

THE SCALABILITY METRIC, BASED ON COST-EFFECTIVENESS IN

DISTRIBUTED SYSTEMS USING FUNDAMENTAL LAWS

BY

GYASI EMMANUEL KWABENA, BSC COMPUTER SCIENCE

A Thesis Submitted to the Department of Computer Science Kwame Nkrumah University

of Science and Technology in partial fulfilment of the Requirements for the degree

Of

MASTER OF PHILOSOPHY

IN

COMPUTER SCIENCE

College of Science

Department of Computer Science

2018

DECLARATION

I hereby declare that this thesis is my own work towards the MPhil and that, to the best of my knowledge, it contains no material previously published by another person nor material which has been accepted for the award of any other degree of the University, except where due acknowledgement has been made in the text.

Emmanuel Kwabena Gyasi

(PG4722615)

Student Name & ID

Signature

Date

Certified by:

Dr. Dominic Asamoah

Supervisor(s) Name

Signature

Date

Certified by:

Dr. Michael Asante

Head of Dept. Name

Signature

Date

ABSTRACT

Today's computer systems are more complex, more rapidly evolving, and more essential to the conduct of business than those of recent past. The complexity becomes more rigid in the case of distributed systems. A distributed system is a collection of independent computers that appear to its users as a single coherent system. A distributed system should be deployable in a wide range of scales, in terms of numbers of users and services, quantities of data stored and manipulated, rates of processing, numbers of nodes, geographical coverage, and sizes of networks and storage devices. The derived scalability metric of this thesis is based on cost effectiveness, in which the effectiveness is a function of the system's throughput and its Quality of Service. It is a strategy based scalability metric that generalizes the well-known metrics for scalability of parallel computations to describe heterogeneous distributed systems. Scalability is measured by the range of scale factors that gives a satisfactory value of the metric, since a good scalability is a joint property of the initial design and the scaling strategy. What makes this derived metric unique is the fact that, it separates the impact of throughput and response time on the metric, formalizing the notation of a scaling strategy, introducing QoS evaluation and more also, introducing formal scalability enablers which are optimized at each scale factor. Throughput curves for all systems with bottleneck demand D_{max} are constrained to lie below the line $1/D_{max}$. If one needs to improve the performance further than this limit, then it is necessary to reduce the demand at the bottleneck center somehow.

ACKNOWLEDGEMENT

What shall I render unto the Lord for all his goodness and mercies towards me, and for seeing me through a successful completion of this course? "Awurade meda wo ase". I will say thank you Lord. There are many people that I must thank who, one way or another, contributed to the success of my education as well as this thesis. I would like to thank my supervisor Dr. Dominic Asamoah under whose constant supervision and guidance I studied and undertook this project. Your immense contribution coupled with relentless effort to this thesis is outstanding. May you forever remain blessed Dr. My sincere thanks goes to my mother, Madam Susana Mintah for her wonderful contribution towards my education, I love you mum. I also thank my dependable friend, Miss Olivia Kondo of Ghana Meteorological Agency, Sunyani for her stupendous assistance. Again, I want to express my profound gratitude to Prof. Ibok Oduro (Dean, College of Science, KNUST) for her cooperation during my tenure as College Board member representing Graduate Students in College of Science. Dr. Michael Asante (Head of Computer Science Department, KNUST), Dr. Acquah (Fmr. Head of Computer Science Department, KNUST), Dr. Missah, Mr. J. K. Panford, Mr. Davis (all of Computer Science Department, KNUST). Finally, my sincere gratitude goes to my family, course mates, as well as my cherished friends. I say, I love you all and I appreciate your effort. May our good Lord bless us all.

TABLE OF CONTENTS

DECLARATION	i
ABSTRACT.....	ii
ACKNOWLEDGEMENT	iii
TABLE OF CONTENTS.....	iv
LIST OF TABLES	viii
LIST OF FIGURES	ix
LIST OF ABBREVIATIONS.....	x
CHAPTER ONE	1
INTRODUCTION.....	1
1.1 Background of Study	1
1.2 Motivation	2
1.3 Problem Statement	3
1.4 Research objectives	3
1.5 Research Questions	4
1.6 Organization of the Thesis	4
CHAPTER TWO	5
LITERATURE REVIEW	5
2.0 Introduction	5
2.1 Review of Available Metrics.....	5
2.1.1 Speedup (S).....	7
2.1.2 The Relative	8
2.1.3 The Real.....	8
2.1.4 The Absolute.....	8
2.1.5 The Asymptotic.....	8
2.1.6 The Asymptotic relative	9
2.1.7 Asymptotic speedup	11

2.1.8 Asymptotic real speedup(n) = $\Theta(n^a/\log n)$	11
2.1.9 Absolute speedup	11
2.2 Real speedup	12
2.2.1 Analytical realspeedup (n,P) = $\Theta(P)$	13
2.3 Relative speedup	14
2.4 Scaled speedup	15
2.5 Sizeup	15
2.6 Limits to speedup.....	16
2.7 Isoefficiency function	16
2.8 Efficiency (E).....	21
2.9 Summary	24
CHAPTER THREE	26
METHODOLOGY.....	26
3.1 Introduction.....	26
3.1.1 The Basic Quantities.....	26
3.1.2 The Little's Law.....	28
3.2 Using The Fundamental Laws To Derive Bounding Equations.....	31
3.2.1 The Asymptotic Bounds.....	33
3.2.2. The Transaction Workloads	33
3.2.3. The Batch and Terminal Workloads	35
3.3.4 Balanced System Bounds	38
3.3.5 The Algorithm for Closed Model Asymptotic Bound.....	40
3.4 The Scalability Metric	43
3.4.1 Generating The Measures And Applying It To Idealized Models	46
3.4.2 Model A: General Speed-Scaled Open System with Proportional Costs.....	46
3.4.3 Model B: General Speed-Scaled Closed System with Fixed User Population	47
3.4.4 Model C: Closed Balanced Ideal Queueing Network, Scaled in Users and Speed	47
3.4.5 Model D: Asymptotic General Closed System Scaled in Users and Speed	48
3.4.6 Model E: System with a Single Non-scalable Bottleneck and Increasing Users	48

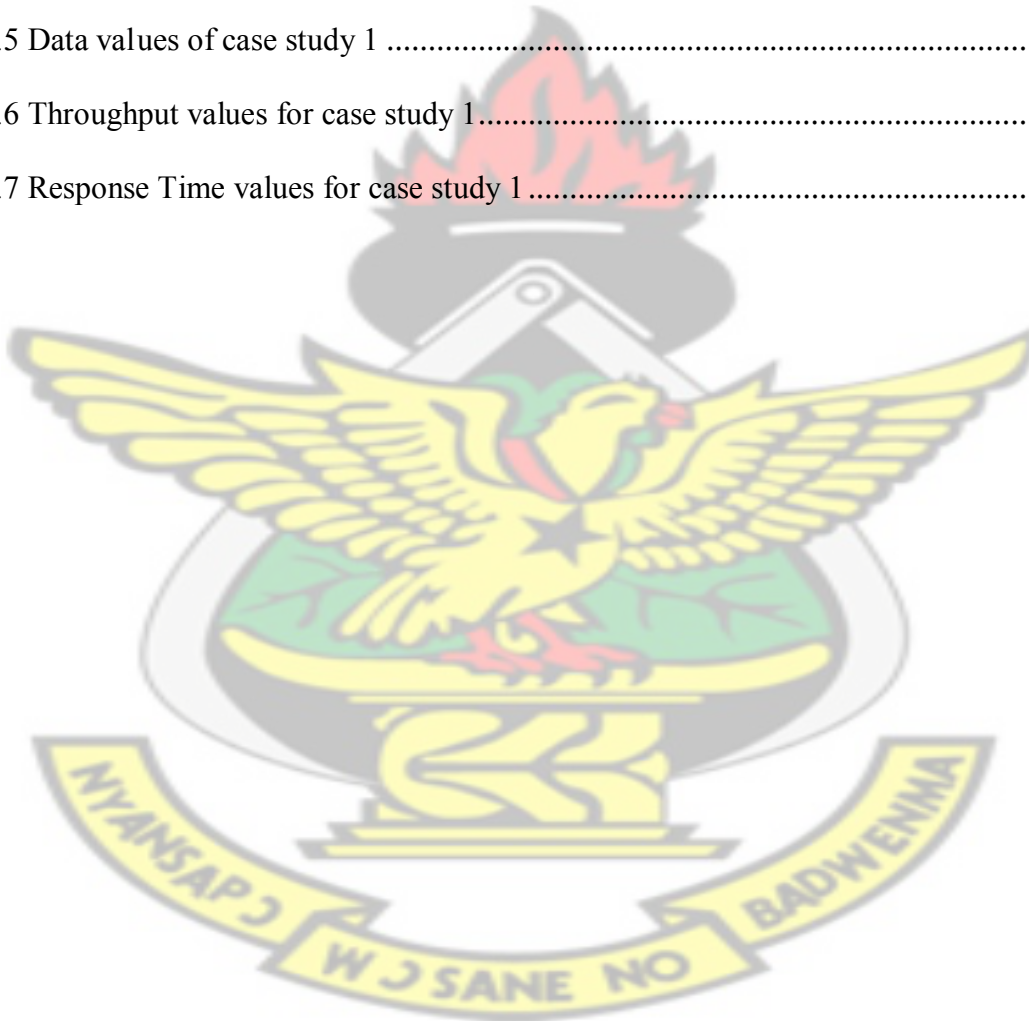
3.4.7 Model F: Closed Balanced System Scaled by Replication of Servers and User Population	49
3.4.8 Model G: The Effect of Overhead Costs.....	49
3.4.9 Model H: A Closed System with Scaled Population and Target Response Times.....	50
3.4.10 Model I: A Single Scaled Open Multiserver Queue.....	51
3.4.11 Algorithm for Calculating Scalability Bound	52
3.5 Overview of The Connection-Management System.....	54
3.6 Using the Metric to Scale the Connection Management System	57
3.6.1 How to obtain the values for the Connection Management system.....	59
3.6.2 Calculating the Scalability Bound of The Connection Management System	62
3.6.3 Obtaining the measures for the Scalability bound	63
3.7 Applying The metric to Asymptotic Bounds (Case Study 1)	64
3.8 Calculating the Throughput bound and Average Response Time of the systems	67
3.8.1 Case Study 2	70
Table 3.8 Data values for case study 2.....	70
3.8.2 Procedure for scaling the system.....	71
3.9 Overview of the Call Processing System.....	72
3.9.1 Applying the metric to call processing system	72
3.9.2 The Call Processing System Architecture.....	73
3.9.3 The Hardware Platform and The Processing Steps	74
3.9.4 Software Organization and Layered Model	75
3.9.5 Scaling strategy for the call processing system.....	76
3.9.6 How to obtain the values for the call processing system.....	76
CHAPTER FOUR.....	81
RESULTS AND DISCUSSION.....	81
4.1 Introduction.....	81
4.2 Results and Discussions of Case Study 1.....	81
4.3 Results and Discussions of Case Study 2.....	83
4.3 Results and Discussions of the Connection Management System.....	85

4.4 Results and Discussions of the Call Processing System.....	87
CHAPTER FIVE.....	91
CONCLUSION AND RECOMMENDATIONS	91
5.1 Conclusion.....	91
5.2 Recommendations	92
REFERENCES.....	93



LIST OF TABLES

Table 2.1. Speedup metric definitions	10
Table 2.2: Performance Metrics	21
Table 3.1 Summery of the Notations.....	31
Table 3.2 The Fundamental Laws.....	31
Table.3.3 Summary of Asymptotic Bound Equations	37
Table.3.4 Summary of Balanced System Bounds Equations	40
Table 3.5 Data values of case study 1	65
Table 3.6 Throughput values for case study 1.....	69
Table 3.7 Response Time values for case study 1	69



LIST OF FIGURES

Fig 2.1: Overheads in a Parallel System.....	19
Fig.3.1 An Abstract System	27
Fig. 3.2a Terminal Throughput Graphical Representation of the Equations	42
Fig. 3.2b Terminal Response Time Graphical Representation of the Equations	43
Fig. 3.3The connection management System infrastructure	56
Fig. 3.4 - Case Study 1 Model Diagram.....	66
Fig. 3.5 Call Processing System's Hardware infrastructure.....	74
Fig. 3.6 Layered model for the call processing software, including the processes and interaction	76
Fig. 4.1a Asymptotic Bounds in the Case Study 1 showing the Throughput	82
Fig. 4.1b Asymptotic Bounds in the Case Study1 showing response Time	83
Fig. 4.2 Graph of the effect of various changes	85
Fig. 4.3a Scalability bound for the connection management system.....	86
Fig. 4.3b Scalability metric for the connection management system	87
Fig. 4.4a A Graph of Productivity vrs Scale factor.....	89
Fig. 4.4b A Graph of throughput vrs Scale factor	89
Fig. 4.4c A Graph of Scalability vrs Scale factor	90

LIST OF ABBREVIATIONS

ATM: Asynchronous Transfer Mode

CPU: Central Processing Unit

CRM: Customer Relation Management

GHC: Ghana Cedis

IP: Internet Protocol

QoS: Quality of Service

RPC: Remote Procedure Call

TCP: Transfer Control Protocol

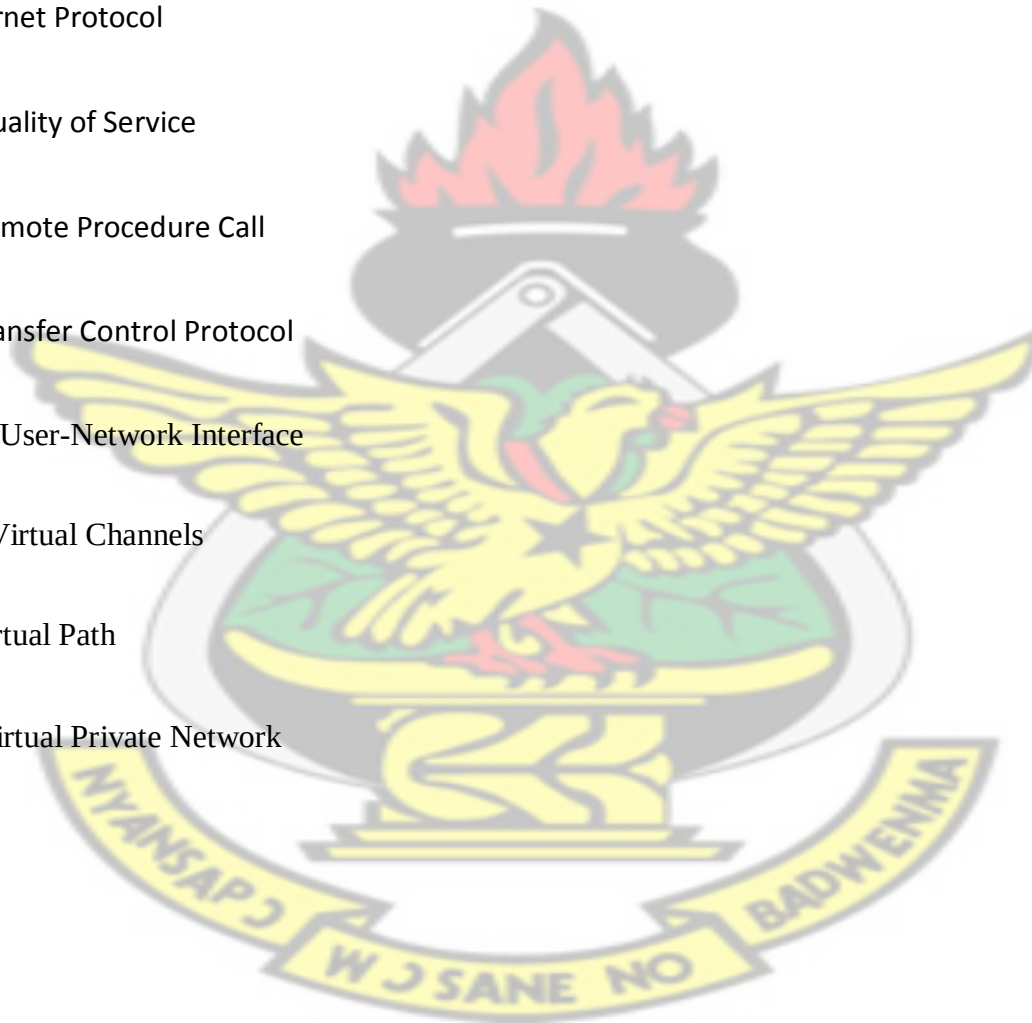
UNI's: User-Network Interface

VC's: Virtual Channels

VP: Virtual Path

VPN: Virtual Private Network

KNUST



CHAPTER ONE

INTRODUCTION

1.1 Background of Study

Today's computer systems are more complex, more rapidly evolving, and more essential to the conduct of business than those of even a few years ago. The result is an increasing need for tools and techniques that assist in understanding the behavior of these systems. Such an understanding is necessary to provide intelligent answers to the questions of cost and performance that arise throughout the life of a system. Right from Scientific institutions, Computer manufacturing industries, businesses of all kinds, educational institutions (Universities), software programmers to individuals, the issue of scalability remains the number one priority in terms of manufacturing and the use of computer systems. This question is of great significance to the organizations involved, because of its potentially serious repercussions from incorrect answers. Unfortunately, this question is also complex as correct answers are not easily obtained. In considering questions such as these, one must begin with a thorough grasp of the system, the application, and the objectives of the study. With this as a basis, several approaches are available. One is the use of intuition and trend extrapolation. To be sure, there are few substitutes for the degree of experience and insight that yields reliable intuition. Unfortunately, those who possess these qualities in sufficient quantity are rare. Another is the experimental evaluation of alternatives. Experimentation is always valuable, often required, and sometimes the approach of choice. It is also expensive and often prohibitively so. A further drawback is that an experiment is likely to yield accurate knowledge of system behavior under one set of assumptions, but not any insight that would allow generalization. These two approaches are in some sense at opposite extremes of a spectrum. Intuition is rapid and flexible, but its accuracy is sketchy because it relies on

experience and insight that are difficult to acquire and verify. Experimentation yields excellent accuracy, but is laborious and inflexible. Between these extremes lies a third approach, the general subject of this thesis.

1.2 Motivation

The world has now become a global village through internet and inter-connectivity. Individuals, organizations, governments, businesses to mention but a few in one way or the other connect or communicate remotely. Businesses have spread up to the extent of defying geographical boundaries while individuals and organizations do transact business and at the same time store their data remotely. Typical examples that readily come in mind are cloud computing. These activities are made possible because the system is distributed. A distributed system is a collection of independent computers that appear to its users as a single coherent system. In other words, a distributed system is a collection of independent computers that are used jointly to perform a single task or to provide a single service (Tannenbaum and Steen, 2016). Most distributed systems are scalable including present and future applications of which web based distributed systems is no exception. Typical applications, e-commerce, multimedia news services, distance learning, remote medicine, enterprise management, and network management are also some of the examples of distributed services. A distributed system should be deployable in a wide range of scales, in terms of numbers of users and services, quantities of data stored and manipulated, rates of processing, numbers of nodes, geographical coverage, and sizes of networks and storage devices. As businesses grow, the systems that support their functions also need to grow to support more users, process more data, or both. As they grow, it is important to maintain their performance in terms of responsiveness or throughput (Smith and Williams 2002). Poor performance in these applications often translates into appreciable costs.

Customers will often shop elsewhere rather than endure long waits. Slow responses in CRM applications mean that more customer-service representatives are needed. And, failure to process financial trades in a timely fashion can result in statutory penalties as well as lost customers. Despite its importance, scalability is poorly understood and few organizations understand how to quantitatively evaluate an application's scalability. As a result, they often make assumptions about the scalability of their software. If wrong, these assumptions can be costly.

1.3 Problem Statement

Many scaled systems suffer from the problem of maintaining productivity and that of delay in transmitting data from one system to another. In the field of telecommunication industry, frequent call drops coupled with high cost of managing distributed networks is a major headache to the managers in telecommunication business as more often than not, scaled networks performance does not commensurate with the cost incurred by scaling the systems, due to unavailable well laid down scalable metrics.

1.4 Research objectives

The objectives of this research are as follows

1. To have metrics that will maintain productivity as the system is scaled so as to enhance call-processing system and make it scalable in order to support large number of callers per hour at an appreciable throughput and at the same time curtail frequent call drops.
2. To come out with an efficient scalability metrics that ensures cost-effectiveness in distributed systems as the system is been scaled.
3. To have a detailed well laid down metric that even an individual can follow in order to scale his/her system .

1.5 Research Questions

1. How can call processing systems be scalable enough to handle the required number of users and devices and at the same time be resilient enough to handle various network and application outages or failures?
2. How can the cost of scaling systems commensurate with productivity?
3. What efficient metrics needs to be adopted?

