Bachelor's thesis

Information Technology

Software Architecture

2013

Liangyi Luo

SOFTWARE EFFICIENCY OR PERFORMANCE OPTIMIZATION AT THE SOFTWARE AND HARDWARE ARCHITECTURAL LEVEL

TURUN AMMATTIKORKEAKOULU
TURKU UNIVERSITY OF APPLIED SCIENCES

BACHELOR´S THESIS | ABSTRACT
TURKU UNIVERSITY OF APPLIED SCIENCES

Information Technology | Software Architecture

December 2013| 51

Instructor: Balsam Almurrani

Liangyi Luo

SOFTWARE EFFICIENCY OR PERFORMANCE OPTIMIZATION AT THE SOFTWARE AND HARDWARE ARCHITECTURAL LEVEL

Some computer scientists point out that the efficiency or performance of software is decreasing when the hardware is becoming more powerful. There are many reasons contributing to this situation. The most important reason is that the cost for developing software is dramatically increasing while, at the same time, computer hardware is becoming cheaper and cheaper. Hence, an intuitive idea for achieving efficient software is to think of a neat way for optimization. Software can be optimized at algorithmic level or at architectural level. The architectural efficiency or performance optimization is mostly about the resource allocation. Therefore, finding a method for resource allocation optimization is crucial to achieve software efficiency or performance optimization at the software architectural level.

This thesis proposes a method which uses the response model R to analyze the relations between software performance/efficiency and resource from the software and hardware architectural perspective, hence provide a neat way for the performance or efficiency optimization.

KEYWORDS:

software, hardware, architecture, resources, optimization, efficiency, performance, response model R

ACKNOWLEDGEMENTS

In order to fulfill my plan of time and expense saving education, I was busy in finishing all kinds of studies in the past few months and at the same time I need to write a thesis with reasonable quality. This is certainly an extremely difficult yet beneficial thing to do. However, this plan would be impossible if I was not helped. Therefore, I would like to thank: Mr. Balsam Almurrani for being a great teacher, thesis supervisor, and friend, who provided me with important guidance in writing the thesis; Mrs. Skarli Poppy for being a great language teacher, who provided me with important language skills and corrected language errors in my thesis very swiftly; Mr. Patric Granholm, who is our department manager, for helping me in speeding up the procedure; Mr. Al-Bermanei Hazem for being my mathematics teacher, who taught me crucial mathematical knowledge for realizing my idea; Mrs. Luigia Petre and Mr. Ion Petre from Åbo Akademi for their courses inspired me to come up with the idea for software performance or efficiency optimization.

Finally, I would like to thank every teacher who armed me with knowledge. Just like what Francis Bacon said: “Knowledge is power.” Without such “power” I would not have been able to write this thesis in a very limited time.

CONTENTS

LIST OF TERMS **5**

1 INTRODUCTION **6**

2 SITUATION OVERVIEW **8**

2.1 The Necessity of Efficiency 8

2.2 The Necessity of Performance 9

2.3 How Other Quality Attributes Affect Performance and Efficiency 11

2.4 Hardware Environment Overview 12

2.4.1 Von Neumann Architecture 12

2.4.2 Architectural Details of Modern Computers 13

2.4.3 Optimization Guideline for the Architecture of Modern Computers 14

2.4.4 Trends of Hardware Architecture 15

2.5 Existing Approaches to Efficiency and Performance 18

2.5.1 Source Code and Algorithms 18

2.5.2 Innovations of the Software Industry 19

3 LITERATURE REVIEW **21**

3.1 Performance Factors 21

3.2 Performance Evaluation 21

3.3 Performance Engineering 22

3.4 Efficiency 23

4 METHODOLOGY **24**

4.1 Rules and Principles 24

4.1.1 General Rules 24

4.1.2 Performance Versus Efficiency 24

4.1.3 Extension Versus Modification 25

4.1.4 Algorithmic Versus Architectural 25

4.2 Method for Requirements 26

4.2.1 Define Performance or Efficiency Requirements 26

4.2.2 The Procedure of Optimization 27

4.3 Response Model R 28

4.3.1 How to Build Response Model R 28

4.3.2 Calculations 32

4.3.3 Formal Proof: Resource consumption view response model R 36

4.4 The Resource Allocation Optimization Problem 42

4.5 An Example of Optimization Using Response Model R 43

4.5.1 Software and Hardware architecture 43
4.5.2 Define the Optimization Target 45
4.5.3 Build Response Model R and Calculate Response Time T 45
4.5.4 Optimization 46
4.5.5 Result of Optimization 48
5 CONCLUSION 50
REFERENCES 51

PICTURES

Picture 1. Resource consumption view example 29
Picture 2. Working flow chart example 30
Picture 3. Graphical representation(Triple-graph) example 31
Picture 4. The resource consumption view and working flow chart of the example 44
Picture 5. The graphical representation of response model R 45
Picture 6. The graphical representation of optimized response model R' 47
Picture 7. The resource consumption view of the optimized architecture 49

TABLES

Table 1. Peformance Engineering Tactics 23

LIST OF TERMS

Architectural Optimal Performance or Efficiency State

A software architecture design is in architectural optimal performance or efficiency state if its resource allocation cannot be further optimized (ideal resource allocation).

Event

An event is the operation of a user, data arrived, or time passage of a software system.

Laws of Inefficiency

Statements such as: “software is becoming slower and compensating performance growth of hardware” are collectively known as Laws of Inefficiency, because the essential problem these statements refer to is that software is becoming inefficient.

Memory economy

Memory economy is the economical efficiency of software on memory consumption.

Optimizable

A software architectural is optimizable if it is not in architectural optimal performance or efficiency state.

Out-of-order execution

“In computer engineering, **out-of-order execution** (**OoOE** or **OOE**) is a paradigm used in most high-performance microprocessors to make use of instruction cycles that would otherwise be wasted by a certain type of costly delay.”(Wikipedia, *Out-of-order execution*)

Resources consumption view

Resource consumption view is a type of software architectural view which indicates resource consumptions and dependency relations among components.

Stakeholders

By convention, the people and organizations who are interested in the construction of a software system are called stakeholders. Stakeholders can be customers, end users, developers, project managers, maintainers, or even marking personnel. (Bass et al. *2003)*

Triple-Graph

The graphical representation of the response model R.

Overabundance of hardware resources

Nowadays, most PCs are on idle in over 90% of their runtime, which suggest that the hardware resources are not in use for the most of the time. This phenomenon is called “overabundance of hardware resources”.

1 INTRODUCTION

Performance is one of the essential quality attribute of software. Users of software usually regard performance as an important standard to decide whether the software is good to use. In many cases, the performance of software reflects the efficiency of software, because the software which makes proper use of resources is usually running fast. However, software development in the real world does not follow the path of hardware development which reduces the cost but doubles the performance per period of time. On the contrary, the software development is neutralizing the advantages brought by the hardware development.

Early in 1995, Computer Scientist Niklaus Wirth (1995) stated that “Software is getting slower more rapidly than hardware becomes faster.” The statement was later known as Wirth’s Law. Because software is becoming more complex as the hardware develops, the actual performance of computers might not be improved as people expected. The term “Software bloat” was coined to describe the phenomenon. Afterwards, computer scientists continue to make similar statements as Wirth’s Law, for example, British computer scientist Michael David May stated that “Software efficiency halves every 18 months, compensating Moore’s Law”. This statement became later known as May’s Law. Wirth’s Law, May’s Law, or any other Laws of Inefficiency counteract the effect of Moore’s Law. As computer scientists continue complaining and the situation nowadays continues acting as corroborative proof, those Laws of Inefficiency turn out to be correct. On the one hand, software is slow and inefficient; on the other hand, hardware industry strictly follows Moore’s Law, providing overabundant hardware resources (Processor’s computing power, RAM, and hard drive capacity): the inefficiency of the software and the overabundance of hardware resources coexist to keep validating Laws of Inefficiency.

This thesis enumerates and analyzes the reasons why software became slow and inefficient, studies the existing efforts to counteract the problem, and introduce a powerful tool for software optimization at the software and hardware architectural level.

The thesis comprise of the following chapters:

Chapter 1 contains the introduction of the background and thesis overview.

Chapter 2 contains the overview of the current situation of software efficiency and performance and discusses stimuli for efficiency and performance of software as well as the effort of software industry to create efficient software. In other words, based on the current hardware the thesis describes the general approaches for efficiency and performance of software.

Chapter 3 contains the literature review of the current methods and approaches in the research field of software architecture for efficiency and performance optimization.

Chapter 4 contains the methodology. In chapter 4, the optimization methed will be introduced in the first place. Afterwards there will be a series of formal proofs on the validity of the method. At last, an example of implementing the method will be presented.

Chapter 5 is the conclusion.

