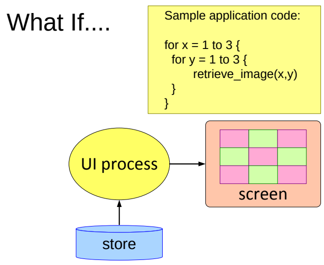
Introduction to System Performance Design

-



Gerrit Muller
University of South-Eastern Norway-NISE
Hasbergsvei 36 P.O. Box 235, NO-3603 Kongsberg Norway
gaudisite@gmail.com

Abstract

What is System Performance? Why should a software engineer have knowledge of the other parts of the system, such as the Hardware, the Operating System and the Middleware? The applications that he/she writes are self-contained, so how can other parts have any influence? This introduction sketches the problem and shows that at least a high level understanding of the system is very useful in order to get optimal performance.

version: 0.5 status: preliminary draft June 21, 2020

1 Introduction

This article discusses a typical example of a performance problem during the creation of an additional function in an existing system context. We will use this example to formulate a problem statement. The problem statement is then used to identify ingredients to address the problem.

2 What if ...

Let's assume that the application asks for the display of $3 \cdot 3$ images to be displayed "instanteneously". The author of the requirements specification wants to sharpen this specification and asks for the expected performance of feasible solutions. For this purpose we assume a solution, for instance an image retrieval function with code that looks like the code in Figure 1. How do we predict or estimate the expected performance based on this code fragment?

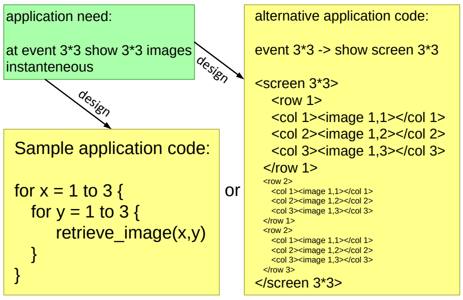


Figure 1: Image Retrieval Performance

If we want to estimate the performance we have to know what happens in the system in the retrieve_image function. We may have a simple system, as shown in Figure 2, where the retrieve_image function is part of a *user interface* process. This process reads image data directly form the hard disk based store and renders the image directly to the screen. Based on these assumptions we can estimate the performance. This estimation will be based on the disk transfer rate and the rendering rate.

However, the system might be slightly more complex, as shown in Figure 3. Instead of one process we now have multiple processes involved: database, user interface process and screen server. Process communication becomes an additional contribution to the time needed for the image retrieval. If the process communication is image based (every call to retrieve_image triggers a database access and a transfer to the screen server) then $2 \cdot 9$ process communications takes place. Every process communication costs time due to overhead as well as due to copying image

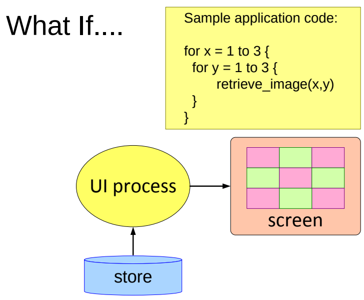


Figure 2: Straight Forward Read and Display

data from one process context to another process context. Also the database access will contribute to the total time. Database queries cost a significant amount of time.

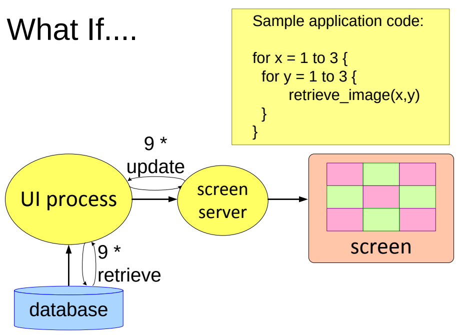


Figure 3: More Process Communication

The actual performance might be further negatively impacted by the overhead costs of the meta-information. Meta-information is the describing information of the image, typically tens to hundreds of attributes. The amount of data of meta-information, measured in bytes, is normally orders of magnitude smaller than the amount of pixel data. The initial estimation ignores the cost of meta-information, because the of amount of data is insignificant. However, the chosen implementation does have a significant impact on the cost of meta-information handling. Figure 4 shows an example where the attributes of the meta-information are internally mapped on COM objects. The implementation causes a complete "factory" construction for every attribute that is retrieved. The cost of such a construction is $80\mu sec$. With 100 attributes per image we get a total construction overhead of $9 \cdot 100\ cdot 80\mu s = 72ms$. This cost is significant, because it is in the same order of magnitude as image transfer and rendering operations.

Figure 5 shows I/O overhead as a last example of potential hidden costs. If the granularity of I/O transfers is rather fine, for instance based on image lines, then the I/O overhead becomes very significant. If we assume that images are 512^2, and

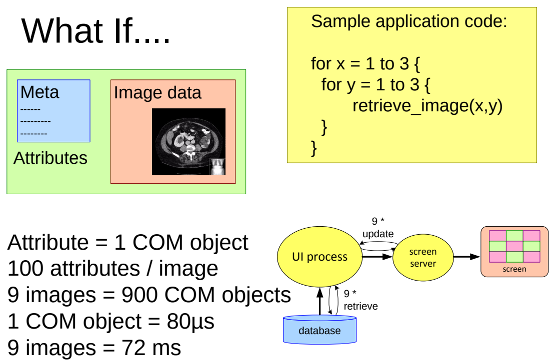


Figure 4: Meta Information Realization Overhead

if we assume $t_{I/O} = 1ms$, then the total overhead becomes $9 \cdot 512 \cdot 1ms \approx 4.5s$!