1.6 Organization of the Thesis

The thesis is divided into five chapters spanning from chapter one up to chapter five. The chapter one deals with the overview of the thesis as it explains the rationale behind the chosen topic as well as the reasons why this research is worth carrying out. This chapter also deals with the introduction of the problem, which this thesis offers a solution to. The second chapter consists of the literature written in the field of scalability which deals closely with the topic of this thesis. The third chapter takes care of the methods and various laws termed in here as the fundamental laws which is used to generate the various functions for scaling up or down a given system. This chapter also sees a detailed discussion of bounding equations, algorithms, and that of two case studies of the system used by the Ghana Meteorological Agency being analyzed. It further explains into details, the procedure for scaling a system and the full calculation of scalability. It again looks at applying the metric to an idealized system. The chapter spends some time in overview of the call processing system. Chapter four presents the results and discussion of the research. Finally, chapter five serves as a logical end to the thesis outlining the important concepts covered in it and proposing possible future work in the field of scalability.

CHAPTER TWO

LITERATURE REVIEW

2.0 Introduction

We often hear of students, individuals and even system administrators complaining of their systems in one way or the other not functioning to their satisfaction and most of them helplessly resort to upgrade their system by simply adding memory or storage space. While most of these people do not achieve their intended purposes and at the same time waste their monies, others too end up damaging their systems. Different scalability metrics have been developed for massively parallel computation, to evaluate the effectiveness of a given algorithm running on different sized platforms, and to compare the scalability of algorithms. These metrics assume that the program runs by itself, on a set of k processors with a given architecture, and that the completion time T measures the performance.

2.1 Review of Available Metrics

Performance metrics such as speedup (Amdahl, 1967), scaled speedup (Gustafson, et al, 1988), sizeup (Sun and Gustafson, 1991), experimentally determined serial fraction (Karp and Flatt, 1990), and isoefficiency function (Kumar and Rao, 1987) have been proposed for quantifying the scalability of parallel systems. While these metrics are extremely useful for tracking performance trends, they do not provide adequate information needed to understand the reason why an algorithm does not scale well on an Architecture. An understanding of the interaction between the algorithmic and architectural characteristics of a parallel system can give us some fair idea. Studies undertaken by Kung (Kung, 1980) and Jamieson (Jamieson, et al, 1987) help identify some of these characteristics from a theoretical perspective, but that one too does not provide any means of quantifying their effects. Several performance studies address issues such

as latency, contention and synchronization. The limits on interconnection network performance (Agarwal, 1991), (Pfister and Norton, 1985) and the scalability of synchronization primitives supported by the hardware (Anderson, 1990), (Mellor-Crummey and Scott, 1991) are examples of such studies undertaken over the years. While such issues are extremely important, it is necessary to put the impact of these factors into perspective by considering them in the context of overall application performance. There are studies that use real applications to address specific issues like the effect of sharing in parallel programs on the cache and bus performance (Eggers and Katz, 1989) and the impact of synchronization and task granularity on parallel system performance (Chen, et al, 1990). (Cypher, et al, 1993), identify the architectural requirements such as floating point operations, communications, and input/output for message-passing scientific applications. Rothberg et al.(Rothberg, et al, 1993) conduct a similar study towards identifying the cache and memory size requirements for several applications. However, there have been very few attempts at quantifying the effects of algorithmic and architectural interactions in a parallel system. The time taken by an algorithm to execute on a single processor is called the sequential execution time and is denoted by T_1 . The execution time of the corresponding parallel algorithm on p identical processors is called the parallel execution time and is denoted by T_p . A parallel algorithm incurs several overheads during execution. These include overheads due to idling, communication, and contention over shared data structures. The sum total of time spent by all processors doing work which is not done by the sequential algorithm is termed as the total overhead and is denoted by T_0 . In general, T_0 is a function of problem size and the number of processors. Since the sum total of time spent by all processors is pT_p , and the total overhead is T_0 , we can deduce that:

$$pT_p = T_1 + T_0$$

$$T_p = \frac{T_1 + T_0}{p}$$

2.1.1 Speedup (S)

The speedup (S) obtained from a parallel system is defined as the ratio of the sequential execution time to the parallel execution time. Therefore,

$$S = \frac{T_1}{T_p} = \frac{pT_1}{T_1 + T_0}$$

Parallel computers promise the following enhancements over their sequential counterparts, each of which leads to a corresponding scaling strategy: 1) the number of processing elements is increased enabling a potential performance improvement for the same problem (**constant-problem size scaling**); 2) other system resources like primary and secondary storage are also increased enhancing the capability to solve larger problems (**memory-constrained scaling**); 3) due to the larger number of processing elements, a much larger problem may be solved in the same time it takes to solve a smaller problem on a sequential machine (**time-constrained scaling**). Speedup captures only the constant problem size scaling strategy. It is well known that for a problem with a fixed size, the maximum possible speedup with increasing number of processors is limited by the serial fraction in the application (Amdahl, 1967). But very often, parallel computers are used for solving larger problems and in many of these cases the sequential portion of the application may not increase appreciably regardless of the problem size (Gustafson, et al, 1988) yielding a lower serial fraction for larger problems. In such cases, memory-constrained and time-constrained scaling strategies are more useful. In parallel computing, as in its serial counterpart, time and memory are the dominant performance metrics.

Between alternate methods that use differing amounts of memory, any user would prefer the faster method, provided enough memory is available to run both methods. That is, there is no advantage to using less memory than might be made available to the application unless the use of less memory reduces execution time or cost. So, the execution time or time complexity remains an important metric. Also, because of the lack of program portability, in considering the complexity of a parallel algorithm we must really talk about the analysis of an algorithm with respect to particular parallel computer architecture. We must talk therefore about the complexity of the system. The desire to know how much faster an application runs on a parallel computer has made performance measurements in the parallel domain more complex. What benefits derive from the use of parallelism? How much speedup results? While there is general agreements that speedups is the ratio of serial execution time to parallel execution time, there are diverse definitions of serial and parallel execution times. This diversity results in at least five different definitions of speedup and they are as follows

2.1.2 The Relative: uses the execution time of the parallel program when run on a single node of the parallel computer for the serial time.

2.1.3 The Real: compares the parallel execution time with the time needed by the fastest serial algorithm or program for the application executing on a single node of the parallel computer.

2.1.4 The Absolute: compares parallel execution time with the fastest sequential algorithm run on the fastest serial computer.

2.1.5 The Asymptotic: compares the execution time of the best serial algorithm for a particular problem with the asymptotic complexity of the parallel algorithm, assuming the parallel computer has all the processors it can gainfully use.

2.1.6 The Asymptotic relative: uses the asymptotic time complexity of the parallel algorithm when run on a single processor. When computing relative, real, or absolute speedup, we could use analytical time complexities or actual measured run time.

The former provides the analytical speedup of the parallel system. The latter yields the system's measured speedup. Because analytical runtime measurements are usually accurate only within a constant factor, the analytical runtime of the best serial algorithm is the same on all serial computers modeled by random access machines (within a constant factor).



Table 2.1. Speedup metric definitions

(I = Problem instance, P = number of processors; Q =Parallel program; n= Size of I)		
METRIC	FORMULA	COMMENTS
Relative Speedup (I,P)	$\frac{\text{time to solve } I \text{ using Program } Q \text{ and } I \text{ Processor}}{\text{time to solve } I \text{ using Program } Q \text{ and } P \text{ Processors}}$	Depends on the characteristics of the instance I being solved as well as the size P of the parallel computer.
Real Speed up (I,P)	$\frac{\text{time to solve } I \text{ using best Serial Program and } I \text{ Processor}}{\text{time to solve } I \text{ using Program } Q \text{ and } P \text{ Processors}}$	The fastest Algorithm might not be known and no single algorithm might be fastest in all instance for some applications, so the runtime of the sequential algorithm is most frequently used in practice.
Absolute Speedup (I,P)	$\frac{\text{time to solve } I \text{ using best serial Program and } I \text{ fastest Proc}}{\text{time to solve } I \text{ using Program } Q \text{ and } P \text{ Processors}}$	Can also use the sequential algorithm most often used in practice.
Asymptotic real speedup (n)	$\frac{\text{Asymptotic complexity of best serial Algorithm}}{\text{Asymptotic complexity of } Q \text{ using as many Processors as ne}}$	For problems such as sorting where the asymptotic complexity is not uniquely characterized by the instance size n, we use the worst-case complexity
Asymptotic relative speedup (n)	$\frac{\text{Asymptotic complexity of } Q \text{ using } I \text{ Processor}}{\text{Asymptotic complexity of } Q \text{ using as many Processors as ne}}$	Like asymptotic real speedup, it does not rely on the number of processors available in the parallel system, because we assume this number is unbounded.

2.1.7 Asymptotic speedup

Consider a matrix multiplication algorithm that multiplies two $n \times n$ matrices on an $n^3/\log n$ processor hypercube in $\Theta(\log n)$ time. Because this is the fastest the algorithm can run even if more processors are available, its asymptotic complexity, assuming the availability of as many processors as can be used, is $\Theta(\log n)$. Its asymptotic real speedup depends on the complexity of the best serial algorithm, which changes with time as faster matrix multiplication algorithms are discovered. Hence, except for problems with known asymptotically optimal algorithms, asymptotic speedup is not time-invariant. If $\Theta(n^a)$ is the complexity of the fastest serial matrix multiplication algorithm currently known,

2.1.8 Asymptotic real speedup(n) = $\Theta(n^a/\log n)$.

Unfortunately, the goodness of our parallel multiplication algorithm or system (as measured by its asymptotic real speedup) declines as years pass and asymptotically superior serial algorithms emerge. This decline has nothing to do with deterioration in the parallel algorithm with respect to its execution time and memory requirements. When the parallel matrix multiplication algorithm runs on a single processor, its complexity is $\Theta(n^3)$. The asymptotic relative speedup of this parallel matrix multiplication system thus is $\Theta(n^3/\log n)$.

2.1.9 Absolute speedup

Because we have equated analytical absolute speedup to analytical real speedup, we are concerned only with measured absolute speedup in this section. In practice, we would expect this to be a critical measure because it tells us how much faster we can solve our problem instance on the target parallel computer versus the fastest serial alternative. Unfortunately, very few experimental studies of speedup report this quantity. Besides difficulties associated with getting access to the fastest serial alternative, this alternative changes with time as faster serial

algorithms are developed or faster serial computers are manufactured. Certainly, in the interval between conducting experiments and publishing their results, the speed of the fastest serial computer, hence the absolute speedup, would have changed! Because many parallel computers don't use the fastest available processors, these parallel computers exhibit an absolute speedup of less than one, even when P is modestly large (say 100). For example, a 64-processor nCube1 computer takes approximately six times as long as a single-processor Cray-2 for image-template matching on a 256x256 image using a 16x16 template. This is despite the fact that the nCube1 program achieved a relative speedup of almost 58. (However, a 64-processor nCube1 provided a tenfold cost-performance advantage over the Cray-2).

2.2 Real speedup

Analytical real speedup is generally easy to determine because one need not write a program and measure execution time. There are two approaches to computing the analytical real speedup of a parallel system, one using the asymptotic complexities of the serial and parallel algorithms, and the other using the number of operations, or workload. Generally, we regard workload (total amount of computation done by the program) and runtime as related by the speed of the computer. At times, we will need to distinguish between workload and time because their assumed relationship may not hold. For example, the effective speed of the computer may vary with workload because larger workloads may require more memory and eventually the use of slower secondary memory. Furthermore, as total workload changes, its mix might change, causing a change in effective speed. As an example of using asymptotic complexities, we can add two $n \times n$ matrices in $\Theta(n^2/P)$ time using a shared memory computer with $P \leq n^2$ processors. The fastest serial algorithm to add the two matrices has complexity of $\Theta(n^2)$. The analytical real speedup of the parallel shared-memory algorithm then is:

2.2.1 Analytical realspeedup $(n,P) = \Theta(P)$.

Notice the difference between this and asymptotic speedup. In the latter, we compute the asymptotic complexity assuming an unbounded supply of processors. As an example of using operation counts or workload, consider the problem of reading two $n \times n$ matrices from a disk, adding them, and writing the result back to disk. Suppose the time to read and write a matrix is n^2 units and that the time to add them using a serial algorithm is n^2/S , where S is the processor speed. Furthermore, suppose that the parallel addition algorithm, because of communication and other overheads, takes $2n^2/(SP)$ time to add the matrices using P processors of speed S . The serial time is $3n^2 + n^2/S$ and the parallel time is $3n^2 + 2n^2/(SP)$. The analytical real speedup is:

$$(3 + 1/S)/[3 + 2/(SP)] \dots \dots \dots (1)$$

Though analytical real speed up can be computed without necessarily programming an algorithm and measuring its execution time, it is not ideal for the practitioner whose concern has to do with constant factors associated with the speed up. The distinction between a speed up of $P/8$ and $P/4$ is very important in practice (in the case of matrix addition example). Although we performed a more precise analysis than when using asymptotic complexities in our example of matrix addition, actually observed speed up differs from that of the predicted speed up. Since the analysis can't account for all overheads encountered in an actual execution. The best serial algorithm and the parallel algorithm needs to be programmed and run if we want to obtain the measured real speed up of a parallel system. The problem instance and that of the number of processors varies with the measured speed up.

2.3 Relative speedup

Because the measurement of real speedup requires us to program not just the parallel algorithm but also the best serial algorithm, it is tempting to look at an easier way to report the parallel algorithm's speedup. In particular, we might avoid using the best serial algorithm and use the single-processor execution time of the parallel algorithm itself (provided the parallel algorithm has been written with P as a parameter that can be set to 1), or use the serial algorithm on which the parallel algorithm is based. We do this when measuring relative speedup, which, like real speedup, can be determined analytically or experimentally. Relative speedup suffers from the same deficiencies as real speedup: memory limitations often make it impossible to obtain the execution time on a single processor, and relative speedup is higher with slower processors. It also favors code that is inefficient when run on a single processor. This does not happen with real speedup, because the single-processor code used is the best available. Consider the problem of multiplying two $n \times n$ matrices on a parallel computer system consisting of a host to which P parallel processors are attached as peripherals. Suppose that the time to transfer the two matrices to the peripheral parallel processors and receive the results is $3n^2$ and the time to multiply the matrices on the parallel processors is $n^3/(SP)$ (assume $P \leq n^2$), where S is the speed of one of the parallel processors. The relative speedup is $(n^3/S)/[3n^2 + n^3/(SP)]$, assuming that if we used a serial computer consisting of one of the parallel processors, we will not need to transfer the metrics to and from. Simplifying this formula yields the formula $n(3S + n/P)$, which implies that for fixed n and P , speedup increases as S decreases. Speedup is greater with slower processors. Suppose we change the program to a less efficient one-one that does more work. For our matrix multiplication example, we could add redundant work such as re-computing the matrix product several times. The execution time of the new code is kn^3/P , where k is the degree of inefficiency. The relative speedup is now

$(kn^3/S)/[3n^2 + kn^3/(SP)] = n/(3S/k + n/P)$, which is an increasing function of k . When $S = 1$, $n = 100$, and $P = 10$, the relative speedup is $100k/(3 + 10k)$. For $k = 1$, this is 7.69 and for $k = 10$, it is 9.71. As in the case of real speedup, using relative speedup as a parallel performance metric independent of actual runtime is hazardous. The potential for erroneous conclusions is even greater because the speedup is not measured with respect to the best serial algorithm. It is possible that a parallel algorithm that exhibits a relative speedup of 10 when solving a problem of size 1,000 using 20 processors actually takes more time than the best serial algorithm running on a single processor of the same parallel computer

2.4 Scaled speedup

Scaled speedup is defined as the speedup curve obtained when the problem size is increased linearly with the number of processors. (Sun and Gustafson, 1991) propose a metric called **size up** to capture the time-constrained scaling strategy.

2.5 Sizeup

Sizeup is defined as the ratio of the size of the problem solved on the parallel machine to the size of the problem solved on the sequential machine for the same execution time. When the execution time does not decrease linearly with the number of processors, speedup does not provide any additional information needed to find out if the deviation is due to bottlenecks in the application or in the machine. Similarly, when a parallel system exhibits non-ideal behaviour in the memory-constrained or time-constrained scaling strategies, scaled-speedup and sizeup fail to show whether the problem rests with the application and/or the architecture. Three other metrics (Kumar and Rao, 1987), (Karp and Flatt, 1990), (Nussbaum and Agarwal, 1991) attempt to address this deficiency.

2.6 Limits to speedup

Suppose two 100x100 matrices has to be added manually. Initially, the metrics will be written on a long wall where the result will also be written. If it has to be done by one person, suppose it would take him or her a day. Using a friend, he/she could do it in a little over one-half of a day he/she would need a bit of time to tell the friend which half to work on, after which it would take them each one-half of a day to do the actual addition. The speedup is almost 2. If 10,000 people were standing as in a 100x100 matrix, one could assign each of them to add a unique pair of matrix elements. They would accomplish the addition task in a little over one-ten-thousandth of a day, for a speedup of almost 10,000. With a million people, however, the job could not be done any faster. There wouldn't be enough work to go around. In fact, the ensuing pandemonium might actually make the job take longer, resulting in a smaller speedup. Assumption can be made that each problem instance is solvable by a finite amount of work, it follows that by increasing the number of processors indefinitely, a point will be reached when there isn't any work to be distributed to the newly added processors. No further speedup will be possible. However, depending on the amount of work available, the attainable speedup might be very large and might be able to gainfully employ (Gustafson, et al, 1988) introduced metric called scaled-speed up that tries to capture the memory-constrained scaling strategy.

2.7 Isoefficiency function

Isoefficiency function (Kumar and Rao, 1987) tries to capture the impact of problem sizes along the application dimension and the number of processors along the architectural dimension. For a problem with a fixed size, the processor utilization (efficiency) normally decreases with an increase in the number of processors. Similarly, if the problem size has to be scaled up keeping the number of processors fixed, the efficiency usually increases.

Isoefficiency function relates these two artifacts in typical parallel systems and is defined as: the rate at which the problem size needs to grow with respect to the number of processors in order to keep the efficiency constant. An isoefficiency whose growth is faster than linear suggests that overheads in the hardware are a limiting factor in the scalability of the system, while a growth that is linear or less is indicative of a more scalable hardware. Apart from providing a bound on achievable performance (Amdahl's law), the theoretical serial fraction of an application is not very useful in giving a realistic estimate of performance on actual hardware. (Karp and Flatt, 1990) use an *experimentally determined serial fraction* for a problem with a fixed size in evaluating parallel systems is computed by executing the application on the actual hardware and calculating the effective loss in speedup. On an ideal architecture with no overheads introduced by the hardware, would be equal to the theoretical serial fraction of the application. Hardware deficiencies lead to a higher and a constant with increasing number of processors implies a scalable parallel system very often suggesting that there is no overhead introduced by the underlying hardware. Decreasing is an indication of super-linear speedup that arises due to reasons such as memory size and randomness in application execution, as outlined in (Helmbold and MacDowell, 1990). (Nussbaum and Agarwal, 1991) quantify scalability as a ratio of the application's asymptotic speedup when run on the actual architecture to its corresponding asymptotic speedup when run on an EREW PRAM (Fortune and Wyllie, 1978), for a given problem size. Application scalability is a measure of the inherent parallelism in the application and is expressed by its speedup on an architecture with an idealized communication structure such as a PRAM. Architectural scalability is defined as the relative performance between the ideal and real architectures. A larger ratio is an indication of better performance obtained in running the given application on the hardware platform. Metrics used by (Kumar and Rao, 1987), (Karp and Flatt, 1990),

(Nussbaum and Agarwal, 1991) thus attempt to identify the cause (the application or the architecture) of the problem when the parallel system does not scale as expected. Once the problem is identified, it is essential to find the individual application and architectural artifacts that lead to these bottlenecks and to quantify their relative contribution towards limiting the overall scalability of the system. For instance, we would like to find the contribution of the serial portion in the application, the work imbalance between the threads of control in the program, the overheads introduced by parallelization of the problem, towards limiting the scalability of the application. Similarly, it would be desirable to quantify the contribution from hardware components such as network latency, contention, and synchronization, and system software components such as scheduling and other runtime overheads. But it is difficult to glean this knowledge from the three metrics. Identifying, isolating and quantifying the different overheads in a parallel system that limit its scalability is crucial for improving the performance of the system by application restructuring and architectural enhancements. It can also help in choosing between alternate application implementations, selecting the best hardware platform, and the different other uses of a performance analysis study. Recognizing this importance, studies (Burkhart and Millen, 1989), (Croville and LeBlanc, 1994), (Sivasubramaniam, et al, 1994) have attempted to separate and quantify parallel system overheads.