2 SITUATION OVERVIEW

2.1 The Necessity of Efficiency

The fast pace of hardware development provides enriched hardware resources that increase the tolerance to inefficiency. Among all kinds of popular software on computers, there are only a few such as games which can make full use of hardware resources which are sensitive to efficiency. For rest and most of the software, hardware resources are truly overabundant. Therefore, developers of less demanding software gradually lower their standard on efficiency. However, this will not compromise their competitiveness, but on contrary, it may even boost their competitiveness. Developing more efficient software requires better design and more optimization, hence higher costs. Moreover, the breakthrough in storage technology also significantly reduces the cost per unit memory. Daily software which takes 100% more memory may only result in a minor increased spend on hardware. Furthermore, the development of computational power ensures that inefficiency software can be as functional as efficient software with a just price of higher (but not full) CPU load. Nevertheless, neglecting software efficiency will not become an issue, because it is compensated by the still increasing hardware performance as long as the semiconductor industry does not reach its bottleneck.

The above analysis suggests that highly efficient software would not bring obvious benefits to software vendors. However, the emerging mobile trend adds considerable weight to the efficiency of software. On mobile computers, such as laptops, tablets, and smart phones, the efficiency of software essentially affects the battery life, which acts as an important factor of user experiences. Given that the same functionality is to be achieved, the less hardware resources consumed, the less electricity consumed, hence longer battery life and better user experiences. Recently, the battery life is extended mostly by improving the efficiency of processors, enlarging the capacity of batteries, and implementing

hardware power saving features. These improvements on hardware technologies effectively extend the battery life when a mobile computer is idle. Once a mobile computer is well loaded by software, the battery life could be drastically shortened, which is a very common issue on smart phones. The mobility of mobile computers is valid if and only if the computer runs on battery. Therefore, the key to enhance mobility is to enhance software efficiency.

A few pieces of inefficient software installed might not be problematic in a short term. Still, the use of inefficient turns out to be unsustainable after all. The situation is that the mainstream software market encourages developers to develop less efficient or even less qualitative software. Nowadays, it is very likely that after a considerablenumber of slow and bloated software installed or updated, the computer will be gradually drag the system to a slow state. To address this kind of situation, most users who cannot bear it would like to upgrade their computer. As a result, both hardware and software manufacturers successfully maintain their source of profit, yet users barely have a choice other than purchasing when they eventually realize the always existing performance issue.

In conclusion, no matter how fast the hardware develops, the efficiency of software is a crucial necessity in order to achieve a continuously better user experience as well as help establish a healthier development model.

2.2 The Necessity of Performance

Similarly, robust hardware has more tolerance to slow software, yet optimizing the performance will generate more cost. Software developers realize that most of the end users may not notice the improvement at the price of extra costs. For example, if there are two pieces of text editor software with exactly the same function, one take 250ms to load, another takes 500ms, the more efficient one is 100% faster than the less efficient one, but for most end users, this 100%

faster makes no distinct difference. In this case, most of the users would like to choose inessentially slower but cheaper software. The result is that commercial software providers found out that the performance enhancement will not be more lucrative than other enhancements, such as user interfaces or functionality, as long as their software meets a general performance attribute for a type of software.

It is true that for most of software for daily use, the performance requirement is not strict. For example, if there is a text editor, when a user presses a letter on the keyboard and the letter appear on the screen where the cursor is within a time constraint of a few microseconds, then, this text editor can be judged as a properly performed text editor in the task of text input. Most of the text editor software nowadays has an acceptable performance attribute.

It is normal that end users are not able to notice a minor performance advantage in a single function of software or a single piece of software. However, performance enhancement in the majority of software of a single PC system will trigger a qualitative improvement to user experience. Subjectively speaking, end users will feel that the use of a PC is “fluent”; in other words, everything runs smoothly. Apple’s Macintosh computer class can be taken as an example for how performance stimulates user experience. The hardware architecture of Macintosh computer system has been transited from IBM PowerPC architecture to Intel x86-64 architecture early in 2006, which means that the Macintosh computer system at present uses basically the same hardware as Windows or Linux PC system. Despite the change on hardware, the user experience of Macintosh remains robust. This is because Apple has a closed ecosystem for software on Macintosh computer systems, hence, the performance requirements of Mac OS operation system and software runs on it are stricter than software run on Window PC. Excellent performance plus other qualities reinforce a superior user experience.

Efficient software usually performs well. Given that average resources used per unit time is the same, if the software finishes a task faster, it means that the software consumes less resources, thus, it saves computing power and has good performance as well. When we see the software which finishes the same task faster than other software, we can basically regard this software as more efficient.

Given that the computer system used is the same, the computer system can run more instances of efficient software (finish more tasks) per unit time, because the average resource consumption per software is lower. Thus, the overall productivity brought by the system is higher than a computer system running less efficient software. Globally speaking, assuming that stakeholders provide fixed investment in computational power , which will also be fixed, for a type of production, the more efficient the software, the more tasks can be done with the fixed amount of computational power within unit time. Therefore, efficient software brings extra productivity.

2.3 How Other Quality Attributes Affect Performance and Efficiency

In addition to performance, there are other/additional quality attributes, namely, security, modifiability, availability, testability, and usability. Among these quality attributes, performance is usually not the only attribute of concern. When another attribute becomes more critical than performance, performance is very likely to be traded-off for the more critical attribute as long as the software meets the pre-defined performance requirements. In fact, achieving most of the attributes listed above will bring more or less impact on performance. If security is more critical, then an encryption module would be added to the architecture to ensure data confidentiality, with a price of performance. If modifiability is more critical, virtual machines or interpreters could be used to avoid less comprehensible low-level coding, with a huge price of performance. If

availability is more critical, redundant system architecture should be adopted, yet the redundancy will be functional only if it keeps synchronized with the part in operation. This has minor impact on performance. To enhance testability, an input/output monitor module will be added which consumes resources at runtime. To enhance usability, the architecture might be designed in a way that user operations are simplified with the cost of the background operations becoming complicated, hence performance will be penalized.

In practical situations, performance optimization is allowed if and only if other quality attributes that required are not compromised. This indubitable principle usually renders performance optimization difficult and expensive. On the occasion that the software is bad designed and unfortunately cannot meet the performance requirement without a change of functionality, it will usually be discarded if the cost for fix is larger than remake. However, performance issues can sometimes be resolved by better hardware, which implies that Laws of Inefficiency are real.

2.4 Hardware Environment Overview

2.4.1 Von Neumann Architecture

Basically, almost all modern computers on the markets (computers under researching in laboratory such as quantum computer and biological computer are not within the scope of this thesis) are built based on the Von Neumann Architecture.

Key components inside a Von Neumann Architecture computer can be categorized to three parts: Control Unit (CU), Arithmetic and Logic Unit(ALU), and Memory. One major bottleneck of the Von Neumann Architecture lies in the data transfer among these three components. Most occasionally, the Control

Unit and the Algorithm Unit might need to wait for memory to fetch the data they require for processing.

2.4.2 Architectural Details of Modern Computers

In addition to ALU, processors of modern computer systems usually have independent Floating-Point Units (FPU). The speed of ALU and FPU depends on how manufacturers design their processors. Usually, CPUs designed by Intel have both strong ALU and FPU, but CPUs designed by AMD in recent years may contain a weaker and less efficient FPU. In most of the cases, ALU is designated for integer arithmetic and logic operation and FPU is designated for the arithmetic of floating typed data. However, there are certain tasks which are solvable with both FPU-dependent algorithms and ALU-dependent algorithms. A simple example can be taken as follows. To calculate the integer part of the sum of 10*0.5 and 7*0.2, if the algorithm is written as 10*0.5+7*0.2, then it runs on FPU; if the algorithm is written as (10*5+7*2)/10, then it runs on ALU. Depending on CPU's architecture, the speed of ALU and FPU might be noticeably different.

Memory is another component which is as important as Processor. The memory of a modern computer has been designed, based on their speed, as multiple stages of memory. The cache of processors is the fast and smallest memory, followed by the RAM of which the speed is significantly slower but also bigger than the processor's cache. Hard Disk Drive (HDD) or Solid State Drive (SSD) is the slowest yet largest memory. Likewise, bandwidths between those memories descend gradually. The highest bandwidth is the bandwidth between processing units and caches. The second higher bandwidth is in between cache and RAM. The bandwidth between a cache and HDD/SSD or between RAM and HDD/SSD is the lowest.

The set of instructions which a processor can understand and execute is called the instruction set of the processor. It is the “language” of the processor. Instruction sets vary according to the architectures of processors. Most of the computers today can be put into two categories, CISC (Complex Instruction Set Computer) or RISC (Reduced Instruction Set Computer) according to the instruction set which their processor can follow. CISC is usually easier to program and more flexible, in addition, it has better memory operation performance. RISC, which is reduced from CISC, provides better efficiency in most of the computations, however, RISC is usually harder to program and less efficient in memory operation. Most of the general purposed computer systems at present belong to CISC.