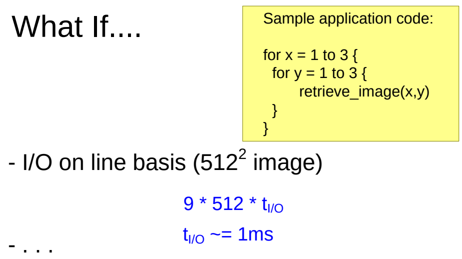


Figure 5: I/O overhead

3 Problem Statement

In the previous section we have shown that the performance of a new function cannot directly be derived from the code fragment belonging to this function. The performance depends on many design and implementation choices in the SW layers that are used. Figure 6 shows the conclusions based on the previous *What if* examples.

Figure 7 shows the factors outside our new function that have impact on the overall performance. All the layers used directly or indirectly by the function have impact, ranging from the hardware itself, up to middleware providing services. But also the neighboring functions that have no direct relation with our new function have impact on our function. Finally the environment including the user have impact on the performance.

Figure8 formulates a problem statement in terms of a challenge: How to understand the performance of a function as a function of underlying layers and surrounding

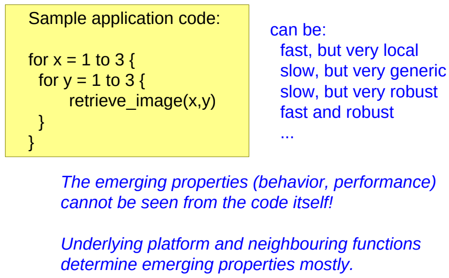


Figure 6: Non Functional Requirements Require System View

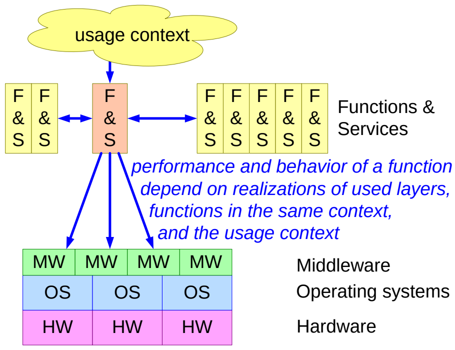


Figure 7: Function in System Context

functions expressed in a manageable number of parameters? Where the size and complexity of underlying layers and neighboring functions is large (tens, hundreds or even thousands man-years of software).

4 Summary

We have worked through a simple example of a new application level function. The performance of this function cannot be predicted by looking at the code of the function itself. The underlying platform, neighboring applications and user context all have impact on the performance of this new function. The underlying platform, neighboring applications and user context are often large and very complex. We propose to use models to cope with this complexity.

5 Acknowledgements

The diagrams are a joined effort of Roland Mathijssen, Teun Hendriks and Gerrit Muller. Most of the material is based on material from the EXARCH course created

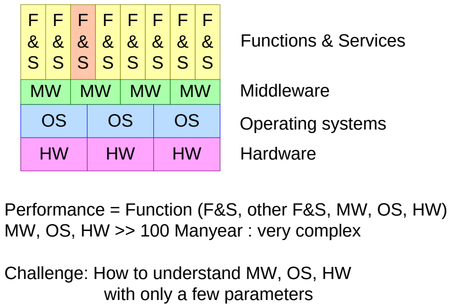


Figure 8: Challenge

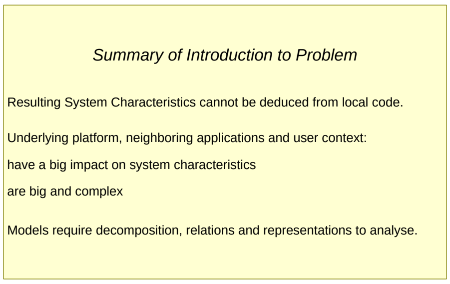


Figure 9: Summary of Problem Introduction

by Ton Kostelijk and Gerrit Muller.

References

[1] Gerrit Muller. The system architecture homepage. `http://www.gaudisite.nl/index.html`, 1999.

History

Version: 0.5, date: December 5, 2006 changed by: Gerrit Muller

- added application question to introduction of image retrieval performance
- removed slides with terminology, and removed section "Ingredients"

Version: 0.4, date: November 24, 2006 changed by: Gerrit Muller

- created article version
- changed logo

Version: 0.3, date: November 17, 2006 changed by: Gerrit Muller

- updated What If slides
- added Slide Title "System Performance Design: prerequisite information items"

Version: 0.2, date: November 10, 2006 changed by: Gerrit Muller

- added slides to connect what if to problem statement

Version: 0.1, date: June 12, 2006 changed by: Gerrit Muller

- relayout and reordering

Version: 0, date: February 8, 2006 changed by: Gerrit Muller

- Created, no changelog yet