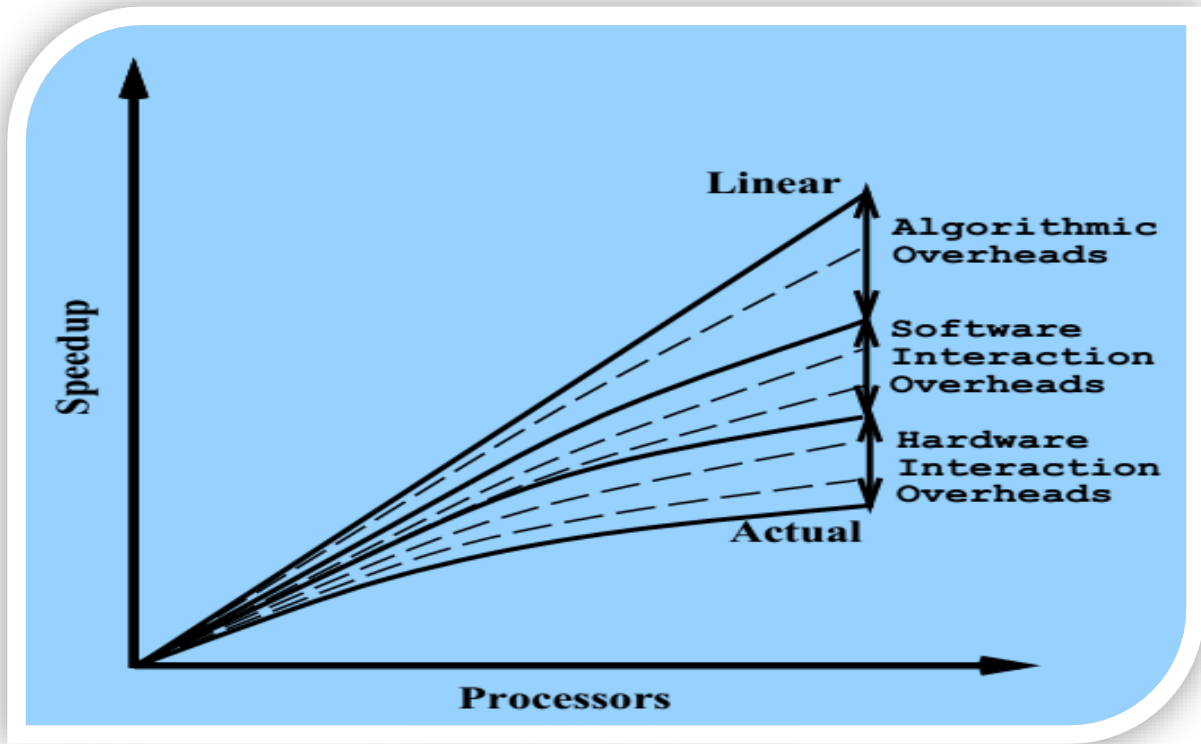


Fig 2.1: Overheads in a Parallel System (Sivasubramaniam, et al, 1994)

Ignoring effects of super linearity, one would expect a speedup that increases linearly with the number of processors, as shown by the curve named “linear” in Figure 2.1. But overheads in the system would limit its scalability resulting in a speedup that grows much slower as shown by the curve labelled “actual” in Figure 2.1. The disparity between the “linear” and “actual” curves is due to growth of overheads in the parallel system. Parallel system overheads may be broadly classified into a purely algorithmic component (*algorithmic overhead*), a component arising from the interaction of the application with the system software (*software interaction overhead*), and a component arising from the interaction of the application with the hardware (*hardware interaction overhead*). Algorithmic overheads arise from the inherent serial part in the application, the work-imbalance between the executing threads of control, any redundant computation that may be performed, and additional work introduced by the parallelization.

Software interaction overheads such as overheads for scheduling, message-passing, and software synchronization arise due to the interaction of the application with the system software. Hardware slow down due to network latency (the transmission time for a message in the network), network contention (the amount of time spent in the network waiting for links to become free), synchronization and cache coherence actions, contribute to the hardware interaction overhead. To fully understand the scalability of the parallel system, it is important to isolate and quantify the impact of different parallel system overheads on the overall execution as shown in Figure 1. *Overhead functions* and *lost cycles* (Crovelle and LeBlanc, 1994) are metrics that have been proposed to capture the growth of overheads in a parallel system. Both these metrics quantify the contribution of each overhead towards the overall execution time. The studies differ in the techniques used to quantify these metrics. Experimentation is used in (Crovelle and LeBlanc, 1994) to quantify lost cycles, while simulation is used in (Sivasubramaniam, et al, 1994) to quantify overhead functions. In addition to quantifying the overheads in a given parallel system, a performance evaluation technique should also be able to quantify the growth of overheads as a function of system parameters such as problem size, number of processors, processor clock speed, and network speed. This information can prove useful in predicting the scalability of systems governed by a different set of parameters. A range of performance metrics, from simple metrics like speedup which provide scalar information about the performance of the system, to more complicated vector metrics like overhead functions that provide a wide range of statistics about the parallel system execution is explained in Table 2.2. The metrics that reveal only scalar information are much easier to calculate. In fact, the overall execution times of the parallel system parameterized by number of processors and problem sizes would suffice to calculate metrics like speedup, scaled speedup, sizeup, isoefficiency function and experimentally determined

serial fraction. On the other hand, the measurement of overhead functions and lost cycles would need more sophisticated techniques that use a considerable amount of instrumentation.

Table 2.2: Performance Metrics

METRICS	MERITS	DRAWBACKS
Speedup, Scaled Speedup, Sizeup	Useful for quantifying performance improvements as a function of the number of processors and problem sizes.	Do not identify or quantify bottlenecks in the system, providing no additional information when the system doesn't scale as expected.
Isoefficiency function, Experimentally determined serial fraction, Nussbaum and Agarwal's metric	Attempt to identify if the application or architecture is at fault in limiting the scalability of the system.	The information provided may not be adequate to identify and quantify the individual application and architectural features that limit the scalability of the system.
Overhead functions, Lost cycles	Identify and quantify all the application and architectural overheads in a parallel system that limit its scalability, providing a detailed understanding of parallel system behaviour.	Quantification of these metrics needs more sophisticated instrumentation techniques.

2.8 Efficiency (E)

Measures the work rate per processor (that is, $E(k) = S(k)/k$), and has an "ideal" value of unity.

It is well known that given a parallel architecture and a problem instance of a fixed size, the

speedup of a parallel algorithm does not continue to increase with increasing number of processors but tends to saturate or peak at a certain value. For a fixed problem size, the speedup saturates either because the overheads grow with increasing number of processors or because the number of processors eventually exceeds the degree of concurrency inherent in the algorithm. For a variety of parallel systems, given any number of processors p , speedup arbitrarily close to p can be obtained by simply executing the parallel algorithm on big enough problem instances. The ease with which a parallel algorithm can achieve speedups proportional to p on a parallel architecture can serve as a measure of the scalability of the parallel system. The isoefficiency function (Fortune and Wyllie, 1978), (Croville and LeBlanc, 1994) is one such metric of scalability which is a measure of an algorithm's capability to effectively utilize an increasing number of processors on a parallel architecture. The isoefficiency function of a combination of a parallel algorithm and a parallel architecture relates the problem size to the number of processors necessary to maintain a fixed efficiency or to deliver speedups increasing proportionally with increasing number of processors. The efficiency of a parallel system is given by:

$$E = \frac{W}{W + T_0(W, p)}$$

If a parallel system is used to solve a problem instance of a fixed size W , then the efficiency decreases as p increases. The reason is that the total overhead $T_0(W, p)$ increases with p . For many parallel systems, for a fixed p , if the problem size W is increased, then the efficiency increases because for a given p , $T_0(W, p)$ grows slower than $O(W)$. For these parallel systems, the efficiency can be maintained at a desired value (between 0 and 1) for increasing p , provided W is also increased. We call such systems scalable parallel systems. Note that for a given

parallel algorithm, for different parallel architectures, W may have to increase at different rates with respect to p in order to maintain a fixed efficiency. For example, in some cases, W might need to grow exponentially with respect to p to keep the efficiency from dropping as p is increased. Such a parallel system is poorly scalable because it would be difficult to obtain good speedups for a large number of processors, unless the size of the problem being solved is enormously large. On the other hand, if W needs to grow only linearly with respect to p , then the parallel system is highly scalable and can easily deliver speedups increasing linearly with respect to the number of processors for reasonably increasing problem sizes. The isoefficiency functions of several common parallel systems are polynomial functions of p ; i.e., they are $O(p^x)$, where $x \geq 1$. A small power of p in the isoefficiency function indicates a high scalability. If a parallel system incurs a total overhead of $T_o(W, p)$, where p is the number of processors in the parallel ensemble and W is the problem size, then the efficiency of the system is given by:

$$E = \frac{1}{1 + \frac{T_o(W,p)}{W}}$$

In order to maintain a constant efficiency, W should be proportional to $T_o(W, p)$ or the following relation must be satisfied:

$$W = K T_o(W, p) \dots \dots \dots \text{Eqn. (a)}$$

$K = \frac{E}{1-E}$ Is a constant depending on the efficiency to be maintained. Equation (a) is the central relation that is used to determine the isoefficiency function. This is accomplished by abstracting W , as a function of p through algebraic manipulations on Equation (a). If the problem size needs to grow as fast as $f E(p)$ to maintain an efficiency E , then $f E(p)$ is

defined to be the isoefficiency function of the parallel algorithm-architecture combination for efficiency E . Isoefficiency analysis has been found to be very useful in characterizing the scalability of a variety of parallel systems. An important feature of isoefficiency analysis is that in a single expression, it succinctly captures the effects of characteristics of the parallel algorithm as well as the parallel architecture on which it is implemented. By performing isoefficiency analysis, one can test the performance of a parallel program on a few processors, and then predict its performance on a larger number of processors. But the utility of the isoefficiency analysis is not limited to predicting the impact on performance of an increasing number of processors. It can also be used to study the behaviour of a parallel system with respect to changes in other hardware related parameters such as the speed of the processors and the data communication channels. As Table 2.2 outlines the drawbacks of the numerous metrics in existence, there is a need for an enhanced metric and this thesis prefer solution to that.

2.9 Summary

The various metrics reviewed in this thesis sums up to Speedup metric, Scaled Speedup metric, Sizeup metric, Isoefficiency function and Fixed-Size metric. The Speedup, Scaled Speedup and Size-up are useful for quantifying performance improvements as a function of the number of processors and problem sizes but the disadvantages or the drawbacks is that, they don't identify or quantify bottlenecks in the system and hence provide no additional information when the system does not scale as expected. The Isoefficiency function attempts to identify if the application or the architecture is at fault in limiting the scalability of the system but the drawback is that, the information provided may not be adequate to identify and quantify the individual application and architectural features that limit the scalability of the system. The overhead functions identify and quantify all the applications and architectural overheads in a parallel

system that limit its scalability, providing a detailed understanding of parallel system behavior, but the limitation of this metric is that, quantification of this metric needs more sophisticated instrumentation techniques. The Quality of Service function considered in most metrics is simply $f=1$, because they only evaluate the time to completion which is already taken into account in the throughput. The strategy of scaling up a distributed system is more complex than simply adding processors, storage and bandwidth. From the numerous drawbacks associated with the already existing metrics, there is a need for an enhanced metric and this thesis prefer solutions to them as it is clearly outlined in chapter three.



CHAPTER THREE

METHODOLOGY

3.1 Introduction

The scalability metric adopted by this thesis is founded on some fundamental quantities of which they will be defined, a number of them that are of interest and then introduce the notation that will be used in referring to these quantities. The second is to derive various algebraic relationships among these quantities, some of which, because of their importance, will be identified as fundamental laws. The third is to explore thoroughly the most important of these fundamental laws, Little's law proposed by J.D.C. Little, which states that the average number of requests in a system must be equal to the product of the throughput of that system and the average time spent in that system by a request. Critical use cases, scenarios that are important to scalability would be identified. Precise, quantitative, measurable scalability requirements would be identified. The scalability will be applied to idealized cases.

3.1.1 The Basic Quantities

The following are the basic quantities,

T = the length of time we observed the system

A = the number of request arrivals we observed

C = the number of request completions we observed

From the above measurements we can define the following additional quantities:



Fig.3.1 An Abstract System

λ is the arrival rate: $\lambda = \frac{A}{T}$

X , is the throughput: $X = \frac{C}{T}$

If the system consists of a single resource, we can also measure:

B , the length of time that the resource was observed to be busy. Two more defined quantities now are meaningful:

U , the Utilization: $U = \frac{B}{T}$

S , the average service requirement per request: $S = \frac{B}{C}$

We can now derive the first of our fundamental law, Algebraically, $\frac{B}{T} = \frac{C}{T} \frac{B}{C}$

From the three preceding definitions, $\frac{B}{T} \equiv U$, $\frac{C}{T} \equiv X$, $\frac{B}{C} \equiv S$

Therefore:

The Utilization Law: $U = XS$ Eqn. 1

That is, the utilization of a resource is equal to the product of the throughput of that resource and the average service requirement at that resource.

3.1.2 The Little's Law

Given a computer system, Little's law can be applied at many different levels: to a single resource, to a subsystem, or to the system as a whole. The key to success is consistency: the definitions of population, throughput, and residence time must be compatible with one another.

Let T = the observation interval of length.

We temporarily denote, W = accumulated time in system.

Define:

N , the average number of requests in the system: $N \equiv \frac{W}{T}$

R , the average system residence time per request: $R \equiv \frac{W}{C}$

Algebraically, $\frac{W}{T} = \frac{C}{T} \frac{W}{C}$, But $\frac{W}{T} \equiv N$, $\frac{C}{T} \equiv X$, $\frac{W}{C} \equiv R$

Therefore:

Little's Law: $N = XR$ Eqn. 2

That is, the average number of requests in a system is equal to the product of the throughput of that system and the average time spent in that system by a request.

If we denote think time by Z , then we can write this incarnation of Little's law as $N = X (R + Z)$. As with the utilization law, this application is so ubiquitous that we give it its own name and notation, expressing R in terms of the other quantities:

The Response Time Law: $R = \frac{N}{X} - Z$ Eqn.3

When considering a disk, it is natural to define a request to be a disk access, and to measure throughput and residence time on this basis. When considering an entire system, on the other hand, it is natural to define a request to be a user-level interaction, and to measure throughput and residence time on this basis. The relationship between these two views of a system is expressed by the Forced Flow law, which states that the flows (throughputs) in all parts of a system must be proportional to one another. Let a variable with the subscript k refer to the k -th resource (a variable with no subscript continues to refer to the system as a whole), then write this definition as:

V_k , the visit count of resource k : $V_k \equiv \frac{C_k}{C}$

If the definition is rewritten as $C_k = V_k C$ and recall that the completion count divided by the length of the observation interval is defined to be the throughput, then the throughput of resource k is given by:

The Forced Flow law: $X_k = V_k X$ Eqn.4

Define:

D_k , the service demand at resource k : $D_k \equiv V_k S_k$

Where:

D_k Refers to: The total service demand

S_k Refers to: the service requirement per visit at resource k

D Refers to: the sum of the (D_k) : the total service demanded by a job at all resources.

In considering using the utilization law to calculate the utilization of a resource, we can express throughput in terms of visits to that resource (V_k), in which case service requirement must be expressed as service requirement per visit (X_k). Using the forced flow law, we can also express throughput in terms of system-level interactions (X), in which case service requirement must be expressed on a per-interaction basis (D_k). In other words, $U_k = X_k S_k = X D_k$.

If a system is observed for an interval of length T , the utilizations of the various resources, U_k , can easily be obtained and the system-level completion count, C . The service demands at the various resources then can be calculated as $D_k = \frac{B_k}{C} = \frac{U_k T}{C}$

Frequently it will be convenient to assume that systems satisfy the flow balance property, namely, that the number of arrivals equals the number of completions, and thus the arrival rate equals the throughput:

The Flow Balance Assumption: $A = C$, therefore $\lambda = X$

Table 3.1 Summery of the Notations

NOTATION	MEANING
T	Length of the observation interval
A_k	Number of arrivals observed
C_k	Number of completions observed
λ_k	Arrival rate
X_k	throughput
B_k	Busy Time
U_k	Utilization
S_k	Service requirement per visit
N	Customer preparation
R_k	Residence Time
Z	Think Time of a terminal User
V_k	Number of Visits
D_k	Service Demand

Table 3.2 The Fundamental Laws

The Utilization Law: $U_k = X_k S_k = X D_k$
Little's Law: $N = X R$
The Response Time Law: $R = \frac{N}{X} - Z$
The Forced Flow law: $X_k = V_k X$

3.2 Using The Fundamental Laws To Derive Bounding Equations

Asymptotic bounds hold for a wider class of systems than do balanced system bounds. They are also simpler to compute. The advantage of off-setting balanced system bounds is that they are tighter, and thus provide more precise information than asymptotic bounds. System sizing studies typically involve consideration of a large number of candidate configurations. Often a

single resource such as the CPU is the dominant concern, because the remainder of the system can be configured to match the power of this resource. To derive the various equations to calculate the bounds, Let:

K = The number of service centers

D = The sum of the service demands at the centers.

D_{max} = The largest service demand at any single center.

- The type of the customer class (batch, terminal, or transaction).

Z = The average think time, that is if the class is of terminal type.

The type of customer class needs to be identified that is either Batch or Terminal or Transaction.

The throughput bounds shows the maximum customer arrival rate that the system can process in the case of transaction type workloads models, whiles the response time bounds indicate the smallest and largest possible response time that the customer is likely to experience as a function of the system arrival rate. The bounds shows the minimum and maximum possible system throughputs and response time as functions of the number of customers in the system in the case of batch or terminal type workloads models. Optimistic bounds are used to refer to response time lower and throughput upper bounds. This is so as it depicts the best possible performance, whiles pessimistic bounds are used to refer to response time upper and throughput lower bounds. This is because; they show the worst possible performance, whiles we treat only bounds on system throughput and response time.

3.2.1 The Asymptotic Bounds

Per their name, they are derived by taking into account asymptotically extreme conditions of light and heavy loads. Analysis of asymptotic bound provides optimistic and pessimistic bounds on system response time and throughput in single class queuing network. To ascertain whether the bounds are valid or not depends on only a single proposition which is if the service demands of a customer at a center doesn't depends on the number of other customers in the system at that moment, or at which service centers they are placed. Whatever information that the asymptotic bounds gives relies on either the system workload is transaction type (open) or batch or terminal type (closed). We commence with the simple scenario, which is the open (transaction type workloads).

3.2.2. The Transaction Workloads

For transaction workloads, the throughput bound indicates the maximum possible arrival rate of customers that the system can process successfully. If the arrival rate exceeds this bound, a backlog of unprocessed customers grows continually as jobs arrive. Thus, in the long run, an arriving job has to wait an indefinitely long time (since there may be any number of jobs already in queue when it arrives). In this case we say that the system is saturated. The throughput bound thus is the arrival rate that separates feasible processing from saturation. The key to determining the throughput bound is the utilization law:

$U_k = X_k S_k$ for each center k . If we denote the arrival rate to the system as λ , then $X_k = \lambda V_k$ and the utilization law can be rewritten as $U_k = \lambda D_k$ where D_k is the service demand at center k . To derive the throughput bound, we simply note that as long as all centers have unused capacity (i.e., have utilizations less than one), an increased arrival rate can be accommodated. However, when any of the centers becomes saturated (i.e., has utilization one), the entire system becomes

saturated, since no increase in the arrival rate of customers can be handled successfully. Thus, the throughput bound is the smallest arrival rate λ_{sar} at which any center saturates. Clearly, the center that saturates at the lowest arrival rate is the bottleneck center i.e, the center with the largest service demand. Let max be the index of the bottleneck center. Then:

$$U_{max}(\lambda) = \lambda D_{max} \leq 1$$

So:

$$\lambda_{sar} = \frac{1}{D_{max}}$$

Thus, for arrival rates greater than or equal to $1/D_{max}$ the system is saturated, while the system is capable of processing arrival rates less than $1/D_{max}$. Asymptotic response time bounds indicate the largest and smallest possible response times experienced by customers when the system arrival rate is λ . Because the system is unstable if $\lambda > \lambda_{sar}$ our investigation will be limited to the case where the arrival rate is less than the throughput bound. There are two extreme situations that is:

1. In the best possible case, no customer ever interferes with any other, so that no queueing delays are experienced. In that case the system response time of each customer is simply the sum of its service demands, which we denote by D.
2. In the worst possible case, n customers arrive together every n / λ time units (the system arrival rate is $\frac{n}{n/\lambda} = \lambda$). Customers at the end of the batch are forced to queue for customers at the front of the batch, and thus experience large response times. As the batch size n increases, more and more customers are waiting an increasingly long time. Thus, for any postulated pessimistic bound on response times for system arrival rate λ , it is possible to pick a batch size

n sufficiently large that the bound is exceeded. Conclusion can be made that there is no pessimistic bound on response times, regardless of how small the arrival rate λ might be. These results are somewhat unsatisfying. Fortunately, the throughput and response time bounds provide more information in the case of closed (batch and terminal) workload types.