One of the most important metrics for computational power of processors is IPS (Instructions per Second), because tasks of every kind handled by processors can be summarized as executing instructions. When software is activated and the first task is assigned, the first action taken by the processor is to retrieve relative instructions from memory (this operation is done by executing instructions about starting software), then the processor follows these instructions to manipulate data or perform computation, hence finish the task**.**

2.4.3 Optimization Guideline for the Architecture of Modern Computers

In general, according to the architecture of modern computers, the working principle of a modern computer can be explained as follows: a processor processes data retrieved from memory and/or sends the memory back to memory by executing instructions.

When software is running, if the processing capacity occupied by the software does not match the allocated memory throughput, either the processor will be waiting for the memory or the memory will be waiting for the processor (in most cases, the processor waits for memory). Although modern processors might

have out-of-order execution capability to improve the efficiency of the entire system, the efficiency of the software is non-ideal, due to the time spent on waiting. Moreover, software will become less efficient if it takes more instructions for finish a task, because more instructions take more time to be executed. Likewise, software that takes unnecessarily more memory for finishing a task will become less efficient because the processor is more likely to wait for the memory to transmit the data needed.

According to the architecture of hardware, there exist the following general optimization guidelines:

- Arrange the architecture of the software in a way that the processing capacity and memory throughput matches each other;
- Minimize instructions needed for finishing a task;
- Minimize the memory occupied for finishing a task.

2.4.4 Trends of Hardware Architecture

2.4.4.1 Trends of Processors

CPU still holds its position as the most important processor of computers. However, the growth of the clock rate of CPU has slowed down since 2003. CPU manufacturers have to implement parallelism to increase the computational power of CPU. CPUs of PCs evolve from 1 core to 4-8 cores. Yet, parallelism will not be a sustainable solution. This is because, for non-parallelism sensitive software such as daily used software on PC, adding more cores to CPU will not able to bring corresponding performance growth. CPU manufactureers have to also improve CPU architecture. All in all, parallelism will still be a main feature of CPU and CPU performance growth is slowing down.

In the near future, computing power-consuming software need to be not only capable of parallelism but also of optimization according to CPU architecture revision and support for new instruction sets per CPU generation as well. In this way, software can surely provide as good performance as possible. For operation systems, process management should be further enhanced.

Heterogeneous-computing is another major trend of processors. Nowadays, a PC has not only a CPU for general purposed processing and control but also a GPU for processing graphics-depended tasks such as rendering game graphics or video decoding. GPU has evolved greatly in the last decade and has become a formidably powerful kind of processor for floating-point process. The physical architecture of GPU has also evolved to be general purpose processing capable, hence, most of the GPUs at present are able to accomplish general purpose floating-point intense tasks which could only be done by CPU a decade ago in greater efficiency. Compared to CPU architecture, GPU architecture is more heavily paralleled, thus software on heterogeneous-computing architecture should be capable of parallelism as well.

In the last few years, CPU manufacturers have begun to blaze trails in the integration of CPU and GPU. AMD named its CPU and GPU integration processor as Accelerated Processing Unit (APU). APU has already been widely used in tablets, laptops, PC, and the next generation of game consoles. Likewise, Intel has most CPU on its consumer level CPU product-line integrated with GPU.

Heterogeneous-computing can also be implemented to either APU architectural computer systems or non-APU architectural computer systems. The main idea of optimization for heterogeneous-computing system is to assign floating-point intense tasks which can be performed more efficiently on GPU than on a GPU

or the GPU “part” of an APU. However, there is a difference in memory architecture between APU architectural and non-APU architectural system. APU has unified memory for both the CPU “part” and the GPU “part”, but on other heterogeneous system, either CPU or GPU has its own memory. For this reason, software optimization differs on these two types of heterogeneous-computing systems. The future development of APU will be “Architectural plus OS Integration” which includes “fully unified memory and scheduling” (Advanced Micro Devices 2011). Therefore, on an APU architectural computer, this is no consideration of exchanging data between RAM and Graphical Memory.

2.4.4.2 Trends of Memories

Because the memory technology is also driven by semi-conductor technology, the space and performance of memory also grows according to Moore’s Law. The price of unit memory is overall decreasing, the power consumption per unit performance is decreasing, and the speed is steady increasing. In other words, with the same cost, end users can purchase larger and faster memory which is more power-saving than previous memory. The next generation of PC will be generally equipped with 8-16GB DDR4 RAM, which is larger and faster than 4-8GB DDR3 RAM of the current generation of PC.

In the field of Non-volatile memory, SSD is emerging in the last few years. Compared to HDD, SSD is notably faster in speed and small in capacity per unit price. Because SSD uses chips as the storage media, it is the kind of memory which the development follows Moore’s Law. Therefore, cheaper and faster SSDs can be expected in the near future. On contrary, the traditional disc storage technology such as HDD makes slow progress in speed and capacity. Nevertheless, it still maintains capacity and lifespan advantage.

2.4.4.3 Optimization Guidelines for Trends of Hardware

The configuration of the next generation of computers is very likely to include highly paralleled processing architecture, larger RAM for runtime data, faster SSD for executable data storage, and larger HDD for normal data storage. In addition, the CPU “part” of the processing architecture might not be significantly faster than the current CPU. Therefore, the abstract principle for software optimization according to hardware development would be as follows:

- Enhance parallelism;
- Reduce CPU power dependency, increase GPU power dependency;
- Trade off memory economy for speed;
- Reduce software size;
- Separate execution data from storage data.

2.5 Existing Approaches to Efficiency and Performance

2.5.1 Source Code and Algorithms

Code optimization can be done from multiple aspects in various ways. Generally, code optimization is targeting several performance bottleneck lies in the software. In addition, local optimization can be automatically done by optimizers. In most of the cases, code optimization is carried out after the building/creation/development of software. Code optimization should be subtly done in order to reduce the impact on software’s maintainability and functionality. That is because code optimization usually renders code less comprehensible, or even worse, turns the originally fully functional software defective. Code optimization can also be done during the designing of software or writing of the original code. This kind of code optimization involves working on the algorithm, which can be also seen as an independent approach performed at the beginning of development.

In addition to code optimization, algorithm improvement is also an important approach. Generally speaking, an algorithm is series of procedures for finishing

a task. In software development, implementing better algorithms can also be seen as a means of code optimization which is done when software is designed. Given a task and a programming language, programs can be written in different ways, yet the outcome remains the same. Because the efficiency of software mostly depends on the efficiency of algorithms, algorithm efficiency can be treated as one of the most important approaches to efficient software.

2.5.2 Innovations of the Software Industry

The IT industry came out with several innovative, also lucrative, ideas for macroscopically enhancing the software efficiency. There are two major approaches: Cloud Computing and Distributed Computing.

Cloud Computing increases efficiency by centralized the resource. In computing cloud hardware resources may be supplied according to need. End users only need to pay for the resources required but not for the resources of an entire computer, hence saving money. The overall resource can also be fully used. However, cloud computing brings security concerns. Users may not trust vendors completely. Moreover, to ensure the safety of data, a private backup is also needed. Cloud Computing may take time to be popularized due to the unbalanced development of network connection speed.

Distributed Computing is another trial of utilizing the idle computing power of computers using an inverse of the idea of Cloud Computing. Owners of computers can install Distributed Computing clients to share tasks of some internal research project which requires high performance computing. In this way, the overabundance resources are utilized for making contributions to mankind. One of the famous and successful Distributed Computing systems is Folding@Home. However, the efficiency of distributed computing also relies on the internet connection speed.

Both of the approaches are optimizing the efficiency macroscopically. They do not essentially improve the efficiency of software, but avoid further waste caused by inefficiency. Moreover, neither Cloud Computing nor Distributed Computing is popular enough to resolve the issue that software lacks efficiency. Be it cloud computing or distributed computing or any other macroscopic approaches, what has been achieved now is preventing the further waste of computational power. The issue of software inefficiency still exists.

3 LITERATURE REVIEW

3.1 Performance Factors

Performance is a very important attribute of software. It used to be the main driving forces behind the development of software architecture. However, as it was mentioned in the previous part, performance is no longer the only quality attribute concern in the design of a software architecture. This shift was explained in the book Software Architecture in Practice:

> "As the price/performance ratio of hardware plummets and the cost of developing software rises, other qualities have emerged as important competitors to performance."

—Bass et al. 2003, p.85

In addition, software functionalities, resources, and the algorithm used can affect performance. In the book "Software Architecture in Practice", which explained that performance may depend on those the following factors:

- the amount of communications needed among components;
- functionalities allocated to the components;
- the way that shared resources are allocated;
- the algorithm selected for a functionality;
- the coding of the selected algorithm.

