloating: main causes

- Poor specification ("what")
- Poor design ("how")
- Genericity and configurability
  - Provisions for future
- Core function
- Support for unused legacy code
- Dogmatic rules
  - For instance fine grain COM interfaces

Legend:
- Overhead
- Value
Necessary functionality \textgtr the intended regular function

- testing
- regular functionality
- instrumentation
  - diagnostics
  - tracing
  - asserts
- boundary behavior:
  - exceptional cases
  - error handling
The danger of being generic: bloating

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

needed code

bad code

copy paste modify

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

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

**Legenda**

- Overhead
- Value

**Core functionality**

- Configurability
- Provisions for future

**Support for unused legacy code**

**Poor design (How)**

- Decomposition overhead

**Poor specification (What)**

- Dogmatic rules (for instance fine grain COM interfaces)
Bloating causes performance and resource problems. Solution: special measures: memory pools, shortcuts, ...