3.2.3. The Batch and Terminal Workloads

To derive the bounds equations, first consider the bounds on throughput, and then use Little's law to transform them into corresponding bounds on response time. The analysis is stated in terms of terminal workloads. By taking the think time, Z , to be zero, we obtain results for batch workloads. To begin with the heavy load (many customer) situation. As the number of customers in the system (N) becomes large, the utilizations of all centers grow, but clearly no utilization can exceed one. From the utilization law we have for each center k that:

$$U_k(N) = XN(D)_k \leq 1$$

Each center limits the maximum possible throughput that the system can achieve. Since the bottleneck center (max) is the first to saturate, it restricts system throughput most severely. We conclude that:

$$X(N) \leq \frac{1}{D_{max}}$$

Intuitively this is clear, because if each customer requires on average D_{max} time units of service at the bottleneck center, then in the long run customers certainly cannot be completed any faster than one every D_{max} time units. In the case of light load (few customers) situation. At the extreme, a single customer alone in the system attains a throughput of $1/(D + Z)$, since

each interaction consists of a period of service (of average length $D = \sum_{k=1}^k D_k$ and a think

time (of average length Z). As more customers are added to the system there are two bounding situations:

1. The smallest possible throughput occurs when each additional customer is forced to queue behind all other customers already in the system. In this case, with N customers in the system, $(N - 1) D$ time units are spent queued behind other customers, D time units are spent in service, and Z time units are spent thinking, so that the throughput of each customer is $1/(ND + Z)$. Thus, system throughput is $N/(ND + Z)$.

2. The largest possible throughput occurs when each additional customer is not delayed at all by any other customers in the system. In this case no time is spent queueing, D time units are spent in service, and Z time units are spent thinking. Thus, the throughput of each customer is $1/(D + Z)$ and system throughput is $N/(D + Z)$. The above observations can be summarized as the asymptotic bounds on system throughput:

$$\frac{N}{ND + Z} \leq X(N) \leq \min \left(\frac{1}{D_{max}}, \frac{N}{D + Z} \right)$$

The optimistic bound consists of two components, the first of which applies under heavy load and the second of which applies under light load. There is a particular population size N^* such that for all N less than N^* the light load optimistic bound applies, while for all N larger than N^* the heavy load bound applies. This crossover point occurs where the values of the two bounds are equal:

$$N^* = \frac{D + Z}{D_{max}}$$

We can obtain bounds on response time $R(N)$ by transforming our throughput bounds using Little's law. We begin by rewriting the previous equation:

$$\frac{N}{ND + Z} \leq \frac{N}{R(N) + Z} \leq \min\left(\frac{1}{D_{\max}}, \frac{N}{D + Z}\right)$$

Inverting each component to express the bounds on $R(N)$ yields:

$$\max\left(D_{\max}, \frac{D + Z}{N}\right) \leq \frac{R(N) + Z}{N} \leq \frac{ND + Z}{N}$$

Alternatively,

$$\max(D, ND_{\max} - Z) \leq R(N) \leq ND$$

Table.3.3 Summary of Asymptotic Bound Equations

	Workload Type	Bounds
X	Batch	$\frac{1}{D} \leq X(N) \leq \min\left(\frac{N}{D}, \frac{1}{D_{\max}}\right)$
	Terminal	$\frac{N}{ND + Z} \leq X(N) \leq \min\left(\frac{N}{D + X}, \frac{1}{D_{\max}}\right)$
	Transaction	$X(\lambda) \leq 1/D_{\max}$
R	Batch	$\max(D, ND_{\max}) \leq R(N) \leq ND$
	Terminal	$\max(D, ND_{\max} - Z) \leq R(N) \leq ND$
	Transaction	$D \leq R(\lambda)$

3.3.4 Balanced System Bounds

With a modest amount of computation beyond that required for asymptotic bounds, tighter bounds can be obtained. These bounds are called balanced system bounds because they are based upon systems that are “balanced” in the sense that the service demand at every center is the same, i.e., $D_1 = D_2 = D_3 = \dots = D_K$. We first establish some special properties of balanced systems. We then show how these properties can be exploited to determine bounds on performance that complement the asymptotic bounds and lead to more precise knowledge of system behavior. For balanced systems, the techniques to be presented is in simple form. The utilization of every service center is given by:

$$U_k(N) = \frac{N}{N + K - 1}$$

By the utilization law, system throughput is then:

$$U(N) = \frac{U_k}{D_k} = \frac{N}{N + K - 1} = \frac{1}{D_k}$$

Where D_k is the service demand at every center.

Let D_{max} , D_{ave} , and D_{min} , denote respectively the maximum, average, and minimum of the service demands at the centers of the model we wish to evaluate. Then bound the throughput of that system by the throughputs of two related balanced systems: one with service demand D_{min} at every center, and the other with service demand D_{max} at every center:

$$\frac{N}{N + K - 1} \times \frac{1}{D_{max}} \leq X(N) \leq \frac{N}{N + K - 1} \times \frac{1}{D_{min}}$$

These inequalities hold because, of all systems with K centers, N customers, and maximum service demand D_{max} the one with the lowest throughput is the balanced system with demand

D_{max} at each center. Similarly, of all systems with K centers, N customers, and minimum demand D_{min} , the one with the highest throughput is the balanced system with demand D_{min} at each center. Corresponding bounds on average response times are:

$$(N + K - 1)D_{min} \leq R(N) \leq (N + K - 1)D_{max}$$

Tighter balanced system bounds can be obtained by constraining not only the maximum service demand, D_{max} but also the total demand, D (or equivalently, the average demand, D_{ave} . Of all

systems with a given total service demand $D = \sum_{k=1}^k D_k$ the one with the highest throughput

and the lowest average response time is the one in which all service demands are equal (i.e, $D_k = D/K$, $k = 1, \dots K$). This confirms our intuition that the increase in delay resulting from an increase in load is greater than the decrease in delay resulting from an equivalent decrease in load. Therefore, optimistic bounds are given by:

$$X(N) \leq \frac{N}{N + K - 1} \times \frac{1}{D_{ave}} = \frac{N}{D + (N - 1)D_{ave}}$$

and:

$$D + (N - 1)D_{ave} \leq R(N)$$

Table.3.4 Summary of Balanced System Bounds Equations

	Workload Type	Bounds
X	Batch	$\frac{N}{D + (N - 1)D_{max}} \leq X(N) \leq \min(\frac{1}{D_{max}}, \frac{N}{D + (N - 1)D_{ave}})$
	Terminal	$\frac{N}{D + Z + \frac{(N-1)D_{max}}{1+Z/(ND)}} \leq X(N) \leq \frac{N}{D + Z + \frac{(N-1)D_{ave}}{1+Z/D}}$
	Transaction	$X(\lambda) \leq 1/D_{max}$
R	Batch	$\max(ND_{max}, D + (N - 1)D_{ave}) \leq R(N) \leq D + (N - 1)D_{max}$
	Terminal	$\max\left(ND_{max} - Z, D + \frac{(N - 1)D_{ave}}{1 + Z/D}\right) \leq R(N) \leq D + \frac{(N - 1)D_{ave}}{1 + Z/D}$
	Transaction	$\frac{D}{1 - \lambda D_{ave}} \leq R(\lambda) \leq \frac{D}{1 - \lambda D_{ave}}$

3.3.5 The Algorithm for Closed Model Asymptotic Bound

To calculate the bounds, the following algorithm needs to be adhered to. (Steps are presented assuming a terminal workload, to treat a batch workload; set Z to zero.)

1. Calculate $D = \sum_{k=1}^k D_k$ and $D_{max} = \max_k D_k$
2. Calculate the intersection point of the components of the optimistic bounds:

$$N^* = \frac{D + Z}{D_{max}}$$

3. Bounds on throughput pass through the points: (optimistic bound):

$$(0,0) \text{ and } \left(1, \frac{1}{D + Z}\right) \text{ for } N \leq N^*$$

$$\left(0, \frac{1}{D_{max}}\right) \text{ and } \left(1, \frac{1}{D_{max}}\right) \text{ for } N \geq N^*$$

For pessimistic bound: it's not linear in N and for that matter, it should be calculated for each population of interest using the derived equation in Table 3.3

4. Bounds on average response time pass through the point:

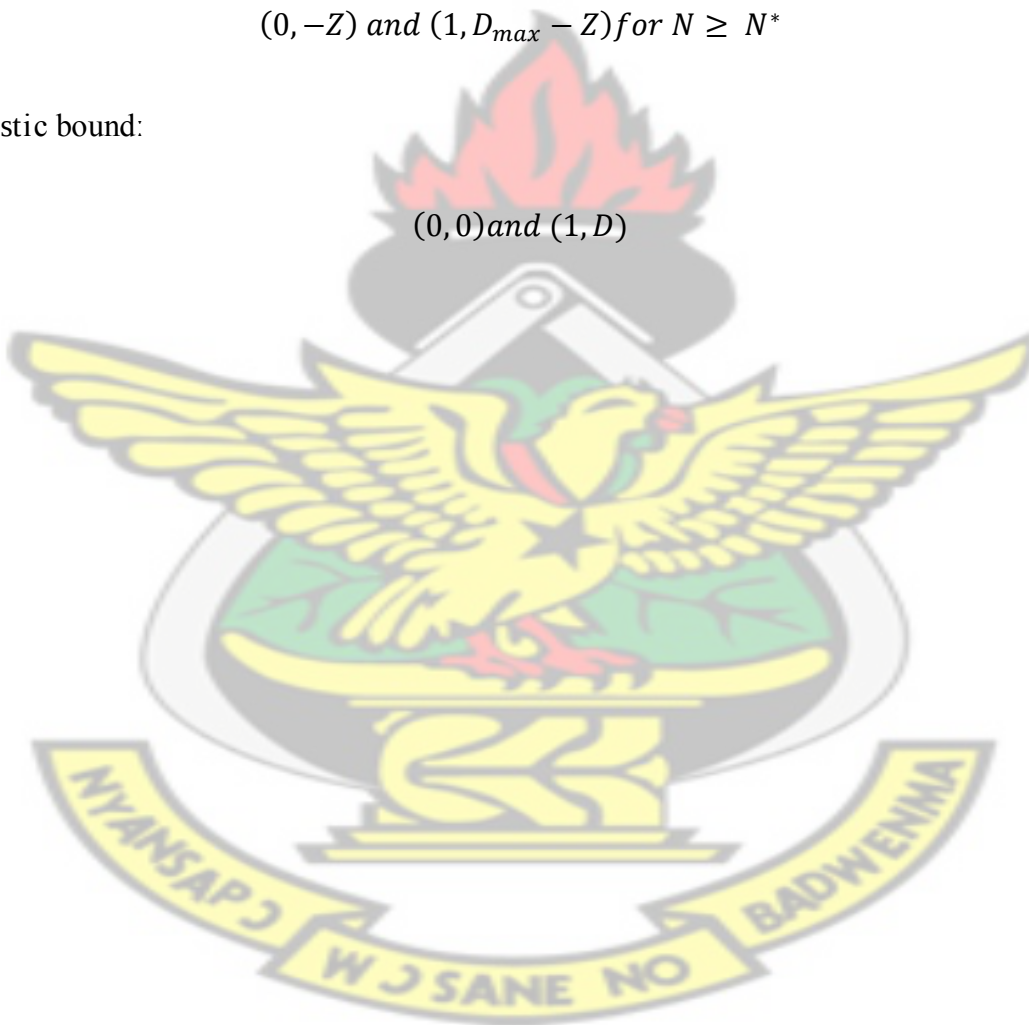
Optimistic bound:

$$(0, D) \text{ and } (1, D) \text{ for } N \leq N^*$$

$$(0, -Z) \text{ and } (1, D_{\max} - Z) \text{ for } N \geq N^*$$

Pessimistic bound:

$$(0, 0) \text{ and } (1, D)$$



Terminal Throughput

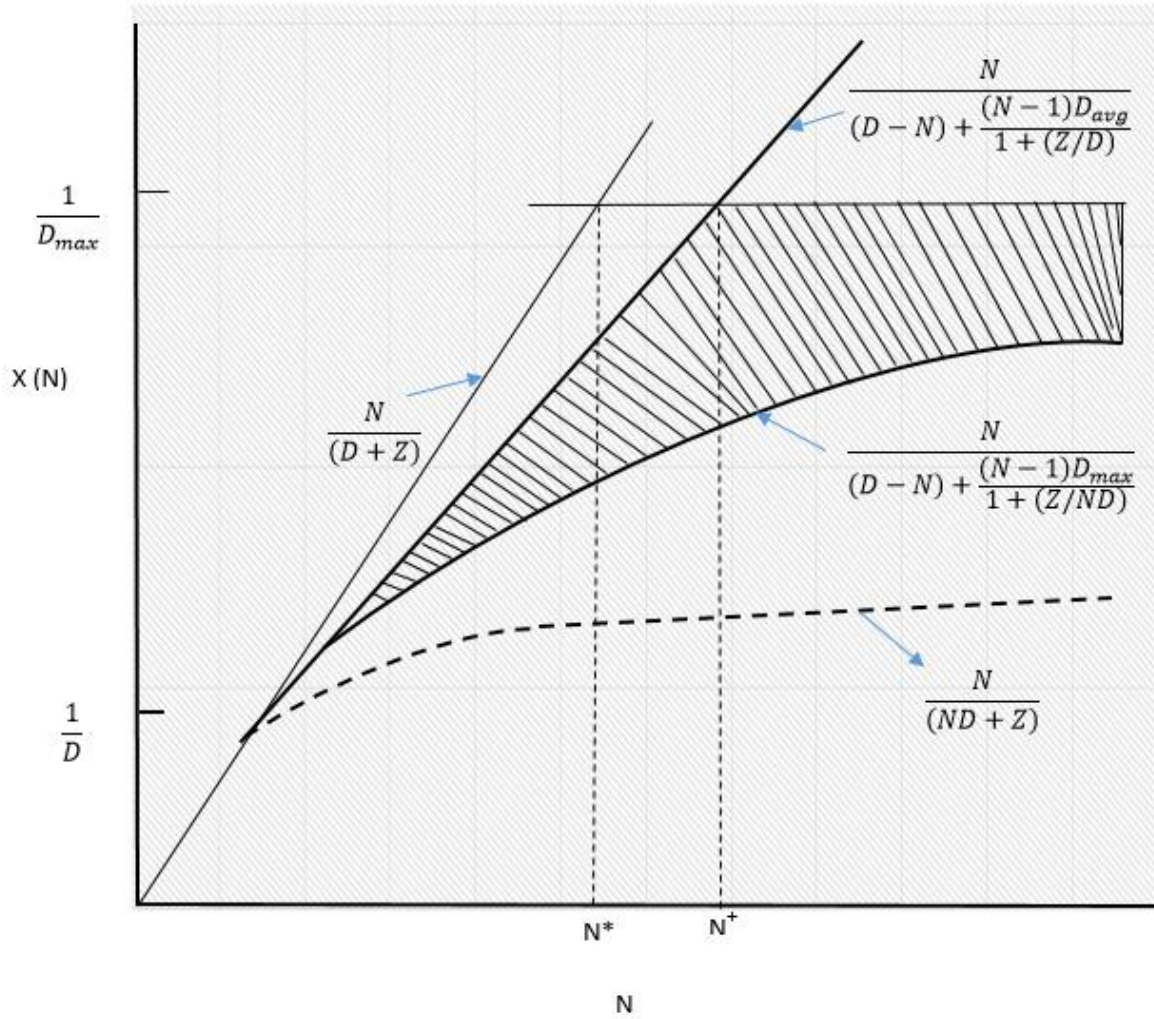
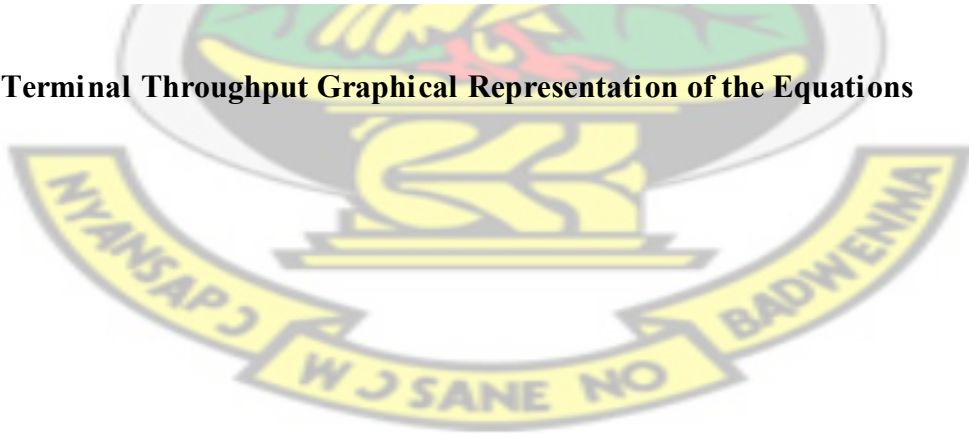


Fig. 3.2a Terminal Throughput Graphical Representation of the Equations



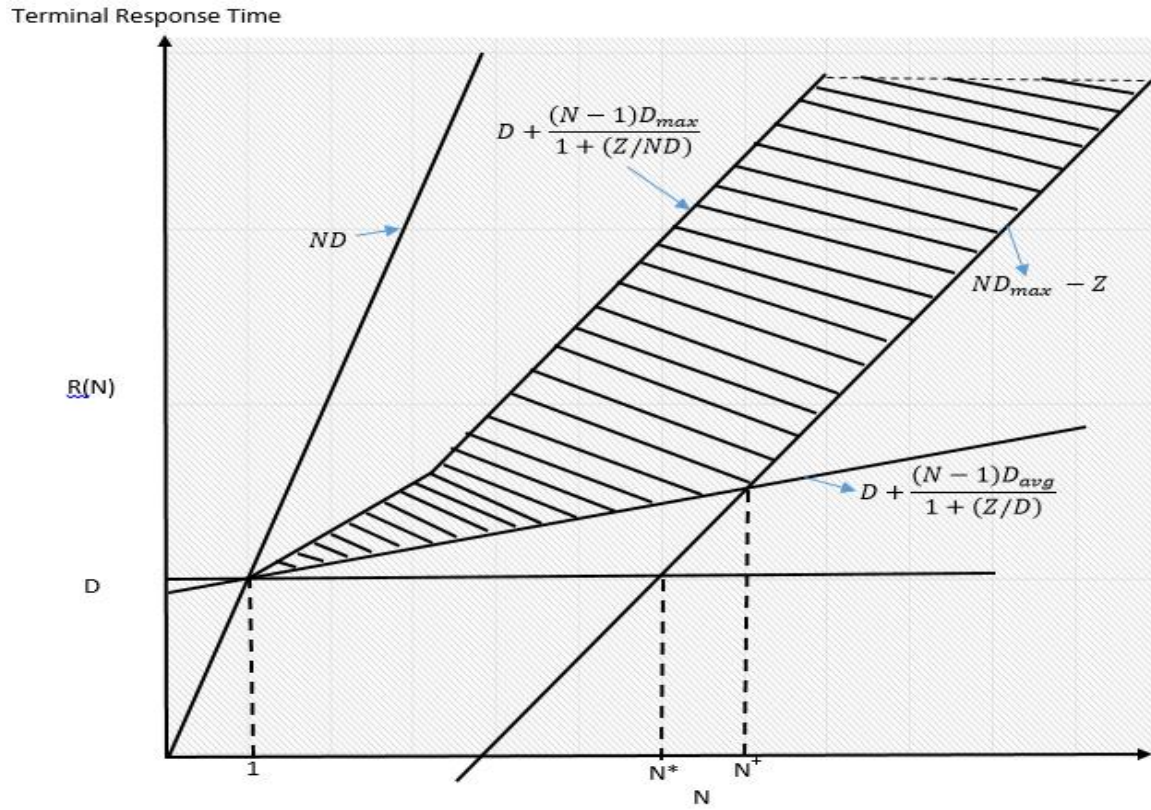


Fig. 3.2b Terminal Response Time Graphical Representation of the Equations

3.4 The Scalability Metric

Scalability $\psi(k_1, k_2)$ from one scale k_1 to another scale k_2 is the ratio of the efficiency figures for the two cases, $\psi(k_1, k_2) = E(k_2) / E(k_1)$. It also has an ideal value of unity. A typical metric is the fixed size speedup, in which the scaled-up base case has the same total computational work, and the speedup S is the ratio of the completion times (*i.e.*, $S(k) = T(1) / T(k)$).

The scalability framework is based on a scaling strategy for scaling up or down a given system, controlled by a scale factor k . Given that each scaled configuration is determined by a set of variables say, $x(k)$, $y(k)$ using numeric values, or enumerated alternative choices, categorized into two groups as follows:

$(x)_k$ represents a set of **scaling variables**, determined by the strategy for each value of k ,

$(y)_k$ represents a set of **adjustable variables**, termed scaling enablers, which are tuned to maximize the productivity for any given k . Since k determines x by the strategy, and x also influences y through the optimal tuning, the values of y are effectively determined by k .