Among these factors, the first three are architectural and the last two are non-architectural. (Bass et al. 2003, p.86)

3.2 Performance Evaluation

Generally speaking, performance is concerned with how swift the software "response when an event occurs" (Bass et al. 2003, p.83) Events arrive the software system in various patterns which can be "characterized as periodic or stochastic" (Bass et al. 2003, p.83). To evaluate whether a system is well performing, the time between the event and the response can firstly be

measured, then compared with a previously determined time constrain.

3.3 Performance Engineering

According to the book Software Architecture in Practice, there are “two basic contributors to the response time”. (Bass et al. 2003, p.112) They are resource consumption and block time. Responses to events require resources which include hardware resources and data stores. When the resource is not available due to the contention or resource dependency, the process for the event will be blocked and suspended until the resource becomes available. Therefore, tactics for performance engineering can be categorized into three categories: resource demand, resource management, and resources arbitration. (Bass et al. 2003, p.113)

According to the book Software Architecture in Practice, performance engineering tactics can be summarized into the following table.

Table 1. Peformance Engineering Tactics

Categories	Tactics	Functions
Reducing or Managing Resource Demand	Increase Computation Efficiency	Improve algorithm efficiency
	Reduce Computational Overhead	Remove intermediaries
	Manage Event Rate	Avoid unnecessary sampling
	Control Frequency of Sampling	Sample at a lower frequency
Resource Management	Introduce Concurrency	Process in parallel and load balancing
	Maintain Multiple Copies of either data or computations	Replicate data or computation instance or make them available to multiple clients
	Increase Available Resources	Increase hardware resources
Resource Arbitration	Scheduling Policy	Schedule the resources to minimize contention

3.4 Efficiency

There hardly is any literature on efficiency engineering from the software architectural point of view. The reason for the lack of literature could be that efficiency engineering is usually seen as a part of performance engineering. As already mentioned, improving efficiency will boost performance in almost all cases. Nevertheless, efficiency engineering is also about optimizing resources allocation. Therefore, in the following methodology part, the method for performance or efficiency optimization will be the same, but what is different is the requirement (time constrain or resource limits).

4 METHODOLOGY

4.1 Rules and Principles

4.1.1 General Rules

There are two general rules for performance and efficiency optimization. The first rule is that performance or efficiency optimization should be done without changing the functionality of the original software. Secondly, performance or efficiency optimization should not cause such a side effect that any other quality requirement can no longer be met. This is because software nowadays needs to meet several quality requirements other than performance and these quality requirements might have the same or even higher priority as performance requirements. However, if there is a quality attribute which far exceeds the requirement on it, it is acceptable to trade off the “superior” quality attribute for performance or efficiency.

4.1.2 Performance Versus Efficiency

As it was explained in the previous part, for most of the time, performance is positively correlated with efficiency which applies the following principle:

- Given the same amount of resources needed per unit time, the faster, the more efficient(less resources needed).

According to this principle, the software which uses less resource such as computational power of CPU to finish a task needs less time. This is true in most cases. However, for some computationally intense tasks, such as solving complicated problems, it is usually considered that the shorter the time needed, the better. In this case, one way to shorten the time needed is to use as much resources as possible, yet the resource investment might not bring the corresponding growth of performance, i.e., the performance of the software system is increasing, and at the same time the efficiency is decreasing.

If performance is the only requirement, it is acceptable to trade off efficiency for performance. When efficiency is also required, a balance between the performance growth and the efficiency penalization should be kept according to the) requirements.

4.1.3 Extension Versus Modification

For a completely functional software system which is able to meet all the requirements including or not including the performance requirement, the preferred approach for performance and efficiency optimization is to make extensions to architecture than modifications on components as an approach. This is because the modifications upon originally fully functional components may render the software defective or even completely nonfunctional. Extensions may cause more memory consumption, but this is a less significant problem due to the continually decreasing memory price (per unit memory).

4.1.4 Algorithmic Versus Architectural

Recalling the factors which exert influence on performance in Chapter 3.1 of Literature review, it can be seen that there are two non-architectural but algorithmic factors: the algorithm selected and the coding for the algorithm. These two factors are actually all about algorithm efficiency. Specifically, “algorithms selected” refers to which mathematically efficient algorithms are used for achieving a function and “the coding for the algorithm” refers to how the mathematical algorithm is represented efficiently in computer codes.

There are the following principles about the efficiency of performance-critical computations:

- “The performance-critical computations should be made as efficient as possible.”
- “The performance-critical computations should be scheduled to ensure the achievement of the timing deadline.”

(Bass et al. 2003)

Although the focus of this thesis is not what algorithm to choose and how the algorithmic optimization can be done, the algorithm is a critical factor of software performance and efficiency. When the optimization methods are explained, we assume that the most suitable algorithms are already chosen and are well programmed, in the words proposed by Software Architecture in Practice, “performance-critical” computations have already been well tuned into efficiency optimal. With such preconditions, all the performance and efficiency issues are in software architecture.

4.2 Method for Requirements

4.2.1 Define Performance or Efficiency Requirements

Performance or efficiency optimization is a meaningful topic if and only if requirements on time constrain or resource limits are given. An optimization subject, which is the response of the software to an event, should also be identified. Therefore, we say: “With given resources limits of amount p in computational power, s in memory space, d in data, how can the response model R(optimization subject) be optimized such that the response time T to a given event e, is in the given time constrain T_c”.

Given that resource in computational power p, resource in memory m, resource in data d, event to be responded e, time constraint T_c, find the optimized response R, then $\mathrm{T} = \mathrm{R}(\mathrm{p}, \mathrm{m}, \mathrm{d};\ \mathrm{e}) \leq \mathrm{T}_c$

In scientific computations where a computational result for solving a problem is expected, the event is the start of computation and the response is the output is given. In this kind of case, the shorter the response time is the better. This case the requirement can be denoted as:

Given that resource in computational power p, resource in memory m, event to be responded e, find the optimized response R, then $T = \min[R(p, m, d;\ e)]$.

In practical software developments, different stakeholders have their own style of performance evaluation hence the metrics of requirement varies. For example, an end user stakeholder will evaluate the “drag” feature of a tablet computer by “feel” if it is “fluent”. The software architect who is in charge of optimization of the “drag” feature should convert this “fluent” into specific requirements such as the response (a new frame generated to show the drag movement) should be made within 30ms, with computational power of p, and memory of m.

4.2.2 The Procedure of Optimization

Software architectural (not algorithmic) optimization to the performance or the efficiency is mostly about the resource allocation optimization. To achieve optimization, there are five basic steps which can be followed.

Step 1: Acquire information about the software and hardware architecture. Here, computational power of the processor, memory speed and capacity is especially crucial. In addition, how the software module responds to an event has to be known. A resource consumption view is needed.
Step 2: Determine the optimization target. Optimization target or efficiency and performance requirement needs to be defined. The upper limits of each type of resources and the time constraint for the response should be set.
Step 3: Build response model R and evaluate the current performance. The resource consumption view needs to be transformed to response model R.
Step 4: Optimize over response model R. Guidelines, rules, and principles mentioned in the previous chapter should be applied in optimization.

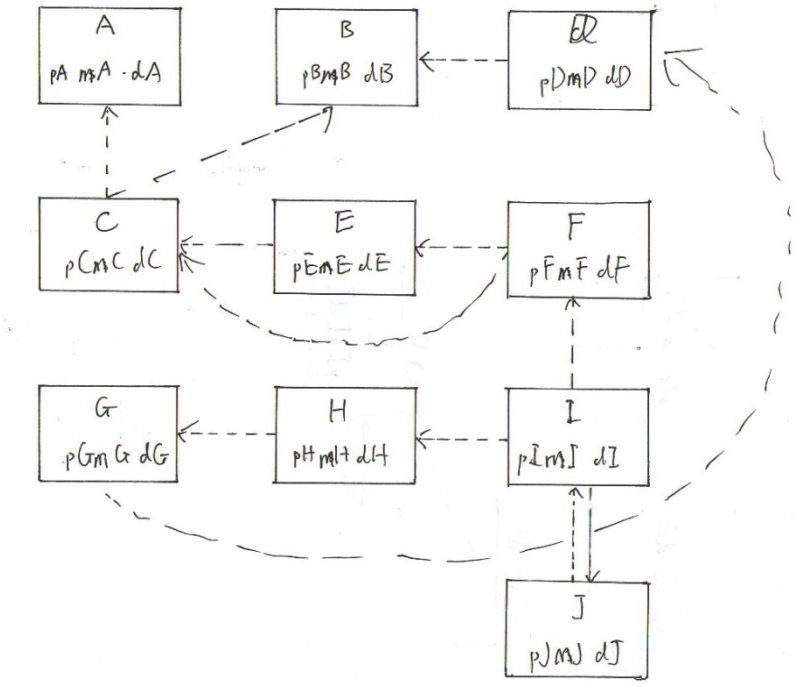
Step 5: Transform the optimized response model back to software architectural view. The result of optimization is an improved architectural view of the software which can be used for practical optimization.