The creation of threads within processes, the memory available for buffers, the allocation of processes to the processors, tuning of the middleware parameters, priorities, replication of processes and data, network bandwidth and the choice of communication protocols are all examples of scalability enablers. For the purpose of this thesis, utilization means the proportion of time the server is busy, residence time is the average time spent at the service center by a customer, both queueing and receiving service, queue length is the average number of customers at the service center, both waiting and receiving service, and throughput is the rate at which customers pass through the service center. Service centers represent system resources, and customers, which represent users or transactions. The scalability metric is based on productivity and for that matter when productivity is maintained as the scale changes, the system is said to be scalable. Given these three quantities:

$\lambda(k)$ = throughput in responses per sec, at scale k

$f(k)$ = average value of each response, calculated from its quality of service at scale k ,

$C(k)$ = cost at scale k , expressed as a running cost per second to be uniform with λ ,

Therefore, the productivity $F(k)$ is the value delivered per second, divided by the cost per second:

$$F(k) = \frac{\lambda(k) * f(k)}{C(k)}$$

The scalability metric relating systems at two different scale factors is therefore defined as the ratio of their productivity figures:

$$\psi(k_1, k_2) = \frac{F(k_2)}{F(k_1)} \quad \text{Eqn. 5}$$

The above equation is the scalability metric that the thesis will be based on. Frequently, k_1 is fixed at a known value and the metric is written as $\psi(k_2)$ or $\psi(k)$. The system is deemed as "scalable" if productivity keeps pace with costs from configuration A to configuration B, in which case the metric ψ will have a value greater than or not much less than unity. In this thesis, a threshold value of 0.8 will be used to say, the system is scalable if $0.8 < \psi$; as the threshold value should reflect what is an acceptable cost-benefit ratio to the system operator. The value of k at the threshold is the scalability limit of the system. If ψ rises above 1.0, then we will say the system has "positive scalability" example is super-linear speedup. With all the three quantities that enter the metric, throughput is self-evident. In the case of the cost, it is not a one-time capital cost, but is expressed as a rental cost, to express costs and benefits consistently per unit time. Examples of cost include the cost of software, processor, networks, storage, help desks, management, etc. This thesis will dwell on few of these examples of cost, for illustration. The value function $f(k)$ is determined by evaluating the performance of the scaled system, and may be a function of any appropriate system measure, including delay measures (mean, variance or jitter, probability of delay exceeding a threshold), availability, or the probability of data loss or timeouts. The thesis, will consider only the mean response time $T(k)$ at scale factor k , compared to a target value $\bar{T}$, in the following value function:

$$f(k) = \frac{1}{1 + T(k)/(\bar{T})} \quad \text{Eqn. 6}$$

From the above value function of Equations (5) and (6), the scalability metric for scale k_2 relative to k_1 is, after a little simplification:

$$\psi(k_1, k_2) = \frac{\lambda_2 * C_1 * (T_1 + \dot{T})}{\lambda_1 * C_2 * (T_2 + \dot{T})} \quad \text{Eqn. 7}$$

3.4.1 Generating The Measures And Applying It To Idealized Models

Ideally, the choice of the function f will depend on the system goals and on what is practical to estimate. Unlike other scalability measures, the metric ψ is based on quantities that can be predicted by an analytic calculation. Though the calculation is much cumbersome compared to other metrics, it is carried out below using a well-established queueing or extended queueing analysis that have been explained in this chapter earlier on. If the scaled-up system is actually constructed and instrumented, the metric can also be calculated from measurements of its operation. To prove the efficacy of the metric ψ , it will be applied to standard idealized systems that are widely understood at an intuitive level, which give analytic solutions for ψ . In all the models, the value of k for the base case is taken as 1, and the metric is written as $\psi(k)$.

3.4.2 Model A: General Speed-Scaled Open System with Proportional Costs

This model applies to any system architecture and behaviour (not necessarily solvable analytically) which has external arrivals and a steady state. Configuration A is an arbitrarily chosen reference, and configuration B is uniformly sped-up or time-scaled by a factor k , therefore the input rate and every component like computers, networks, and storage is faster. Cost is scaled by a factor k so that $C_2 = kC_1$, and $\lambda_2 = k\lambda_1$, $T_2 = T_1/k$. Further simplification of (7) gives:

$$\psi(k) = 1 + \frac{(k-1)T_1}{(T_1 + k\dot{T})}, \lim_{k \rightarrow \infty} \psi(k) = 1 + \frac{T_1}{\dot{T}} \quad \text{Eqn. 8}$$

This example has positive scalability with a bounded increase. If $T_1 \approx \bar{T}$, then Ψ levels off around a value of 2. This agrees with intuition. The throughput increase pays for the extra cost in exact proportion, and the shorter response time provides a bonus which is bounded because f is bounded as T goes to 0.

3.4.3 Model B: General Speed-Scaled Closed System with Fixed User Population

In this model, the system is closed instead of open and what it means is that, its load is generated by users or jobs which remain in the system, cycling, and creating a new request the moment that the previous response is over. When there are N users or jobs and a mean response time T , the system throughput is $\lambda = N / T$. More also, we consider any system architecture and behaviour with a single class of users, that is all jobs make the same demands, and an arbitrary starting point with $k = 1$. The cost is scaled by k , but the number of users is fixed at N and is not scaled, and the target mean response time is also fixed. Configuration B is again uniformly sped-up or time scaled by a factor k , hence every device and server is faster. Thus just as in Model A, $C_2 = kC_1$, $\lambda_2 = k\lambda_1$, and $T_2 = T_1 / k$, and $\Psi(k)$ is again given by (8). The intuition about this system is identical to that in Model A and the results agree in a similar way.

3.4.4 Model C: Closed Balanced Ideal Queueing Network, Scaled in Users and Speed

This is similar to the previous one except that it is more restricted in its assumptions about the system, which has a separable queueing network performance model with N jobs and k servers, with equal demands D seconds per response to all servers, and single, constant-rate servers. The throughput and response time are as follows:

$$\lambda = \frac{n}{[(N + k - 1)D]}, \quad T = \frac{n}{\lambda} = (N + K - 1)D$$

In configuration B, the number of jobs N , and the server speeds and costs are scaled by a factor k . Thus, configuration B has server demands of D/k seconds per response, at each of the K servers. Substituting into Equation 8 with $N_2 = kN_1$, $D_2 = D_1/k$, and $C_2 = kC_1$ we get:

$$\psi(k) = \frac{k(N_1 + K - 1)}{(kN_1 + K - 1)} \cdot \frac{(N_1 + K - 1)D_1 + \dot{T}}{(kN_1 + K - 1)\left(\frac{D_1}{k}\right) + \dot{T}}$$

$$\lim_{k \rightarrow \infty} \psi(k) = \frac{(N_1 + K - 1)[(N_1 + K - 1)D_1 + \dot{T}]}{N_1[N_1 D_1 + \dot{T}]} \quad \text{Eqn. 9}$$

By inspection, this limit is greater than 1.0, but if $K \ll N_1$ the limit approaches 1.0.

3.4.5 Model D: Asymptotic General Closed System Scaled in Users and Speed

This combines both model B and C, but only considers the asymptotic condition in which the system is effectively bottlenecked at its slowest server, even in the base configuration. The system throughput is effectively determined by the demand at this server, giving $\lambda \approx 1/D_{max}$. In configuration B, $D_{max,2} = D_{max,1}/k$, and thus throughput follows $\lambda_2 = \frac{1}{D_{max,2}} \approx k \cdot \lambda_1$. As $N_2 = kN_1$, and $C_2 = kC_1$, the response time is roughly constant at $T = kN / k\lambda = N / \lambda$, and $\Psi \approx 1.0$.

3.4.6 Model E: System with a Single Non-scalable Bottleneck and Increasing Users

This is likened to Model C which is a general closed system with a bottleneck, in which the bottleneck device cannot be speeded up, and hence limits λ to some constant value λ_{max} . The population N and costs C are proportional to k . Then, for large $kT \rightarrow N/\lambda_{max}$ and:

$$\psi(k) \approx \frac{\lambda_{max}}{\lambda_1} \cdot \frac{C_1}{kC_1} \cdot \frac{N_1/\lambda_1 + \dot{T}}{kN_1/\lambda_{max} + \dot{T}} \rightarrow \left(\frac{\lambda_{max}}{\lambda_1}\right)^2 \frac{1}{k^2} \quad \text{Eqn. 10}$$

Thus, the scalability declines as $1/k^2$, for any closed system that has a single dominant bottleneck.

3.4.7 Model F: Closed Balanced System Scaled by Replication of Servers and User Population

Like **Model C**, this system has a separable queueing model with balanced demands on K servers. In order to scale it up, each server is replicated k times while its load is divided equally among the k replicas of each server. In case there's no overhead for managing replicas, it seems intuitively clear that this is a perfectly scalable system. The scaling path: $N_2 = kN_1$; $K_2 = kK_1$; $D_2 = D_1/k$; $C_2 = kC_1$ must be considered. Following **Model C**, but with $K_2 = kK_1$ it can be seen that $\Psi > 1$ for all k , and for large k the limit is almost the same:

$$\lim_{k \rightarrow \infty} \psi(k) = \frac{(N_1 + K - 1)[(N_1 + K - 1)D_1 + \dot{T}]}{N_1[(N_1 + K)D_1 + \dot{T}]}$$

Moreover, if $K \ll N_1$, $\Psi \rightarrow 1:0$. The result remain the same for an unbalanced system.

3.4.8 Model G: The Effect of Overhead Costs

With reference to Model F, suppose that for $k > 1$ the demands D are augmented by coordination overhead, for example, to maintain overall system state data. The replicas, costs, and user population are all scaled by a factor k . An efficient coordination mechanism might limit the overhead cost to a slowly increasing amount of overhead, giving a scaling relationship for the server demands, such as: $D_1 = (D_1/k) + D_0 \log k$

Therefore, using the approach of Model C, Equation 6 result to the following:

$$\psi(k) = \frac{k(N_1 + K_1 - 1)D_1}{(kN_1 + K_1 - 1)\left(\frac{D_1}{k} + D_0 \log k\right)} * \frac{(N_1 + K_1 - 1)D_1 + \dot{T}}{\left[(kN_1 + K_1 - 1)\left(\frac{D_1}{k} + D_0 \log k\right) + \dot{T}\right]} \quad \text{Eqn. 11}$$

This decreases slowly towards zero as $k \rightarrow \infty$. Meaning, scalability is moderate for “small” values of k . If the baseline system is quite large, so that N_1 is large compared to K_1 and to $\dot{T}/D_1$, then Equation 11 can be simplified to $\psi \approx k/(1 + (D_0/D_1)k \log k)^2$, which is greater than unity for values of k that satisfy the approximate inequality $\sqrt{k} * \log k < D_0/D_1$. As an example, suppose that D_0/D_1 , which describes the relative magnitude of the coordination overhead, is 0.1. Then at $k = 10$, it gives $\Psi \approx 2.5$, and $\Psi > 1$ for values of k up to nearly 40. A larger overhead ratio will limit the scalability more.

3.4.9 Model H: A Closed System with Scaled Population and Target Response Times

This Model looks at a scaling path in which response degradation is accepted in proportion to a rising user population in an otherwise unscaled system. It is quite a different system goal, and illustrates the flexibility of the framework. Like the balanced closed system of Model C, but with constant values of C and D . The analysis of Model C result to the value function:

$$f = 1/(1 + T/\dot{T}) = 1/(1 + (N + K - 1)D/\dot{T})$$

If $\dot{T}_2 = k\dot{T}_1$ and $N_2 = kN_1$, then $\frac{d}{dk}f > 0$ and it is also well-known that $\frac{d}{dk}\lambda \geq 0$. Then, it can be deduced from Equation 7 that, ψ is an increasing function of k , and hence it approaches a constant limit greater than 1. This gives clear indication that the response time rises at the same asymptotic rate as the population. Therefore, it can be concluded that, if response time degradation is accepted in proportion to users and is included in the scalability function, closed balanced systems are infinitely scalable in population.

3.4.10 Model I: A Single Scaled Open Multiserver Queue

Sometimes systems are scaled up by adding servers and for that matter, this Model looks at an ideal multiserver queue (M/M/m queue) in which the number of servers m , the arrival rate λ , and the cost C are all scaled by a factor k . There is no server coordination overhead, and the queue shares the load in an ideal fashion, so the metric should show infinite scalability. The solution is well-known but quite lengthy and for that matter, it will not be shown here, but it does show that $\Psi > 1$ for all k , which supports the intuition.

As $k \rightarrow \infty$, the metric approaches the form:

$$\Psi \rightarrow \left(1 + \frac{1}{T} \left(S + \frac{\rho \cdot P_Q}{\lambda \cdot (1 - \rho)} \right) \right) / \left(\frac{1 + S}{T} \right) > 1 \quad \text{Eqn. 12}$$

Where S is the service time at any server, λ is the arrival rate, ρ is the utilization of each server ($\rho = \lambda s / m$), and P_Q represents the Erlang-C formula for the probability that all m servers are busy. The fraction with $\rho \cdot P_Q$ in the numerator approaches zero when k is large, and Ψ approaches 1 in the limit. The Models A-I give some confidence in applying it as it covers a wide range of well-understood systems and of scaling policies, and also reveal how the metric derived in this Thesis will evaluate different kinds of systems. Moreover, it also agrees with intuitive judgement. If we consider a steady state with one job at a time being executed, one after another, and fixed-size scalability, then a parallel computer is a closed system with replication and overhead, as in Model G (except the user population is not scaled). The time to completion is rewarded through the throughput term, with $\lambda = 1/T$. The QoS function considered in most metrics is simply $f=1$, as they only evaluate the time to completion which is already taken into account in the throughput. Before scaling a system, it is advisable to first

look at the type of system that you intend scaling in order to inform you of the choice of Model to be used.

3.4.11 Algorithm for Calculating Scalability Bound

The following are the detailed steps to follow in calculating the scalability bound:

Step 1. Determine the productivity for the base case, $F(1)$ by a detailed calculation.

Step 2. For each scale factor k , determine the scaled system configuration from the scaling strategy and then compute the total seconds of execution of each device, averaged per response, As:

1. Execution and overhead, which is determined and assigned to each device by the scaling strategy is calculated first.

2. The remaining execution demand is added up over the remaining tasks and spread (optimistically) over all the devices, so as to produce the most even distribution of the total demand, expressed in seconds of execution per response. Meaning, it is allocated without regard to allocating entire tasks to one device, but with regard to whether the device can do the work (so, CPU demand is spread over CPUs and disk demand over disks). Optimistic assumptions about overheads mean that they are set to the lowest value consistent with the scaling strategy; thus, if two tasks included in the remaining demand should be allocated separately (by the scaling strategy), internode communications overhead is included. The result of this step is a set of demands which may still be unequally distributed over the devices, because of constraints in spreading the workload.

Step 3. At scale k , set $C(k)$ to the cost of the scaled system, and find bounds on λ and T :

- a. set $\lambda(k)$ to the minimum of 1) the balanced system throughput bound for a queuing network with the same servers, and 2) the asymptotic throughput bound for the given set of demands
- b. Set $T(k)$ to the balanced job value
- c. Compute $F(k)$ from (6).

Step 4. Set the scalability metric bound to $\psi = F(k) / F(1)$, and then the bound-based scalability limit is the first value of k giving that y drops below the "moderate scalability" limit of $1 - \epsilon$.

The queueing network model with the evenly spread workload is constructed so that it intuitively gives a performance bound; that not withstanding, the relationship is not rigorously proven. The intuitive reasons for believing it gives a bound as:

- a. software resource constraints are ignored, which can only improve performance,
- b. allocation decisions which are enablers in the strategy are represented in the bound by the greatest possible degree of load balancing, which should give better performance than the best feasible allocation that respects task granularity, and
- c. overhead that is not explicitly required by the scaling strategy is omitted. The bounds can show the consequences of changing demands and power with k . Suppose that the scaling strategy resulted in a total demand (in seconds of execution, adding over all nodes) of $D(k) = g_1(k)$, the number of nodes (all equally fast) is $g_2(k)$, and there is a user delay (not included in the response time) of Z_0 . Then the bound calculation is:

$$D_{avg}(k) = D/g_2(k) = g_1(k)/g_2(k)$$

$$R(k) = D(k) + (N - 1)D_{avg})/(1 + Z_0/D(k))$$

$$= g_1(k) + (N - 1)(g_1(k)/g_2(k))(g_1(k)/(Z_0 + g_1(k)))$$

$$T_{(k)} = R(k) + Z_0$$

The bound on the scalability metric can then be expressed as:

$$\psi_{bnd}(k) = \frac{F_{bnd}(k)}{F(1)}$$

$$= \frac{\min \left\{ \frac{kN}{Z_0 + g_1(k) + (N-1) \frac{g_1(k)}{g_2(k)} * \frac{g_1(k)}{(Z_0 + g_1(k))}}, \frac{1}{D_{max}} \right\}}{C_k \left(1 + \frac{1}{T} \left(Z_0 + g_1(k) + (N-1) \frac{g_1(k)}{g_2(k)} * \frac{g_1(k)}{(Z_0 + g_1(k))} \right) \right) * F(1)}$$

Eqn. 13

When the system is saturated, both the numerator and denominator are dominated by the terms in the big round brackets multiplied by (N - 1). The direct effect of adding work (increasing $g_1(k)$) is always to decrease ψ . The direct effect of adding nodes is to increase $g_2(k)$ and $C(k)$ both, so as far as the bound is concerned the effect is neutral when the system is saturated, and harmful to scalability when it is not. The direct effect of causing a bottleneck node, due to a scaling path that does not allow the load to be properly balanced, is to increase D_{max} and decrease scalability through the last term in the numerator. All of these effects are expected, but the equation gives a picture of the order of the relationship. A second version of the bounds analysis, which is closer to a kind of approximation, is to use the bounding value for performance and productivity in the base case also. This puts all scale factors on an equal footing in regard to the looseness of the bounds. However, it reduces the certainty that the value of ψ_{bnd} is in fact a bound, since the denominator may be overestimated.

3.5 Overview of The Connection-Management System

The connection management system discussed in this thesis is based on the design and parameters of a real industrial prototype. It is a design which evolved out of a connection-management design described by (Jogalekar and Woodside, 1997). Fig. 3.3 shows the major

components in a prototype connection management system for virtual private networks, intended to support applications such as video conferencing. The prototype was heavily influenced by standards such as G.805. It was designed to be able to:

1. set up a virtual private network joining user specified end-points, and allocating the network resources in such a manner as to meet the QoS requirement,

2. Manage a variety of heterogeneous switching equipment, for the purpose of setting up end-to-end connections,

3. Use the allocated resources of the virtual private network and let the user set-up/tear down connections arbitrarily, among any of the sites. The prototype was implemented using a network of workstations running UNIX, with DCE middleware to handle inter task communications and transparency, and a backbone network based on a SONET OC-12 (622 Mbit/s) optical fiber ring with proprietary switching equipment on which cross-connections can be made or released as required. The software tasks can be roughly classified into three logical layers:

1. The topology layer that deals with the connection topology of the virtual private network (VPN), connecting all the user-specified endpoints (e.g., the User-Network Interface identifiers UNI's in the case of an ATM network). Once a virtual private network is established, the objects in the topology layer can directly communicate with the lowest layer (called SONET here), in order to set up virtual channels over this VPN.

2. The virtual path (VP) layer, that deals with connecting all the sites in a virtual private network with a virtual path. This corresponds to provisioning the network resources to meet user specified bandwidth and QoS, to support future connections. The SONET layer that supports a virtual path by setting up appropriate connections on the SONET ring.



Fig. 3.3 The connection management System infrastructure

From Fig. 3.3, the client tasks represents the users that set up (or dismantle) the virtual private network and set up (or dismantle) connections on an existing virtual private network. The clients could be the software tasks that manage higher level applications, e.g., a video conferencing system that uses the given connection management system. The clients interact with the topology layer to set up the virtual private network, as well as the connections on it (VC's or the virtual channels). The frequency of setting up/releasing a VPN, which is like a leased line, is much lower than that of setting up/releasing temporary connections by a ratio of 1:50.

1. Topo_setup and Topo_delete: these tasks belong to the topology layer discussed above, and support setting up VPNs as well as connections within a VPN. The necessary routing functions are built into the setup entries of these tasks and of their servers.

2. VP: This task sets up and deletes virtual paths (VPs) that make up a VPN.

3. SONET: This task manages the fibre-level port-to-port connections required to support the setting up of the VP layer trails, which in turn help set up the VPN.

Subnet_connect: This task directly controls the SONET network elements.

Database: The database stores objects related to the various functional layers in the system and provides state data to all the functions. The database, which is accessed heavily by almost all the tasks in the system, clearly is a potential hot spot in the system. By measurement it was verified that the database indeed had the greatest demands for both VPN setup/release, as well as connection setup/release, and would limit scalability if its capacity were not increased. One approach to this is database replication, which was considered as an element in the scaling strategy. As we shall see, the hazard in replication is heavy overhead. The prototype system was instrumented and measured to obtain workload parameters for the performance model, which was used to evaluate the scalability.