It is possible that software might not be optimizable without the changing of its functionality or penalizing other quality attributes from architectural point of views. In this case, we say that this software reaches its architectural optimal performance or efficiency state. The software in its architectural optimal performance or efficiency state only means the software gives its best performance or efficiency with certain resource limits in this architectural design. It does not suggest that the software can meet the pre-determined performance or efficiency requirement. If the improperly designed software is not able to meet the requirements by optimization, then this software architecture is not optimizable and needs to be redesigned.

4.3 Response Model R

4.3.1 How to Build Response Model R

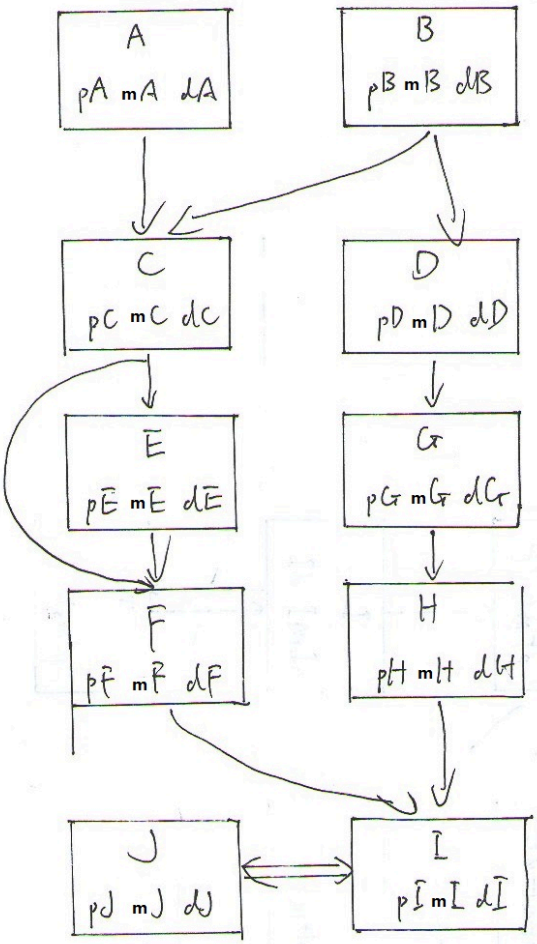
The response model R is the tool we use for optimization. To obtain a model R, firstly we need the resource consumption view (Picture 1) of the software module.



Picture 1. Resource consumption view example

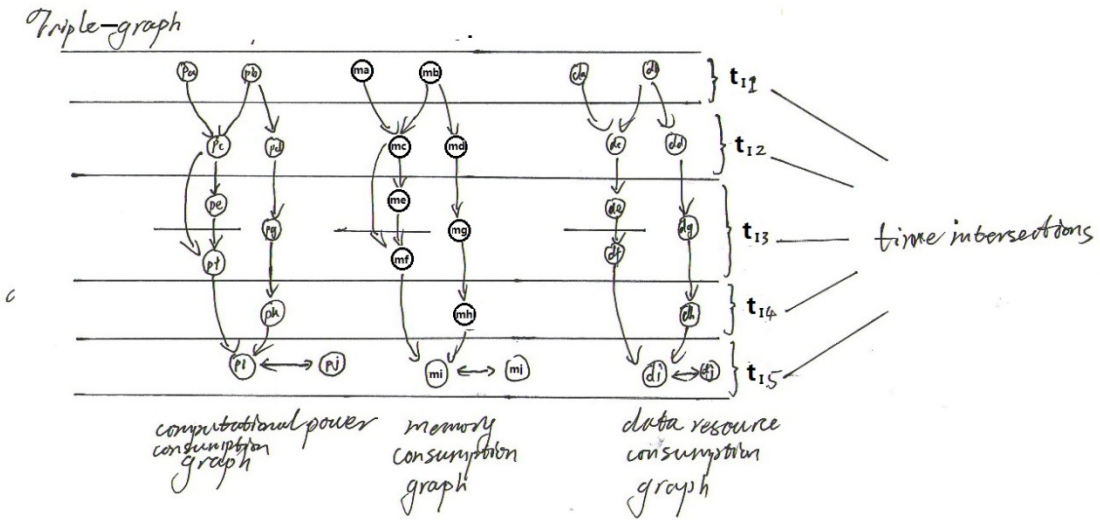
In the resource consumption view we have dash-lines with arrows which indicate the dependency relations among components. If there are two dash-lines with arrows of which the directions are opposite to each other, this indicates that the two components are dependent on each other and that there might be a redundancy here with requires perfect synchronization. In addition, in each component, the resources it needs will be marked. The resource is marked in this way, for a component X, we define a standard time t_s(unit: ms). The computational power consumption pX (unit: MIPS) is the computational power rate the component X needed to finish its task in the standard time t_s. If the component does not need a certain type of resource or the amount needed is trivial, then we mark the resource consumption as 0.

Secondly, we transform the resource consumption view into a working flow chart (Picture 2). Now the lines with arrow indicate the data flow. Two components which are in synchronization are indicated by lines with two arrows.



Picture 2. Working flow chart example

Thirdly, we draw the Triple-Graph, which act as the graphical representation of the response model R, according to work flow chart as the following.



Picture 3. Graphical representation(Triple-graph) example

Each graph in the Triple-Graph represents a type of resource consumption. The resource consumption are shown in numbers (without units) marked inside the node which represents the corresponding component in resources consumption view. The numbers used to represent resource consumption can be calculated in this way: Define the amount of resources that number 1 represents, and then use this relation to calculate the number used for represents the resource consumption. For example, if component X has the resource consumption marked as the following: pX=10MIPS, mX=63MB, dX=1(copy), and we let number 1 represent 10MIPs computational power rate, 5MB memory, and 1 copy of data; then we have the following resources consumption representation in model R: px=1, sx=12.6, sd=1.

Lines drawn are to illustrate time intersections indicating the time needed for a component or several components which work in parallel to finish their tasks. The following rules regarding nodes in time intersections are applied.

- The three nodes in three graphs which represent the same component should be in the same state (either working or blocked).
- The node which depends on another node has to be placed inside a later

time intersection, except the mutual dependencies.

- In a time intersection, the sum of resource consumptions of all nodes in computational power should not exceed the limit.
- Nodes working in parallel should be placed in the same time intersection and the length of the time intersection is the time needed by the nodes which take longer time to finish their task.
- In a time intersection if the resources consumption in memory and data exceed resource limits, there will be time penalties.
- A time intersection can be partitioned into several sub-intersections.

If a node is blocked until the other component finishes its task, a time intersection should be created to contain the node and be inserted after the time intersection of the time intersection of the other component.

4.3.2 Calculations

4.3.2.1 Resource limits

There are multiple methods that can be used to determine resource limits. Resource limits can be the maximum resources that the computer system provided, or else resource limits can be artificially decided. Here is an example of calculating the approximate resource limits on computational power and memory which requires time constraint and some performance specifications (processor computational power and RAM bandwidth) of the computer system as prerequisites.

Performance requirement: $\mathrm{T} = \mathrm{R}(\mathrm{p}, \mathrm{m}, \mathrm{d};\ \mathrm{e}) \leq\ \mathrm{T}_c = 50ms$

Computer specifications:

Processor power: 20,000MIPS

RAM bandwidth: 25.6Gbps(3.2GB/s)

The computational power rate of 20,000 MIPS (Millions of Instructions per Second), if we decide the module should not occupy over 2% of the computational power of the processor throughout the entire process, then the computational power limit p can be calculated as 20,000MIPS*2% which is 400MIPS.

For memory, we know that the time constraint is $\mathrm{T}_c = 50\mathrm{ms}$, then let time constraint be multiplied by RAM bandwidth: 50ms*3.2GB/s, we get 160MB. Because 160MB is approximately the maximum size of input/output that RAM can provide to processor in 50ms. So the resource limit m will be even smaller because 50ms might be partitioned by multiple time intersections.

It should be noted that resource limit m is the limit of the active memory which keeps interacting with the processor. It does not include all the memory usage.

Data resource limits depend on practical situation. For example, if the software system takes data from a sensor and it has no duplication for the data acquired, then the data resource coming from the sensor is only one.

4.3.2.2 Resource Allocation Configuration

Resource allocation configuration C_r is the set of resource allocations to each component in each time intersection. We denote r_{xij} as a resource allocation, here x denotes the type of the data, here i denotes the ith component, j denotes the jth time intersection. Because data resource consumption is calculated by the sum of resource need of each component, so resource allocation for data resource is denoted as $r_{di',j}$, where i' denotes the ith data type.