3.6 Using the Metric to Scale the Connection Management System

The scaling strategy was to introduce replications of the database, using the location-based replication paradigm described by (Trantafilou and Taylor, 1995). For each database replica, an additional processor was also added to the system. (note that the location-based paradigm was motivated by reliability as well as performance, and the reliability effects are not rewarded in the value function f used here.) The scale factor was set to be the number of database replicas. A fixed number of five processors was provided to run the other tasks in a fixed configuration, and the number of users was taken as a scalability enabler. Further enablers that

were not used could have been the allocation of the tasks other than the database tasks to the processors, and replicas and additional processors for the other functions. For each scale factor a performance model was set up with the replicas and their overheads, with overhead amounts calculated from the number of replicas, and the requests sent from any client entry to the database task were equally divided among all the replicas. The fixed remote invocation overheads were incorporated in the execution demands of the task entries. The fact that the accesses to the database replicas were symmetric happens to permit a special efficient approximation for symmetric replication of subsystems to be used in the solver (Pan, 1996). In order to model the consistency management overhead (in terms of extra execution), each replica of the database is associated with a transaction overhead pseudo-task on the same CPU. The transaction overhead task accounts for the synchronous and asynchronous broadcasting overheads, locking overheads, etc., for consistency management, and the calls made by the database entries to the overhead task during the operation prepare, commit, and abort phases are proportional to the number of database replicas in the system. The number of write transactions is significant, but the granularity of the database objects is small, so the probability of conflict on locks was assumed to be negligible and lock queueing delays were not modeled. However, the execution overheads of locking were substantial and were included. The response of the system was modeled as a cycle of effort for one conference, including setting up and tearing down five virtual channels for a video conference between the two sites, plus one time in ten it included setting up a VPN, as well. The cycle had a target time of 15 minutes ($\bar{T} = 15 \text{ min.}$). Load was generated by a number of users, who were modeled as having a "thinking time" of 10 minutes, between one cycle and the next. The provisioning cost for the base configuration, including one copy of the database server, and one processor per software task, is taken as GHc100,000. Each extra copy of the database server (including a new dedicated processor) is

assumed to cost an additional GHc 5,000. This gives a cost per unit time of the form Constant $(1 + 0.05k)$. The reference configuration of the system had a single database copy, and was also optimized with respect to the number of clients, giving a reference productivity of 702 cycles of activity per hour per unit cost, and a reference throughput of 95 cycles of activity per hour. (That is, setting up and tearing down 9.5 virtual private networks, and setting up and tearing down about 475 virtual channels per hour).

3.6.1 How to obtain the values for the Connection Management system

$$\text{Productivity } F(k) = \frac{\lambda(k) * f(k)}{C(k)}$$

From the model the quality of service of the value function determined are as follows

Scale Factor	1	2	3	4	5	6	7
Value function	0.021606	0.020460	0.019666	0.018819	0.017988	0.017413	0.016665

To calculate the cost:

$$C(k) = 1 + .005k$$

When $k = 1$ Then

$$C(k_1) = 1 + 0.05 * 1 = 1.05$$

When $k = 2$ Then

$$C(k_2) = 1 + 0.05 * 2 = 1.1$$

When $k = 3$ Then

$$C(k_3) = 1 + 0.05 * 3 = 1.15$$

When $k = 4$ Then

$$C(k_4) = 1 + 0.05 * 4 = 1.2$$

When $k = 5$ Then

$$C(k_5) = 1 + 0.05 * 5 = 1.25$$

When k = 6 Then

$$C(k_6) = 1 + 0.05 * 6 = 1.3$$

When k = 7 Then

$$C(k_7) = 1 + 0.05 * 7 = 1.35$$

From the model the Throughput measured are as follows

Scale Factor	1	2	3	4	5	6	7
Throughput	95	108.62	111.18	112.01	112.26	110.95	110.94

Therefore:

$$\text{Productivity } F(k_1) = \frac{\lambda(k_1) * f(k_1)}{C(k_1)}$$

$$\text{Productivity } F(k_1) = \frac{95 * 0.021606}{1.05} = 1.95483$$

$$\text{Productivity } F(k_2) = \frac{\lambda(k_2) * f(k_2)}{C(k_2)}$$

$$\text{Productivity } F(k_2) = \frac{108.62 * 0.020460}{1.1} = 2.02033$$

$$\text{Productivity } F(k_3) = \frac{\lambda(k_3) * f(k_3)}{C(k_3)}$$

$$\text{Productivity } F(k_3) = \frac{111.18 * 0.019666}{1.15} = 1.90128$$

$$\text{Productivity } F(k_4) = \frac{\lambda(k_4) * f(k_4)}{C(k_4)}$$

$$\text{Productivity } F(k_4) = \frac{112.01 * 0.018819}{1.2} = 1.75660$$

$$\text{Productivity } F(k_5) = \frac{\lambda(k_5) * f(k_5)}{C(k_5)}$$

$$\text{Productivity } F(k_5) = \frac{112.26 * 0.017988}{1.25} = 1.61547$$

$$\text{Productivity } F(k_6) = \frac{\lambda(k_{10}) * f(k_{10})}{C(k_{10})}$$

$$\text{Productivity } F(k_6) = \frac{110.95 * 0.017413}{1.3} = 1.48613$$

$$\text{Productivity } F(k_7) = \frac{\lambda(k_{15}) * f(k_{15})}{C(k_{15})}$$

$$\text{Productivity } F(k_7) = \frac{110.94 * 0.016665}{1.35} = 1.36949$$

To Calculate the Scalability Metric Value

$$\psi(k_1, k_2) = \frac{F(k_2)}{F(k_1)} = \frac{2.02033}{1.95483} = 1.0335$$

$$\psi(k_1, k_3) = \frac{F(k_3)}{F(k_1)} = \frac{1.90128}{1.95483} = 0.9726$$

$$\psi(k_1, k_4) = \frac{F(k_4)}{F(k_1)} = \frac{1.75660}{1.95483} = 0.8986$$

$$\psi(k_1, k_5) = \frac{F(k_5)}{F(k_1)} = \frac{1.61547}{1.95483} = 0.8264$$

$$\psi(k_1, k_6) = \frac{F(k_6)}{F(k_1)} = \frac{1.48613}{1.95483} = 0.7602$$

$$\psi(k_1, k_7) = \frac{F(k_7)}{F(k_1)} = \frac{1.36949}{1.95483} = 0.7006$$

3.6.2 Calculating the Scalability Bound of The Connection Management System

Step 1. The base configuration with six processors is optimized with respect to the number of clients, to obtain 23 clients, 95 operation units per hour and productivity $F = 1.95 \times 10^{-5}$ units/hour.

Step 2. At each scale factor, with k database replicas and k database processors, the balanced demand is calculated, including the overheads. In this case,

Total demand, $D = 14.44 + (22.11k)$ sec

Average demand, $D_{avg}(k) = 14.44 + (22.11k)/(k + 5)$ sec

$D_{max} = 35.08$ sec

Response time = $D + (N - 1)D_{avg} / (1 + (Z/D))$

$Z_0 = 600$ sec

Cost, $C_{(k)} = 1 + 0.05k$ units/ sec

Steps 3 and 4. The solution gives the response time $T = D + (N - 1)D_{avg}/(1 + (Z/D))$ and throughput $\lambda = N(Z_0 + T)$ for the balanced system. Substituting into Equation 13 gives the following expression for the scalability bound:

$$\psi_{bnd}(k) = \frac{5.13 \times 10^4 \min \left\{ \frac{kN}{Z_0 + D + (N-1) \frac{D}{k+5} * \frac{D}{Z_0 + D}}, \frac{1}{D_{max}} \right\}}{(1 + 0.05k) \left(1 + \frac{1}{T} * (Z_0 + D + (N-1) \frac{D}{(k+5)} * \frac{D}{(Z_0 + D)}) \right)} \quad \text{Eqn. 14}$$

3.6.3 Obtaining the measures for the Scalability bound

From the above data:

When $k = 1$, Then

Total demand, $D = 14.44 + (22.11k)$ sec

Total demand $D(k_1) = 14.44 + (22.11 * 1)$

$D(k_1) = 36.55$ sec

Average demand, $D_{avg}(k_1) = 14.44 + (22.11k)/(k + 5)$ sec

$D_{avg}(k_1) = 14.44 + (22.11 * 1)/(1 + 5)$

$D_{avg}(k_1) = 14.44 + (22.11)/(6)$

$D_{avg}(k_1) = 6.0197$

$D_{max} = 35.08$ sec

$Z_0 = 600$ sec

Cost, $C_{(k)} = 1 + 0.05k$ units/ sec

$C_{(k_1)} = 1 + 0.05k$ units/ sec

$C_{(k_1)} = 1 + 0.05 * 1$

$$C_{(k_1)} = 1.05$$

Substituting the values into the Equation:

$$\psi_{bnd}(k) = \frac{5.13 \times 10^4 \min \left\{ \frac{kN}{Z_0 + D + (N-1) \frac{D}{k+5} * \frac{D}{Z_0 + D}}, \frac{1}{D_{max}} \right\}}{(1 + 0.05k) \left(1 + \frac{1}{T} * (Z_0 + D + (N-1) \frac{D}{(k+5)} * \frac{D}{(Z_0 + D)}) \right)}$$

$$\psi_{bnd}(k_1) = \frac{5.13 \times 10^4 \min \left\{ \frac{1 \times 80}{600 + 36.55 + (80-1) \frac{36.55}{1+5} * \frac{36.55}{600 + 36.55}}, \frac{1}{35.08} \right\}}{(1 + 0.05 * 1) \left(1 + \frac{1}{900} * (600 + 36.55 + (80-1) \frac{36.55}{(1+5)} * \frac{36.55}{(600 + 36.55)}) \right)}$$

$$\psi_{bnd}(k_1) = \frac{5.13 \times 10^4 \min \left\{ \frac{80}{636.55 + (79) \frac{36.55}{6} * \frac{36.55}{636.55}}, \frac{1}{35.08} \right\}}{(1.05) \left(1 + \frac{1}{900} * \left(636.55 + (79) \frac{36.55}{(6)} * \frac{36.55}{(636.55)} \right) \right)} = 1.33$$

Solving for $k_2, k_3, k_4, k_5, k_6, k_7, k_8$ gives the following results in the table bellow

Scale Factor	1	2	3	4	5	6	7	8
Scalability Bound	1.33	1.23	1.12	1.05	0.95	0.90	0.84	0.80

3.7 Applying The metric to Asymptotic Bounds (Case Study 1)

The Ghana Meteorological Agency used to have twenty synoptic weather observation stations across the entire regions of Ghana which was geographically distributed. The stations were using IBM3790s to process their weather information data but was providing a response time which was not acceptable. The Agency thought of the need to have selection, acquisition and

conversion cycle for the period of three years, but until then, the systems required interim upgrade. Both IBM8130s and IBM8140s were in the position to run the software application called METEOSOFT. The Agency then considered to use these two systems for the time been of three year transition-period. The Agency got convinced after deliberations with the vendor that, using IBM8130s could offer a performance improvement factor ranging from 1.5 to 2 over that of IBM3790s while a performance improvement factor ranging from 2 to 3.5 was anticipated with the use of IBM8140s. There wasn't any emphasis on the significance of the performance improvement factor. The Agency had to initiate a modeling studies to determine the weather observations stations that will host the cheaper IBM8130 system. From the studies it came to light that, both IBM8130 and IBM8140 systems had a faster disk than that of IBM3790. The Central Processing Unit (CPU) speed of IBM8130 processor was quite slower than that of the IBM3790. On the contrary, IBM8140 was roughly 1.5 times faster. From the above information, benchmark-experiments on IBM3790 and IBM8140 systems were carried out and also, "live-measurements" of the existing IBM3790 systems.

Table 3.5 Data values of case study 1

System	Service Demands / seconds	
	CPU	Disk
3790 (observed)	4.6	4.0
8130 (estimated)	5.1	1.9
8140 (estimated)	3.1	1.9

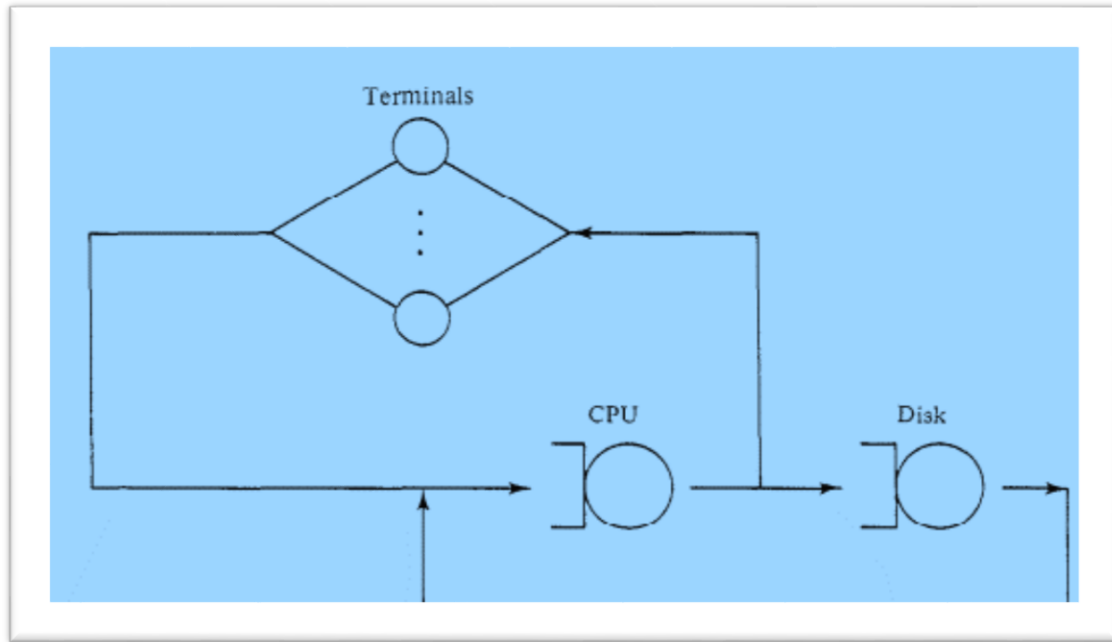


Fig. 3.4 - Case Study 1 Model Diagram

With the service demands established, the metric was used to assess the performance to be expected from each of the three systems. Figure 3.4 shows the queueing network. (Some stations had two physical disk drives, but the disk controller did not permit them to be active simultaneously. For that matter, having only a single disk service center in the model is enough.) The parameters are:

K , the number of service centers (2) ;

D_{max} , the largest service demand (4.6 seconds for the 3790, 5.1 for the 8130, and 3.1 for the 8140);

D , the sum of the service demands (8.6, 7.0, and 5.0, respectively) ;

The type of the observation stations is terminal;

Z, the average think time (an estimate of 60 seconds was used).

3.8 Calculating the Throughput bound and Average Response Time of the systems

Following the Algorithm on section 3.3.5: The Algorithm for Closed Model Asymptotic Bound,

Given:

D_{max} of IBM3790 = 3.6 IBM8130 = 5.1 And IBM8140 = 3.1

D of IBM3790 = 8.6 IBM8130 = 7.0 and IBM8140 = 5.0

Z = 60 sec

From the above data, the intersection point (N^*) is calculated using the formulae $N^* = \frac{D+Z}{D_{max}}$

The intersection point of IBM3790 is calculated as follows:

$$N^* = \frac{D + Z}{D_{max}} = \frac{8.6 + 60}{4.6} = \frac{68.6}{4.6} = 14.9130 \approx 15$$

The intersection point of IBM8130 is calculated as follows:

$$N^* = \frac{D + Z}{D_{max}} = \frac{7.0 + 60}{5.1} = \frac{67.0}{5.1} = 13.1373 \approx 13$$

The intersection point of IBM3790 is calculated as follows:

$$N^* = \frac{D + Z}{D_{max}} = \frac{5.0 + 60}{3.1} = \frac{65.0}{3.1} = 20.9677 \approx 21$$

Since the intersection point (N^*) of IBM3790 is approximately 15, it means that the bound on throughput will pass through the point **(0, 0) and $(1, \frac{1}{D_{max}})$** for $N \leq N^*$ (all values of N less than or equal to 15). When $N = 16$ the throughput will pass through the point **$(0, \frac{1}{D_{max}})$ and $(1, \frac{1}{D_{max}})$** for $N \geq N^*$ (all values of N greater than or equal to 16).

This means that the first point will be (0, 0) and the last point will be when $N = 16$

Hence:

$$(16, \frac{1}{D_{max}}) = (16, \frac{1}{4.6}) = (16, 0.217)$$

Therefore the throughput points of IBM3790 are: **(0, 0) (16, 0.22)**

The same way, the intersection point (N^*) of IBM8130 is approximately 13, for $N \leq N^*$ hence when $N = 14$ the throughput will pass through the point as follows:

$$(14, \frac{1}{D_{max}}) = (14, \frac{1}{5.1}) = (14, 0.196)$$

Therefore the throughput point of IBM8130 are: **(0, 0) (14, 0.20)**

The intersection point (N^*) of IBM8140 is approximately 21, for $N \leq N^*$ hence when $N = 22$ the throughput will pass through the point as follows:

$$(22, \frac{1}{D_{max}}) = (22, \frac{1}{3.1}) = (22, 0.323)$$

Therefore the throughput point of IBM8130 are: **(0, 0) (22, 0.32)**

Table 3.6 Throughput values for case study 1

System	Bound on Throughput values to plot.
3790	(0 , 0) (16 , 0.217)
8130	(0 , 0) (14 , 0.196)
8140	(0 , 0) (22 , 0.323)

Using the optimistic bound equation **(0, D) and (1, D)** for $N \leq N^*$ for the Response Time.

Since the calculation of the intersection point (N^*) gave the values, $N = 15$ for IBM3790, $N = 12$ for IBM8130 and $N = 21$ for IBM8140. This implies that, the positive maximum response time based on the N values will be as follows:

Table 3.7 Response Time values for case study 1

System	Bound on Response Time values to plot.
3790	(0 , 8.6) (15 , 8.6)
8130	(0 , 7.0) (12 , 7.0)
8140	(0 , 5.0) (21 , 5.0)

3.8.1 Case Study 2

This case study looks at the use of the metric in asymptotic bounds to assess the impact of modifications to an existing system. Let's consider a simplified interactive system for which the following measurements have been obtained

Table 3.8 Data values for case study 2

DATA	MEANING
T = 900 seconds	Length of the measurement interval
B1 = 400 seconds	CPU disk
B2 = 100 seconds	Slow disk busy
B3 = 600 seconds	Fast disk busy
C = 200 jobs	Completed Jobs
C2 = 2,000	Slow disk operations
C3 = 20,000	Fast disk operations
Z = 15 seconds	Think time

The service demands per job are $D_1 = 2.0$, $D_2 = 0.05$ and $D_3 = 3.0$. The visit counts to the disks are $V_2 = 10$ and $V_3 = 100$. The service times per visit to the disks are $S_2 = 0.05$ and $S_3 = 0.03$.

3.8.2 Procedure for scaling the system

Four improvements that can be made to the system using the models of the metric are considered here. These are listed below, along with an indication of how each would be reflected in the parameters of the model:

1. Replace the CPU with one that is twice as fast. $D_1 \leftarrow 1$
2. Shift some files from the faster disk to the slower disk, balancing their demands. Consider only the primary effect, which is the change in disk speed, and ignore possible secondary effects such as the fact that the average size of blocks transferred may differ between the two disks. The new disk service demands are derived as follows.

$V_2 + V_3 = 110$. Because $S_2 = 0.05$ and $S_3 = 0.03$, this is the same as

$$\frac{V_2 S_2}{0.05} + \frac{V_3 S_3}{0.03} = 110$$

Aiming at $D_2 = V_2 S_2 = V_3 S_3 = D_3$

$$D_2 \left[\frac{1}{0.05} + \frac{1}{0.03} \right] = 110$$

And $D_2 = D_3 = 2.06$ dividing by the appropriate service times, gives the new visit counts:

$$V_2 = 41 \text{ and } V_3 = 69$$

3. Add a second fast disk (center 4) to handle half the load of the busier existing disk. Once again, consider only the primary effects of the change. $K \leftarrow 4, D_3 \leftarrow 1.5, D_4 \leftarrow 1.5$

4. The three changes made together: the faster CPU and a balanced load across two fast disks and one slow disk. Service demands become $D_1 = 1, D_2 = 1.27, D_3 = 1.27, \text{ and } D_4 =$

1.27. These are derived in a manner similar to that employed above. Since $V_2 + V_3 + V_4 = 110$, to ensure that $D_2 = D_3 = D_4$:

$$\frac{V_2 S_2}{0.05} + \frac{V_3 S_3}{0.03} + \frac{V_4 S_4}{0.03} = 110$$

$$D_2 \left[\frac{1}{0.05} + \frac{1}{0.03} + \frac{1}{0.03} \right] = 110$$

$$D_2 = D_3 = D_4 = \left[\frac{0.0015}{0.13} \right] = 110 = 1.27$$

3.9 Overview of the Call Processing System

The handling and processing of voice and video calls is a critical function provided by IP telephony systems. This functionality is handled by some type of call processing entity or agent. Given the critical nature of call processing operations, it is important to design unified communications deployments to ensure that call processing systems are scalable enough to handle the required number of users and devices and are resilient enough to handle various network and application outages or failures.