The resource consumption configuration of a module is represented asfollows:

$$C_r = \{(r_{p11}, r_{m11}), (r_{p21}, r_{m21}), (r_{p31}, r_{m31}), \dots, (r_{p12}, r_{m12}), (r_{p22}, r_{m22}), (r_{p32}, r_{m32}), \dots, (r_{pnk}, r_{mnk})\}$$
$$\cup \{(r_{d11}, r_{d21}, r_{d31}, \dots, r_{dn1}), (r_{d12}, r_{d22}, r_{d32}, \dots, r_{dn2}), (r_{d13}, r_{d23}, r_{d33}, \dots, r_{dn3}), \dots$$
$$\dots, (r_{d1k}, r_{d2k}, r_{d3k}, \dots, r_{dnk})\}$$

4.3.2.3 Length of Time Intersections

We denote t_{Ij} as the length of the time intersection, here i denotes the ith component working in the time intersection, j denotes the jth time intersection. t_{Ij} is calculated as follows:

$$t_{Ij} = \max[(t_{pij} + t_{mij})] + t_{dj}; i, j \in \{1,2,3, \dots, n\}$$

Where t_{pij} is the processing time of the ith component in the jth time intersection, t_{mij} is the time penalty brought by memory of the ith component in jth time intersection, t_{dj} is the time penalty caused by data resource in the jth time intersection.

In order to get t_{Ij}, we need to calculate t_{pij} , t_{mij}, and t_{dj}.

Calculate t_{pij}:

Given the resources (computational power) allocated r_{pij} , the resource consumption x, standard time t_s is

$$t_{pij} = t_s \cdot (x/r_{pij})$$

For example, set standard time t_s to 10ms, and resource consumption x to 2. If there is 1 resource allocated, $t_{pij} = 10\text{ms} \cdot (\frac{2}{1})$, then it needs 20ms to finish its

task. If there are 3 computational power unit allocated, $t_{pij} = 10\text{ms} \cdot (\frac{2}{3})$, then it needs about 6.67ms to finish its task.

Calculate t_{mij}:

In the resource allocation view, the memory consumption refers to the consumption of active memory in RAM or Processor cache (Fast type of memory) in the system. Nowadays, computers manage memories very efficiently and the speed of RAM and cache is so fast that the time needed for Read/Write data is trivial. In addition, memory resource limit is given by taking the bandwidth of memory into consideration. Therefore, we only consider memory and time relation in this way. As long as the memory needed by all components in the same time intersection does not exceed the memory limit s, the time penalty t_{mij} is equal to 0. Otherwise, we need to calculate t_{mij}. Because, when there is not enough RAM, a real computer uses virtual RAM, which is saved in the slower type of memory(HDD/SSD). Time penalty t_{mij} is calculated in this way.

Given the resources (memory) allocated r_{mij}, the resource consumption x, and the speed of virtual memory are represented in numbers without unit S_{vm}.

$$t_{mij} = 2 \cdot (\mathrm{x} - \mathrm{r}_{mij})/(S_{vm})$$

For example, let number 1 represent 1MB memory, the speed of virtual memory is 200MB/s, if memory consumption is 5, allocate memory is 3, then the time penalty is (5-3)/(200/s)*2=0.02s=20ms

Calculate t_{dj}:

Some modules may require data as a resource. If the data resource is less than the number of components in the same time intersection which wants to use the data, then there will be a time penalty which is calculated in this way:

Given the resources(data) limit d copies, the resource consumption x copies, and the function of time penalty on data resource f(n) which depends on where the data is and how the data is used, then

$$t_{dj} = \mathrm{f}(\mathrm{x} - \mathrm{d})$$

If there are multiple types of data as resources, then there time penalty is

$t_{dj} = MAX(t_{d1j}, t_{d2j}, t_{d3j}, \dots t_{dkj})$ where k is the number of types.

4.3.2.4 Response Time

After we know the length of each time intersection, the response time T of the model R is calculated as follows:

$$\mathrm{T} = \sum_{j=1}^{n} t_{lj} = \max[\left(\sum_{j=1}^{n} t_{pj} + \sum_{j=1}^{n} t_{mj}\right)] + \sum_{j=1}^{n} t_{dj}$$

4.3.3 Formal Proof: Resource consumption view ⇔ response model R

4.3.3.1 Computational Power and the Time Relation

Given that a module contains only one component A which consumes only computational power, it takes xMIPS of computational power rate to finish its task which responds to event e in 10ms(0.01s). Transform component A of resource consumption view into the node b in model R as follows, use number 1 to represent 1MIPS of computational power rate, write number x inside the node a to represente its computational power consumption, let standard time t_s = 10ms. Because there is only one node, draw only one time intersection area with node a in it.

The resource limit over the module is pMIPs

Allocate yMIPS of computational power to component A, hence

y is the resource allocate for node a, $\mathrm{y} \leq \mathrm{p}$

Denote: $\mathrm{T} = \mathrm{R}(\mathrm{p}, 0, 0;\ \mathrm{e})$

Resource consumption view ⇔ response model R:

Time needed for component A to finish its task:

$\mathrm{t} = 10(\mathrm{xMIPS}/\mathrm{yMIPS}) = 10\mathrm{x}/\mathrm{y}(\mathrm{ms})$

The response time of model R:

Because there is only one time intersection contains only one node,

$\mathrm{T} = t_{I1} = t_{p1} = 10\mathrm{x}/\mathrm{y}(\mathrm{ms})$

4.3.3.2 Memory and the Time Relation

Given that a module contains only one component B which consumes only memory, it takes at least xKB of memory to finish its task. Set the resource limit over the module to sKB. Set the virtual memory speed as zKB/s. Transform component B of resource consumption view into the node b in model R as follows, use number 1 to represent 1KB of memory, write number x inside the node b to presented its memory consumption. Because there is only one node, draw only one time intersection area with node a in it.

Allocate yKB of memory to component A, hence number y is the resource allocated for node b, $y \leq m$

Denote: $T = R(0, m, 0;\ e)$

Resource consumption view ⇔ response model R:

Time needed for component A to finish its task:

$$t = 2 \cdot (xKB - yKB)/(zKB/s) = 2 \cdot (x - y)/z(s)$$

The response time of model R:

Because there is only one time intersection contains only one node,

$$T = t_{I1} = t_{m1} = 2 \cdot \frac{x-y}{\frac{z}{s}} = 2 \cdot (x - y)/z(s)$$

4.3.3.3 Data Resource and Time relation

Given a module which consumes x copies of same type of data resource. Set the resource limit over the module to d copies. Transform resource consumption view of the module into model R as follows, find function t=f(n), where t is time and n is the difference between data needed and data sources provided. Because the data needed is counted as x copies, and data provided is counted as d copies, so $n = x - d$

Denote: $T = R(0, 0, d;\ e)$, where

Resource consumption view ⇔ response model R:

Time needed for module to finish its task:

$$t = f(n) = f(x - d)(s)$$

The response time of model R:

$$T = t_{Ij} = f(x - d)(s)$$

4.3.3.4 Multiple Resource Consumption

Given that a component X. X consumes computational power, memory, and data resources, the time needed for X to finish its task is time spend on computation plus penalty from memory and data resources shortage. Transform resource consumption view of X into model R. According to the proofed resource and time relations, the time needed for the component X to finish its task is equal to the time intersection where the component locates.

4.3.3.5 Parallelism

Given that a module includes multiple components A, B, C,…,N, these components work in parallel, hence the limit amount of resources are shared by those components. Transform the resources consumption view of the module into response model R in this way: for each component in resource consumption view, draw a corresponding node. Then there is node a,b,c,…,n. Components in resource consumption view work in parallel, so all nodes will be placed into the same time intersection. The sum of resource allocations for the each node is the limit amount of resource allocation.

Resource consumption view ⇔ response model R:

The time the module needed for making the response is the time needed by the component which takes longer time to finish its task. The response time T of response model R is the length of the time intersection which is the same as the time of the node which yields a longer time needed. According to previous proof of resource and time consumption relations, the time needed by the module to respond to the event is the same as the response time T of the response model R.

4.3.3.6 Dependency

Uni-denpendency

Given that a module includes only two components A and B, B depends on A that B can start if and only if A finished its task. Transform the resources consumption view of the module into response model R in this way: draw node a which corresponds to component A, and draw node b which corresponds to component B; if B depends on A, there will be a directed edge from the node a to the node b; the node a and the node be will be placed in two time intersections.

Resource consumption view ⇔ response model R:

The time the module needed for making the response is the sum of the time needed by component A and the time needed by component B. The response time T of response model R is the sum of lengths of the two time intersections. According to previous proof of resource consumption and time relations, the time needed by the module to respond to the event is same as the response time T of the response model R.

Multiple dependencies:

If a component depends on multiple other components, it can start working if and only if the latest depended component has finished its task. In this sense, multiple dependencies can be reduced to several uni-dependencies in parallel. According to previous proof of resource consumption and time relations, parallelism, and uni-denpendency, the time needed by the module to respond to the event is same as the response time T of the response model R.

Mutual dependencies (synchronization):

Given that a module includes only two components A and B, A and B depend on each that A and B start working and finish working at the same time. Transform the resources consumption view of the module into response model R in this way: draw node a which corresponds to component A, and draw node b which corresponds to component B. Because the time needed by A and B is exactly the same, according to previous proof of resource consumption and time relations, parallelism, and multiple dependencies, the time needed by the module to respond to the event is the same as the response time T of the response model R.

4.3.3.7 Working/Blocked

Given that a module includes only two components A and B, B is blocked until A finishes its task (B may or may not depend on A). Transform the resources consumption view of the module into response model R in this way: draw node a which corresponds to component A, and draw node b which corresponds to component B; draw another time intersection which follows after the time intersection of node a, and place b into the time intersection drawn. Because the components A and B work one by one and the nodes a and b are in two consecutive time intersections, according to previous proof of resource consumption and time relations, the time needed by the module to respond to the event is the same as the response time T of the response model R.

4.3.3.8 Add Component

Given a module with components, we add a new component into the module with a relation to other modules, then in the response model R there will be a new node added with relations to other nodes corresponds to relations between new components and other modules. According to previous proof of resource consumption and time relations, parallelism, dependencies and working/blocked, the time needed by the module to respond to the event is the same as the response time T of the response model R.

4.4 The Resource Allocation Optimization Problem

Decision Version

Given a response model R which responds to the event e and resources limits p, m, d. Does a resource allocation configuration C_r exist so that the response time T of the response model is less than a pre-determined time constraint T?

Denote: Is there a C_r for $\mathrm{T} = \mathrm{R}(\mathrm{p}, \mathrm{m}, \mathrm{d};\ \mathrm{e}) \leq \mathrm{T}_c$?

Optimization Version

Given a response model R which responds to the event e and resources limits p, m, d. What is the optimal resource allocation configuration C_r for minimizing the response time T?

Denote: What is the C_r for $\mathrm{T} = \min[\mathrm{R}(\mathrm{p}, \mathrm{m}, \mathrm{d};\ \mathrm{e})]$?

It should be noted that finding an optimized resource allocation is not the only approach for optimization, modification on the software architecture is also acceptable as long as the rules and principles mentioned in Section4.1 are obeyed.

4.5 An Example of Optimization Using Response Model R

4.5.1 Software and Hardware architecture

We have a software module which takes data from two sensors. The module processes the data and gives a response. Here the event e is the arriving of the data.

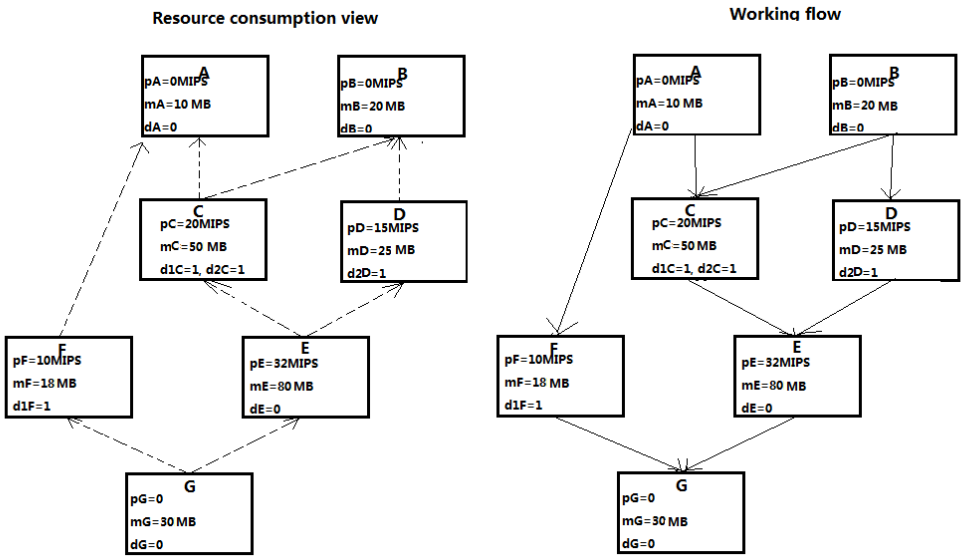
The software runs on a Von Neumann architecture computer which has resources of computational power and memory. The computer has the following performance specification:

Processor computational power: 1000MIPS

Memory Bandwidth: 25.6Gbps(3200MB/s)

Hard Drive Speed: 200MB/s

Here is the resource consumption view and working flow chart (Picture 4) of the module. To transform the resource consumption view into working flow, change the dash-lines which indicate dependencies into lines with arrows that indicateworking sequence.



Picture 4. The resource consumption view and working flow chart of the example

The software works in this way. Given standard time $t_s = 10ms$. Components A and B take data from sensors, thus they are the source of data resources. A receives data from sensor 1 and provides data resource d1. A needs 10MB of memory to save the data. B receives data from sensor 2 and provides data resource d2. B needs 20MB of memory to save the data. Component C receives output of A and B, so C needs 1 copy of both data resources. C also needs 20MIPS to process data and 50MB of memory to work. D only receives output from B, hence D only needs 1 copy of data resource d2. In addition, D needs 15MIPS computational power and 25MB of memory to work. E receives output from C and D, E does further processing by comparing the data processed by C and D. E needs no data resources but 32MIPS of computational power and 80MB of memory. F receives another flow of data from A and processes it. F needs 10MIPS of computational power and 18MB of memory to work. Eventually, F and E give their result to G. G needs only memory to make a response. In the resource consumption view, A and B work in parallel, C and D work in parallel, F and E work in parallel.

4.5.2 Define the Optimization Target

Now computer architecture determines that this module should take no more than 3% of computational power of the processor throughout the response process and the response time requirement is 50ms.

Let number 1 represent 1MIPS in computational power or 1MB in memory or 1 copy of data in data resources.

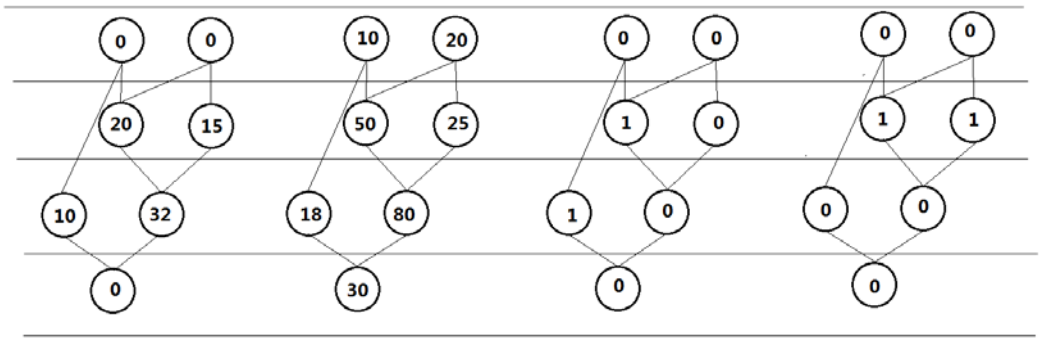
Then we can have the performance and efficient requirement defined as the following:

$$\mathrm{T} = \mathrm{R}(30, 80, \mathrm{d};\ \mathrm{e}) \leq \mathrm{T}_c, \mathrm{d} \in \{\mathrm{d1}, \mathrm{d2} | \mathrm{d1} = 1, \mathrm{d2} = 1\}$$

Here there are two types of data resources, namely d1 & d2

4.5.3 Build Response Model R and Calculate Response Time T

Transform the resource consumption view into the Triple-Graph of model R (Picture 5) Because there are 2 types of data resource, here there are actually four graphs)



Picture 5. The graphical representation of response model R

Here there are four time intersections.

Given a data resource penalty function: $\mathrm{f(n)} = 20\mathrm{n}$

Define an arbitrary resource allocation configuration

$$\mathrm{C}_r = \{(0,10),(0,20),(17,50),(13,25),(7,18),(23,62),(0,30)\} \cup \{(0,0),(1,1),(1,0),(0,0)\}$$

Calculate lengths of time intersections:

$t_{I1} = 0$

$$t_{I2} = \max\left[\left(t_{p12} + t_{m12}\right),\left(t_{p22} + t_{m22}\right)\right] + t_{d12} + t_{d22}$$
$$= \max\left[\left(\left(\frac{20}{17}\right)10ms + 0\right),\left(\left(\frac{15}{13}\right)10ms + 0\right)\right] + 0 + f(2-1)$$
$$= 31.8ms$$

$$t_{I3} = \max\left[\left(t_{p13} + t_{m13}\right),\left(t_{p23} + t_{m23}\right)\right] + t_{d13} + t_{d23}$$
$$= \max\left[\left(\left(\frac{10}{7}\right)10ms + 0\right),\left(\left(\frac{32}{23}\right)10ms + 2\cdot(80-62)/(200/\mathrm{s})\right)\right]$$
$$+ 0 + 0 = 193.9ms$$

$t_{I4} = 0$

Response time T of model R: $\mathrm{T} = t_{I2} + t_{I2} + t_{I4} + t_{I4} = 225.7ms$. This response time does not meet the requirement.