3.9.1 Applying the metric to call processing system

The metric is applied to the call processing system of digital telephony, based on proprietary message oriented middleware. The objective is to assess the following cardinal points.

1. Up to what point a product would be scalable, if built using the same basic design decisions,
2. How much investment should be made in the hardware and software components of the system for supporting different numbers of users, and lastly

3. The impact of the location service-based replication model for database transactions, (Trantafiliou and Taylor, 1995).

3.9.2 The Call Processing System Architecture

For the purpose of this thesis, this system differs from the traditional call processing systems used in digital telephony. It is implemented using a message-oriented middleware based on a fast ATM network. The motive is to minimize the latency, and to bypass the overheads in RPC stubs, the TCP/IP protocol, etc., in order to make the distributed call processing as fast as possible. The system is based on some of the concepts dealt with in (Eicken et al, 1995). The U-Net communication architecture uses a virtual view of the network interface that can be directly accessed at user level, allowing direct access to high-speed communication devices. Removing/bypassing the communication subsystem's boundary with the application-specific protocols achieves efficient communication protocols with reduced system call overheads, and more importantly, allows buffer management at the user level. Multiplexing/ demultiplexing is embedded directly into the network interface. Thus, the network interface is virtualized, and each process has an illusion that it owns the direct interface to the network. This enables abstracting out the network interface for some applications, while still supporting legacy protocols through the kernel. Dedicated ATM virtual channel circuits are used for call setup, with additional channels for the actual voice traffic. This scalability analysis assumes that there is sufficient ATM network capacity for the voice traffic, so it is not modeled. The model concentrates on the objects that collaborate to set up an end-to-end call connection.

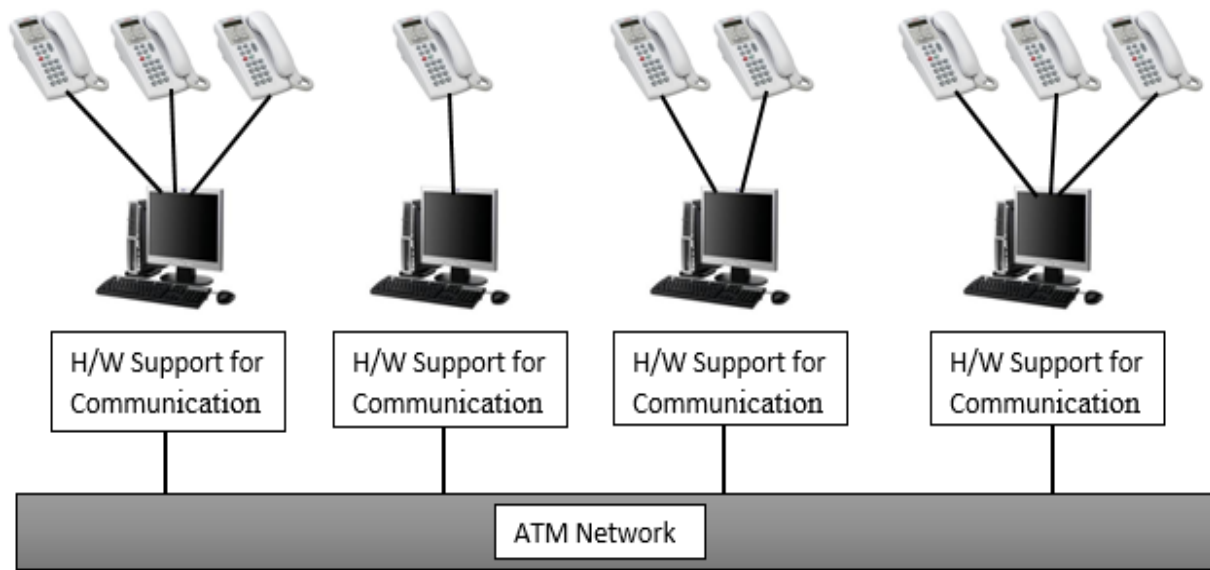


Fig. 3.5 Call Processing System's Hardware infrastructure

3.9.3 The Hardware Platform and The Processing Steps

Fig. 3.5 Depicts the hardware platform used in the prototype call-processing system. The host systems (IBM Power PCs running AIX, 133 MHz) are connected to an ATM network. Each host has the call processing software that supports the two half-call models, one half for call origination and one for call termination (McDonald, 1990). Any process that wishes to access the network creates one or more objects called endpoints, allocates memory for storing the messages (called the communication segment) and creates a set of send, receive, and free message queues with each endpoint. To send a message, a user process composes the data in its communication segment and pushes a descriptor for the message onto its send queue. The network interface, which is embedded in an integrated device driver, then picks up the message and sends it over the existing connection in the ATM network. Incoming messages are demultiplexed and transferred to the appropriate communication segment, and a message descriptor is pushed on the corresponding receive queue. Each process polls its receive queue periodically, to check for arrivals. The system has a "true zero copy" architecture in which the

data is transferred from the sender's communications segment to the receiver's, without intermediate buffering. The communication segments span the process address space and the sender specifies an offset within the destination communication segment, at which the message data is to be deposited directly by the network interface. Error checking and correction is handled at the application level. In this thesis, it is assumed that the communications medium is sufficiently reliable, and sufficient memory is available, to let us ignore the performance effects of recovery from errors and buffer overflows.

3.9.4 Software Organization and Layered Model

Fig. 3.6 demonstrates the software involved in a call, at the level of processes and interactions. The figure depicts a layered queueing network model which, as well as the software tasks, includes components for all the Users, the hardware network drivers, and the network delay, as "tasks" with their own delays. Nevertheless, it doesn't model the individual software objects. The prototype which was measured had two nodes, and invoked a total of about 80 objects per call, among all the collaborating processes. Call setup is managed by an originating half-call-setup process communicating with a terminal half-call-setup process. The originating process uses allocation server to decide which database replica to use, and then obtains authentication data and network addresses from the database. Almost all the database requests in normal operation are reads, as writes only occur for new customers, changes in a customer's service profile, and changes in the physical network. Each node has a sufficient number of agents, and may or may not have one or both of a database replica as well as location server replica.

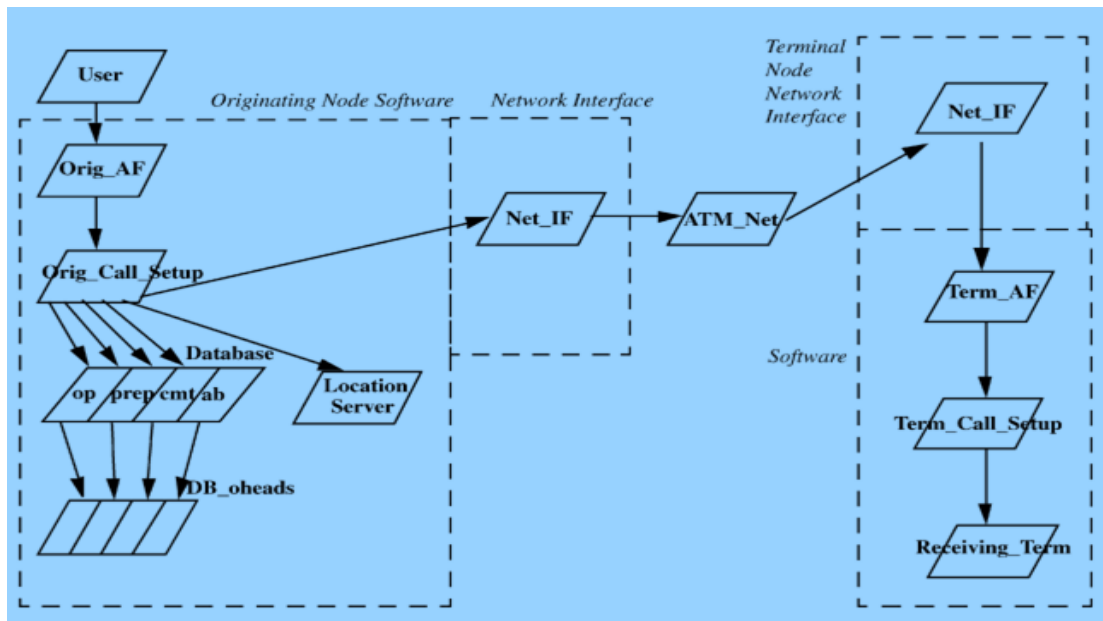


Fig. 3.6 Layered model for the call processing software, including the processes and interaction (Jogalekar, et al, 1997)

3.9.5 Scaling strategy for the call processing system

The scale factor is the number of CPUs, and the scalability enablers are the allocation of software objects, the location server, replication of the database, as well as the number of clients. Threads are not allocated explicitly, since they are assumed to be inexpensive, and are provided as a thread-per request. The scalability was analyzed over a scale factor interval of 1-15. The results of the optimization are presented in Table 4.1. The same data is plotted in Fig.4.4b.

3.9.6 How to obtain the values for the call processing system

$$\text{Productivity } F(k) = \frac{\lambda(k) * f(k)}{C(k)}$$

From the model the quality of service of the value function determined are as follows:

Scale Factor	1	2	3	4	5	10	15
Value function	0.160613	0.147285	0.146618	0.142691	0.128173	0.133703	0.130912

To calculate the cost:

$$C(k) = k * \frac{db}{10}$$

When k = 1 Then

$$C(k_1) = 1 * \frac{1}{10} = 1.1$$

When k = 2 Then

$$C(k_2) = 1 * \frac{1}{10} = 2.1$$

When k = 3 Then

$$C(k_3) = 3 * \frac{1}{10} = 3.1$$

When k = 4 Then

$$C(k_4) = 4 * \frac{1}{10} = 4.1$$

When k = 5 Then

$$C(k_5) = 5 * \frac{4}{10} = 5.4$$

When k = 10 Then

$$C(k_{10}) = 10 * \frac{2}{10} = 10.2$$

When k = 15 Then

$$C(k_{15}) = 15 * \frac{2}{10} = 15.2$$

From the model the Throughput measured are as follows:

Scale Factor	1	2	3	4	5	10	15
Throughput	1.0958	2.1273	3.0531	3.3848	3.6485	4.4934	4.4934

Therefore, to calculate the productivity:

$$\text{Productivity } F(k_1) = \frac{\lambda(k_1) * f(k_1)}{C(k_1)}$$

$$\text{Productivity } F(k_1) = \frac{1.0958 * 0.1606}{1.1} = 0.1600$$

$$\text{Productivity } F(k_2) = \frac{\lambda(k_2) * f(k_2)}{C(k_2)}$$

$$\text{Productivity } F(k_2) = \frac{2.1273 * 0.147285}{2.1} = 0.1492$$

$$\text{Productivity } F(k_3) = \frac{\lambda(k_3) * f(k_3)}{C(k_3)}$$

$$\text{Productivity } F(k_3) = \frac{3.0531 * 0.146618}{3.1} = 0.1444$$

$$\text{Productivity } F(k_4) = \frac{\lambda(k_4) * f(k_4)}{C(k_4)}$$

$$\text{Productivity } F(k_4) = \frac{3.3848 * 0.142691}{4.1} = 0.1178$$

$$\text{Productivity } F(k_5) = \frac{\lambda(k_5) * f(k_5)}{C(k_5)}$$

$$\text{Productivity } F(k_5) = \frac{3.6485 * 0.128173}{5.4} = 0.0866$$

$$\text{Productivity } F(k_{10}) = \frac{\lambda(k_{10}) * f(k_{10})}{C(k_{10})}$$

$$\text{Productivity } F(k_{10}) = \frac{4.4934 * 0.133703}{10.2} = 0.0589$$

$$\text{Productivity } F(k_{15}) = \frac{\lambda(k_{15}) * f(k_{15})}{C(k_{15})}$$

$$\text{Productivity } F(k_{15}) = \frac{4.4934 * 0.130912}{15.2} = 0.0387$$

To Calculate the Scalability Metric Value:

$$\psi(k_1, k_2) = \frac{F(k_2)}{F(k_1)} = \frac{0.1492}{0.1600} = 0.9325$$

$$\psi(k_1, k_3) = \frac{F(k_3)}{F(k_1)} = \frac{0.1444}{0.1600} = 0.9025$$

$$\psi(k_1, k_4) = \frac{F(k_4)}{F(k_1)} = \frac{0.1178}{0.1600} = 0.7363$$

$$\psi(k_1, k_5) = \frac{F(k_5)}{F(k_1)} = \frac{0.0866}{0.1600} = 0.5413$$

$$\psi(k_1, k_{10}) = \frac{F(k_{10})}{F(k_1)} = \frac{0.0589}{0.1600} = 0.3681$$

$$\psi(k_1, k_{15}) = \frac{F(k_{15})}{F(k_1)} = \frac{0.0387}{0.1600} = 0.2419$$

The factors which have to be balanced in the optimization of the scalability enablers include:

- a. Collocation and distribution of software tasks as Collocation reduces the remote invocation overhead but may cause a load imbalance in the overall system.
- b. Replication of the database, which affects load balancing, remote invocation costs, overheads for coordination (and thus, latency), and system cost.



CHAPTER FOUR

RESULTS AND DISCUSSION

4.1 Introduction

This chapter explains the results of the two case studies, call processing system, and connection management system as the metric is used to scale and analyze their performance and output.

4.2 Results and Discussions of Case Study 1

To calculate the bounds, the Algorithm for Closed Model Asymptotic Bound in section 3.3.5 needs to be adhered to. Applying the Algorithm to the model of each of the three systems leads to the optimistic bounds graphed in Fig. 4.1 (a) and (b). These results shows that, at high loads IBM3790s performance will be superior to IBM8130 and is due to the fact that IBM8130's CPU is slower and hence is the bottleneck device. This means that, instead of performance-gain, ranging from 1.5 to 2, it rather gave performance-degradation in migrating from IBM3790s to that of IBM8130s anytime the number of active terminals surpass certain threshold. Fig 4.1 (a) and (b) shows a performance-gain in moving from IBM3790s to IBM8140s even though it isn't two or more of the expected factor. Based on the study, a benchmark-tests were performed to determine whether it will be prudent to move IBM8130s in the Agency's transitional-plan. It became evidently clear that, the performance of IBM8130s will be worse than IBM3790s when the number of terminals was approximately fifteen or more, while the performance-gain of IBM8130s over IBM3790s at lighter-loads will be insignificant. For these reasons, there is no need of the Ghana Meteorological Agency investing in 8130s for any of their weather observation stations in Ghana as the metric showed clearly how the systems will perform.

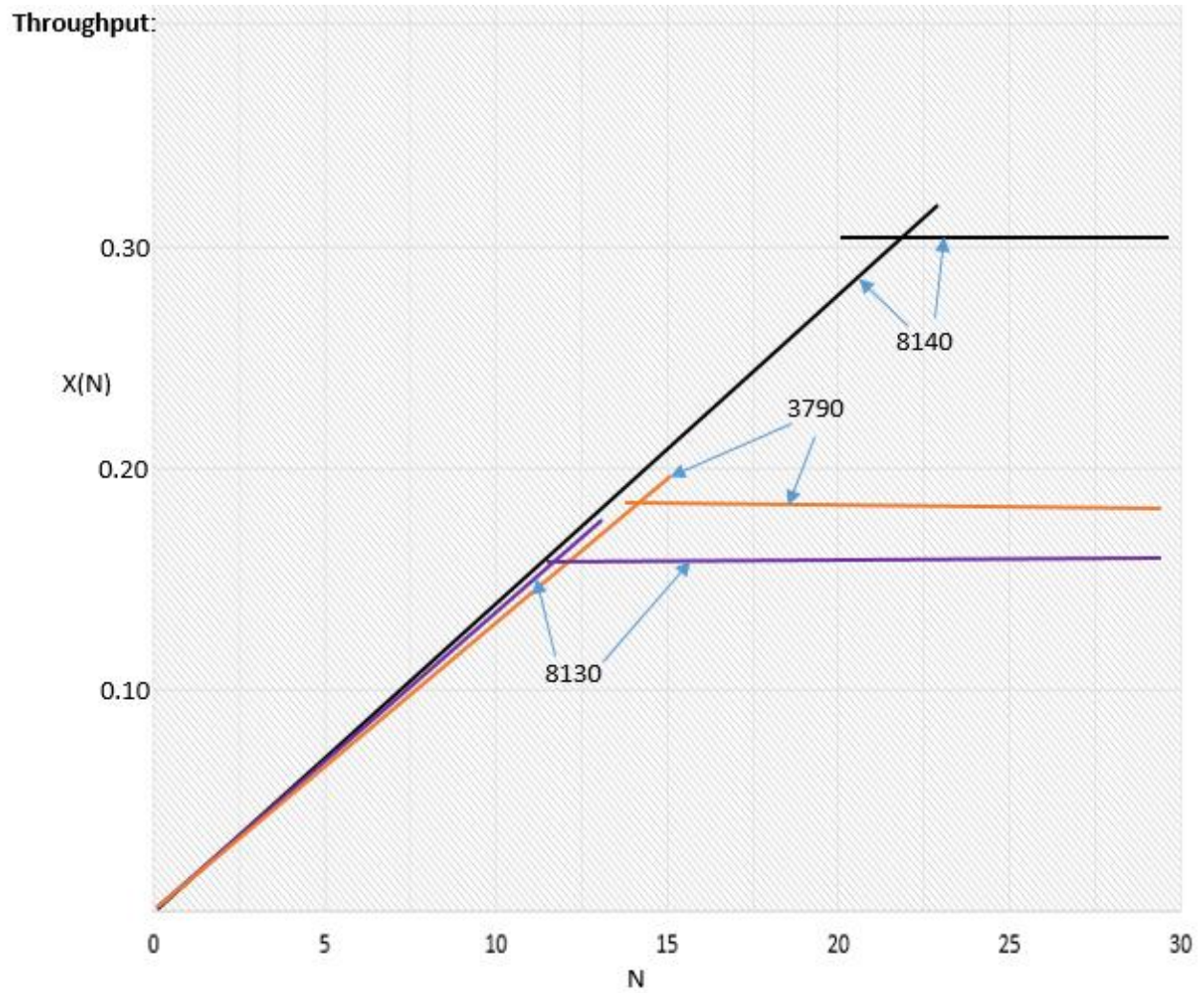
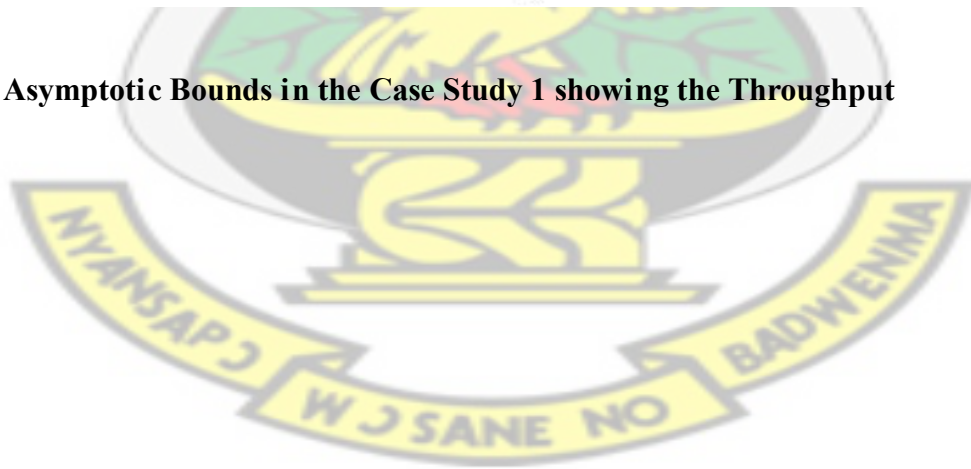