4.5.4 Optimization

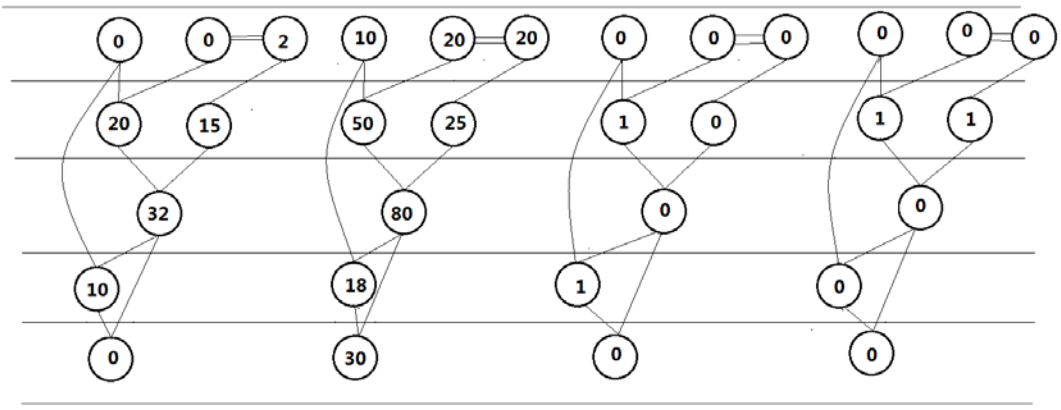
According to the previous calculation, we can find out that the main problem is the memory overflow in the third time intersection. In addition, in the second

time intersection, there are two components that want data resource type 2. This also generates a time penalty.

Make the following changes to response model R:

1. Introduce node 3 which has a mutual dependency relationship with node 2 in the first time intersection. What it does is duplicating the data of node 2. It takes 2(units) computational power and 20(units) memory to work.
2. Place the node 1 originally in the third time intersection into a newly created fourth time intersection by setting an artificial dependency (one is blocked until the other one finishes its task) on the node staying in the third time intersection. The original fourth time intersection is now the fifth time intersection.

By applying the changes we can get an optimized model R'



Picture 6. The graphical representation of optimized response model R'

Define a new resource allocation configuration

$$C_r = \{(0,10),(0,20),(30,20),(17,50),(13,25),(30,80),(30,18),(0,30)\} \cup \{(0,0),(1,2),(1,0),(0,0)\}$$

Now we calculate the response time T' after optimization.

$$t_{I1} = \left(t_{p31} + t_{m31}\right) + 0 = \left(\left(\frac{2}{30}\right)10ms + 0\right) = 1.5ms$$

$$t_{I2} = \max\left[\left(t_{p12} + t_{m12}\right), \left(t_{p22} + t_{m22}\right)\right] + t_{d12} + t_{d22}$$
$$= \max\left[\left(\left(\frac{20}{17}\right)10ms + 0\right), \left(\left(\frac{15}{13}\right)10ms + 0\right)\right] + 0 + 0 = 11.8ms$$

$$t_{I3} = \left(t_{p13} + t_{m13}\right) + t_{d13} = \left(\left(\frac{32}{30}\right)10ms + 0\right) + 0 = 10.7ms$$

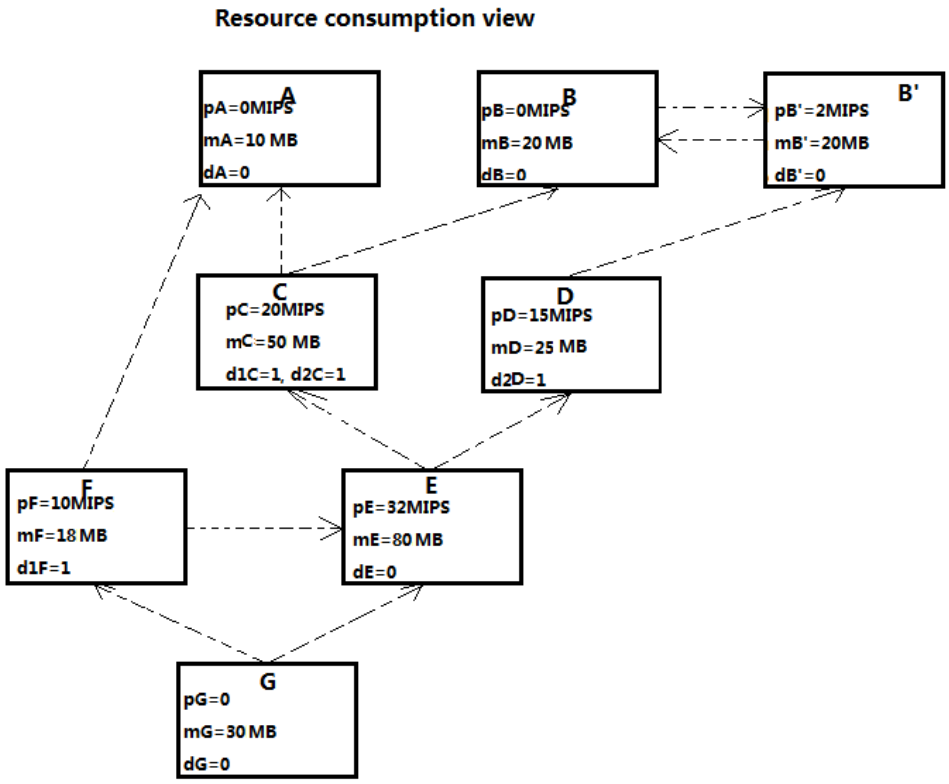
$$t_{I4} = \left(t_{p14} + t_{m14}\right) + t_{d14} = \left(\left(\frac{10}{30}\right)10ms + 0\right) + 0 = 3.3ms$$

$$t_{I5} = \mathbf{0}$$

Response time T' of optimized model R: $T' = t_{I2} + t_{I2} + t_{I4} + t_{I4} + t_{I5} = 27.3ms$. This response time $T' < T_c$

4.5.5 Result of Optimization

The method for transforming the optimized Model R' back to a resource consumption view is just the reverse procedure of transforming the resource consumption view to response model R. Here we can see the optimized architecture (Picture 7).



Picture 7. The resource consumption view of the optimized architecture

It is notable that this example only shows the procedure for optimizing at the software architectural level using response model R. Detailed methods for optimization on the response model R may vary according to the details of the software and hardware architectural. Moreover, there could be more methods for finding the suitable resource allocation configuration C_r.

5 CONCLUSION

The response model R can be used as a power tool for optimizing software efficiency or performance. The optimization is done by optimizing resource allocation and software architecture.

There is a general method for transform views of software architecture to a response model R. To achieve the transformation, we need to transform other type of views into a resource consumption view in the first place. Once the resource consumption view is formed, it can be further transformed into a response model R. Optimization is carried out/implemented on the response model R. Afterwards, the response model R will be transformed back to software views.

Response model R has two great properties which make it even more powerful. One property is that it supports scaling. In the practical optimization, if we want to optimize a large software system, we can decompose the software system into modules, modules decompose to sub-modules. A component in resource consumption view might be a sub-module which has its own components. Each level of decomposition can be mapped to a resource consumption view, hence can be optimized through response model R. The other property which contributes to the powerfulness of the response model R is that it can be turned into a resource allocation optimization problem. The optimization problem including its decision version can possibly be solved through existing methods for other optimization problems through reduction.

All in all, the method using response model R has great potential in enhance software performance and efficiency, hence bringing benefits to both software developer and end users.

REFERENCES

Advanced Micro Devices, Inc. (2011) *APU101 All about AMD Fusion Accelerated Processing Units and How They're Changing, Well Everything.* pp. 35

Bass, L. Clements, P. and Kazman, R. (2003) *Software Architecture in Practice*, Boston: Addison-Wesley Publishing Company, Inc.

Gamma, E. Helm, R. Johnson, R. and Vlissides, J. (1995) *Design Patterns Elements of Reusable Object-Oriented Software*, Massachusetts: Addison-Wesley Publishing Company, Inc.

Riley, H. (1987) *The von Neumann Architecture of Computer Systems* California: Computer Science Department, California State Polytechnic University

Riley, J. (2006) *Writing Fast Programs: A Practical Guide for Scientist and Engineers.* Cambridge: Cambridge International Science Publishing Ltd.

Wirth, N. (1995) *'A Plea for Lean Software.'* Computer, 28(2):64-68