Fig. 4.1a Asymptotic Bounds in the Case Study 1 showing the Throughput



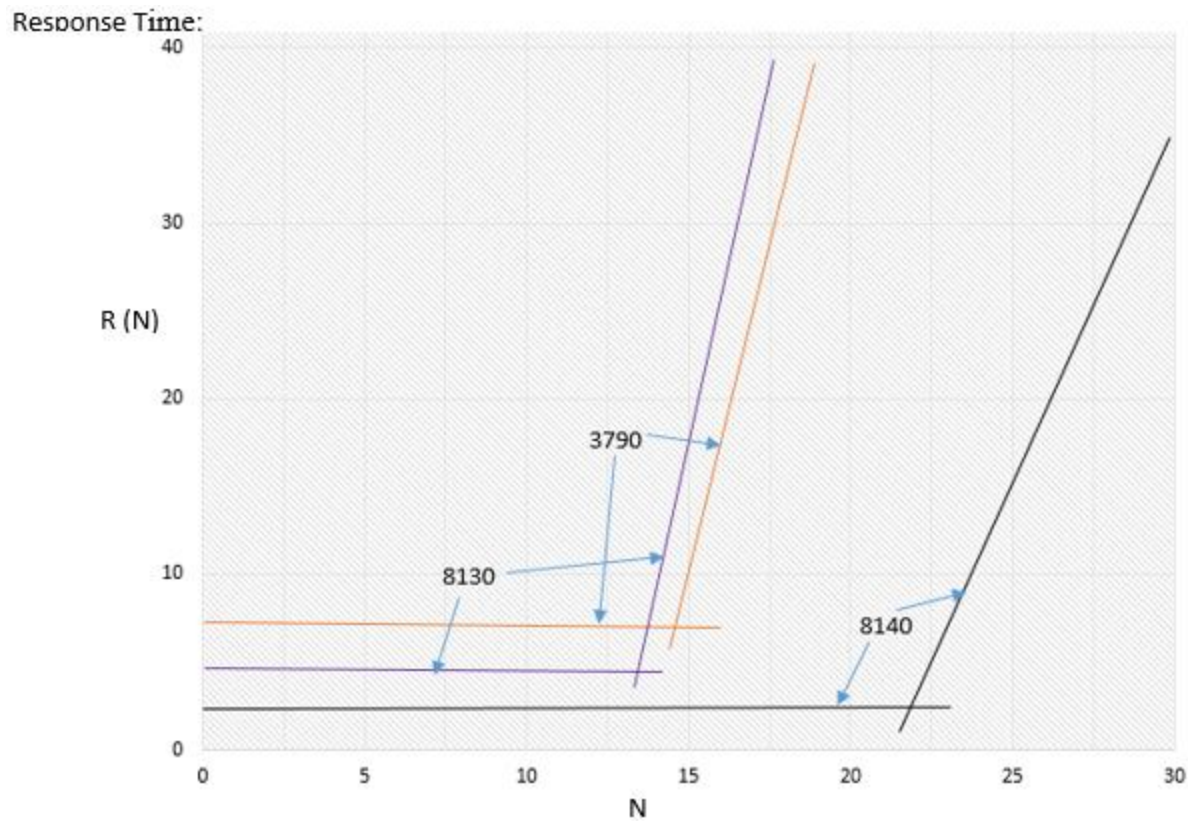
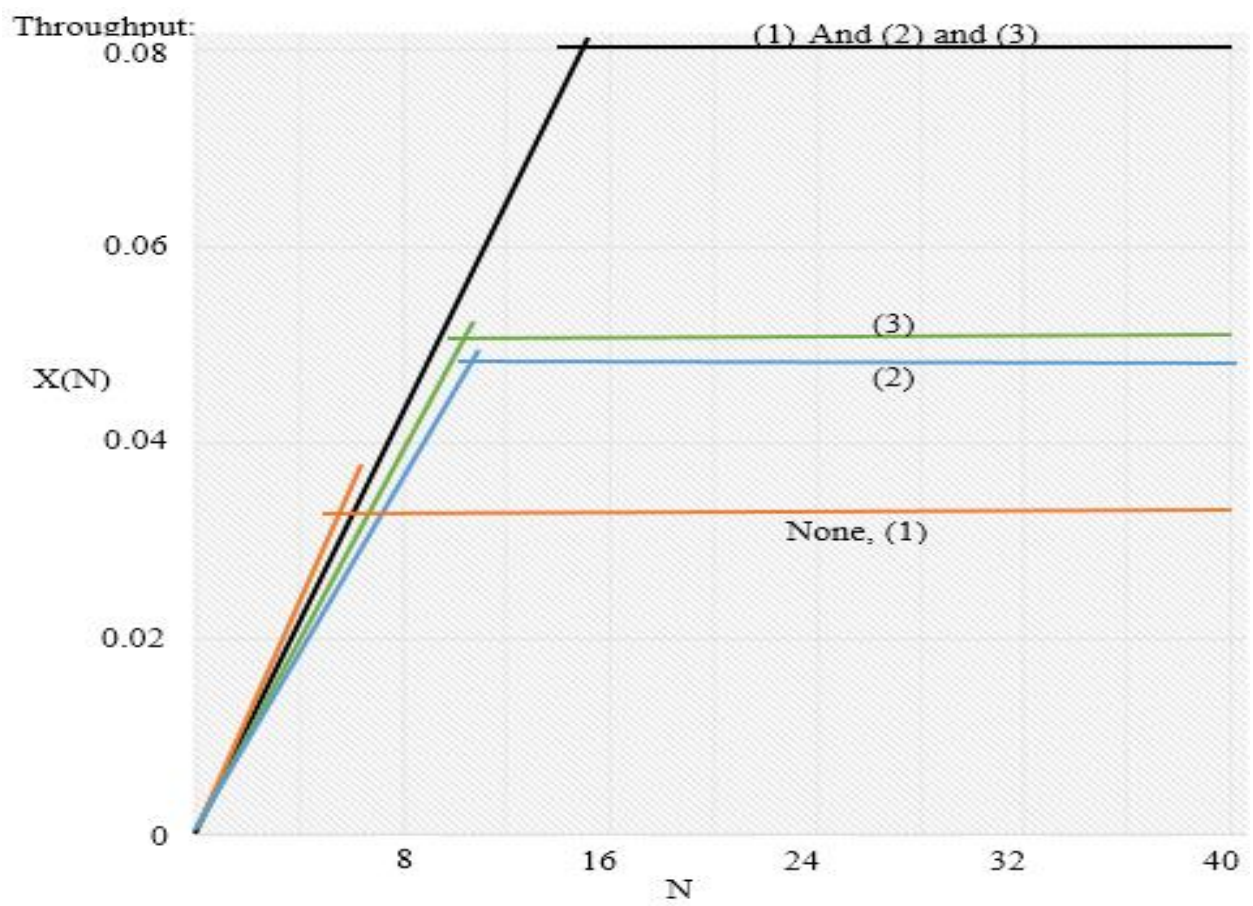


Fig. 4.1b Asymptotic Bounds in the Case Study1 showing response Time

4.3 Results and Discussions of Case Study 2

The results of the case study 2 after the metric model has been applied to an existing system shows the optimistic asymptotic bounds for the original system which is indicated by "None" on the graph of Fig. 4.2, for each modification labelled "(1)" "(2)" "(3)" separately and with respect to the three together [as indicated by "(1)" and "(2)" and "(3)"]. One may think that the initial change might seem to be the most "significant" but Fig. 4.2 shows clearly that is not exactly the case. Since fast disk happens to be the initial bottleneck, changes "2" and changes "3" are significantly very influential. It is worthy to observe that, change "2" produced virtually great improvement as change "3" even though additional hardware is not required. All the three modifications put together yields appreciable results.



Response Time:

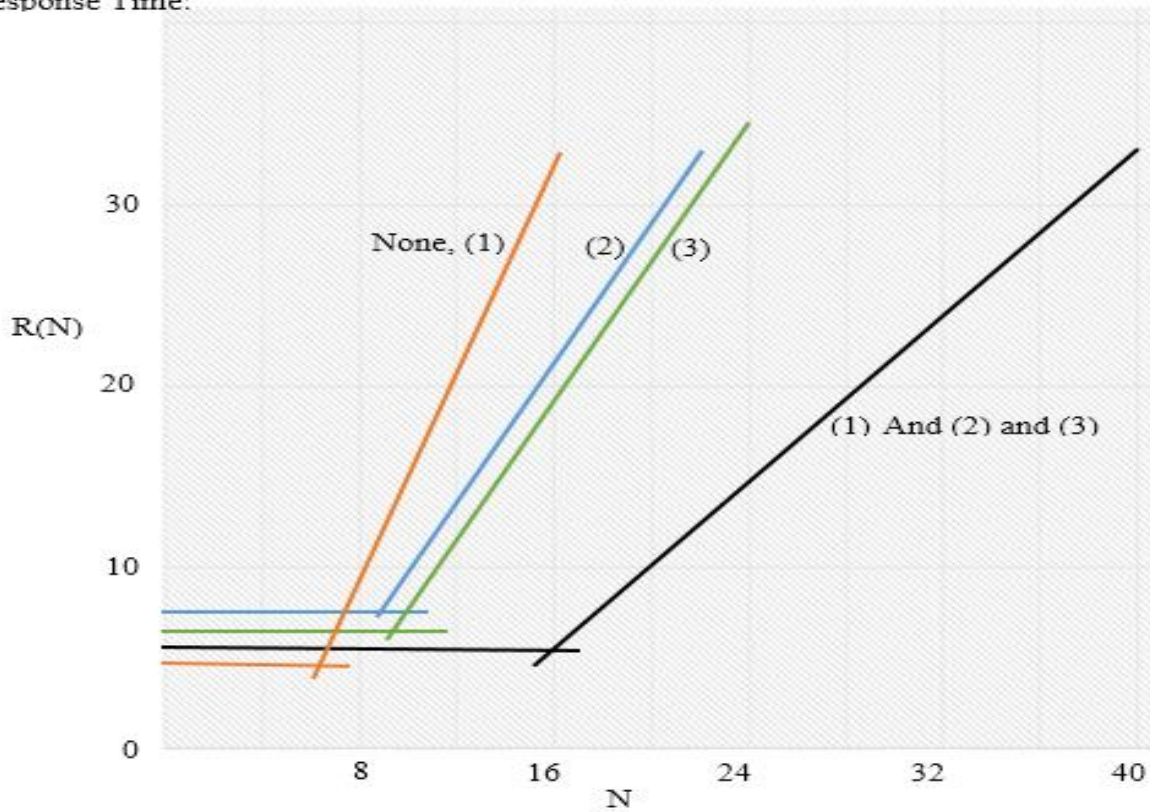


Fig. 4.2 Graph of the effect of various changes

4.3 Results and Discussions of the Connection Management System

From the results summarized in Table 4.1, the full calculation optimizes the productivity function with respect to the number of clients which is the scalability enabler at each scale factor. The results in the table shows that the scaling strategy and optimization gives response times which are well within the target at all scales, except that the scalability is only moderate. The results also shows that, the throughput increases from $k = 1$ to $k = 2$ and then levels off whereas costs rises, which drives the scalability down. More also, the database CPU columns shows that most of the database work is overhead and at the larger scales. The graphs of Fig.4.3a and 4.3b shows the plots of detailed scalability measure and the bound. The results

shows that, the system is spinning its wheels and thereby generating overhead but not performance.

Table 4.1 Scalability Metric Results for the Connection Management System

(The Normalized Response Time is the mean response time divided by the target of 15 min.)							
Scale Factor	Productivity (Optimized) (sec^{-1} per unit cost) x 1e-2	Scalability metric value (Optimized)	Throughput (operations per hour)	Normalized Response Time	Database CPU Utilization		System Cost (Units)
					Total	Due to transaction overheads	
1	1.95483	1.0	95	0.3017	92.59	*	1.05
2	2.02033	1.0335	108.62	0.3645	86.86	67.85	1.1
3	1.90128	0.9726	111.18	0.4126	82.43	69.45	1.15
4	1.75660	0.8986	112.01	0.4761	79.77	69.97	1.2
5	1.61547	0.8264	112.26	0.5449	77.98	70.12	1.25
6	1.48613	0.7602	110.95	0.5953	75.77	69.30	1.3
7	1.36949	0.7006	110.94	0.6675	74.84	69.30	1.35

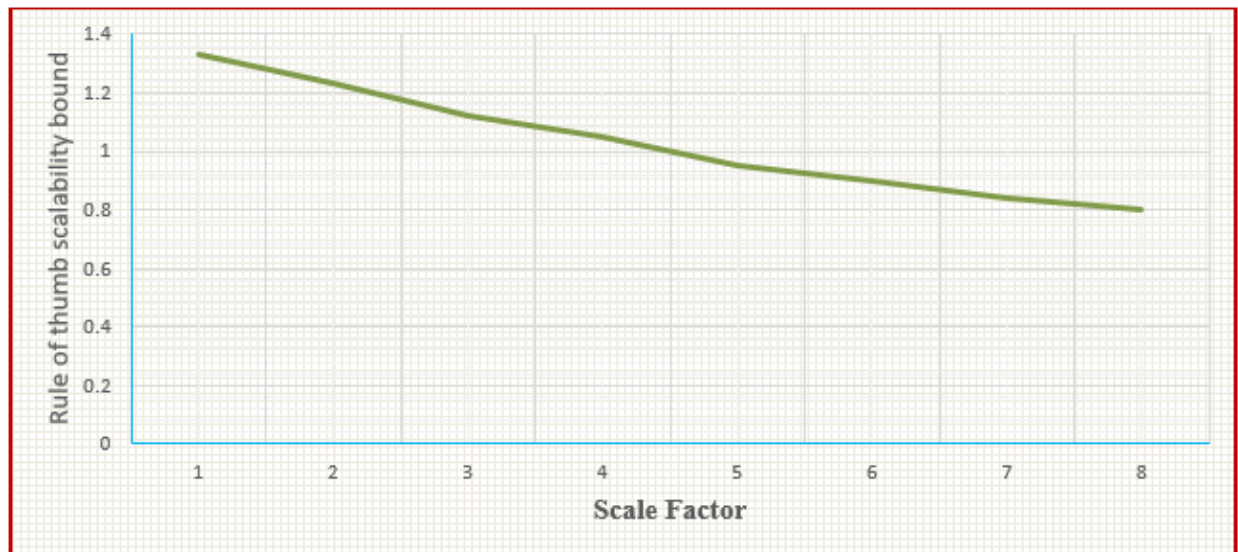


Fig. 4.3a Scalability bound for the connection management system

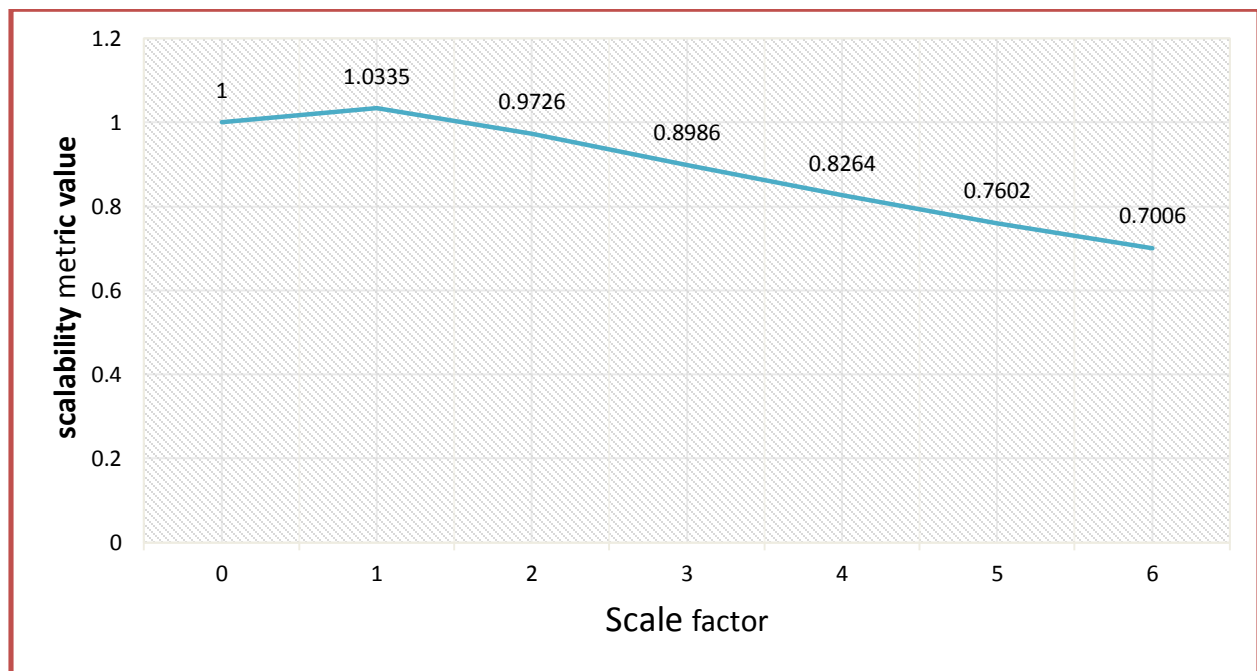


Fig. 4.3b Scalability metric for the connection management system

4.4 Results and Discussions of the Call Processing System

The results of the call processing system as scaled by the metric and same tabulated shows that, the scalability metric values plotted in Fig. 4.4a drops to 0.8 at the scale factor which is about $k = 4$. This indicate that the system is scalable up to scale factor of 4. The graph of throughput Verses scale factor in Fig. 4.4b shows the throughput with a knee at about $k = 3$ and at this point it has been increased from about 1.08 million calls per hour to about 3.4 million per hour whiles at the same time maintaining good quality of service (QoS). For scale factors (k) beyond 3, the optimization gives a response time which is a little higher than the target value. This is encouraged by the metric, because it also gives a higher throughput. The results of the table and that of the graph plotted in Fig. 4.4a also shows that the available productivity of the system drops gradually up to $k = 3$ and from there very sharply. In summary, the results indicate that the scalability is reasonable up to the scale factor of 3 and beyond that, the

scalability metric degrades, even though the capacity continues to increase up to about the scale factor of 10. Beyond this factor, the system is bottlenecked and at the same time capacity is saturated. Fig. 4.4c also shows the graph of scalability against scale factor.

Table 4.2 Scalability Metric Results for the Call Processing System

(*The response time is normalized to the target mean response time of 10min.)							
Scale factor	Optical No. of replica		Productivity (Optimized)	Scalability metric value (Optimized)	Throughput (Calls per hour) x 10 ⁶	Normalized* Response Time	System Cost (units)
	Database	Location server	(ms ⁻¹ per unit cost)				
1	1	1	0.1600	1	1.0958	0.72	1.1
2	1	1	0.1492	0.9325	2.1273	0.88	2.1
3	1	6	0.1444	0.9025	3.0531	0.89	3.1
4	1	3	0.1178	0.7363	3.3848	0.92	4.3
5	4	2	0.0866	0.5413	3.6485	1.16	5.4
10	2	4	0.0589	0.3681	4.4934	1.11	10.2
15	2	10	0.0387	0.2419	4.4934	1.12	15.2

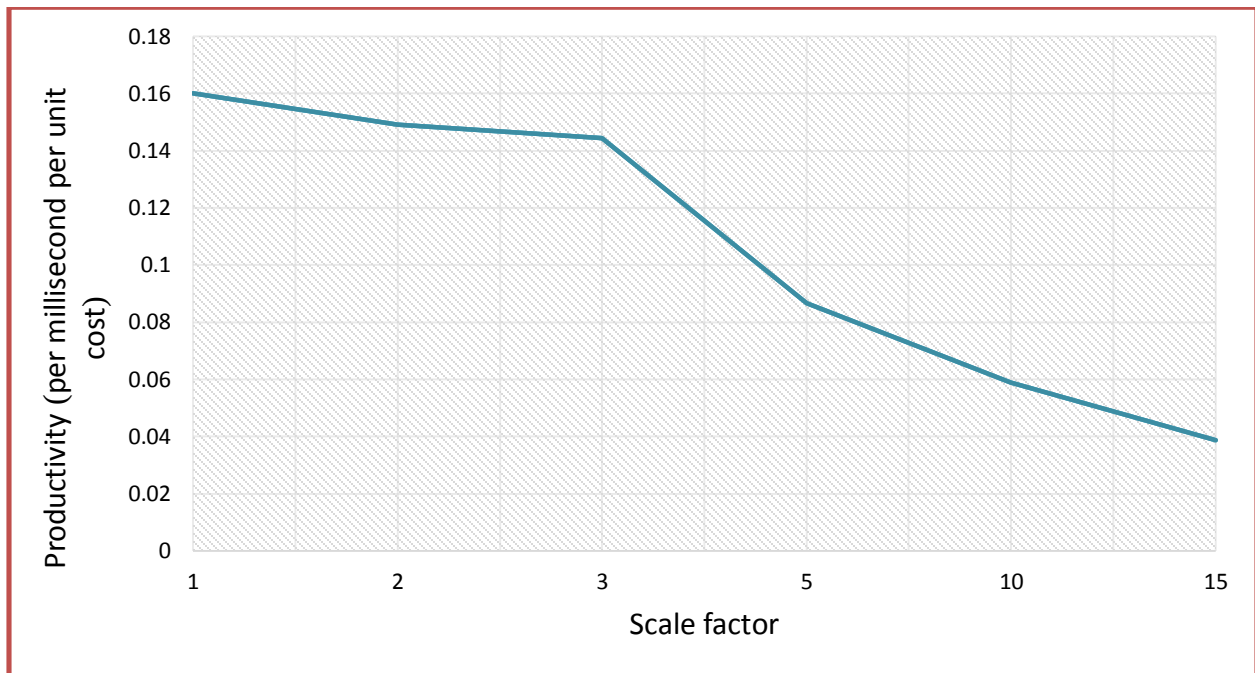


Fig. 4.4a A Graph of Productivity vrs Scale factor

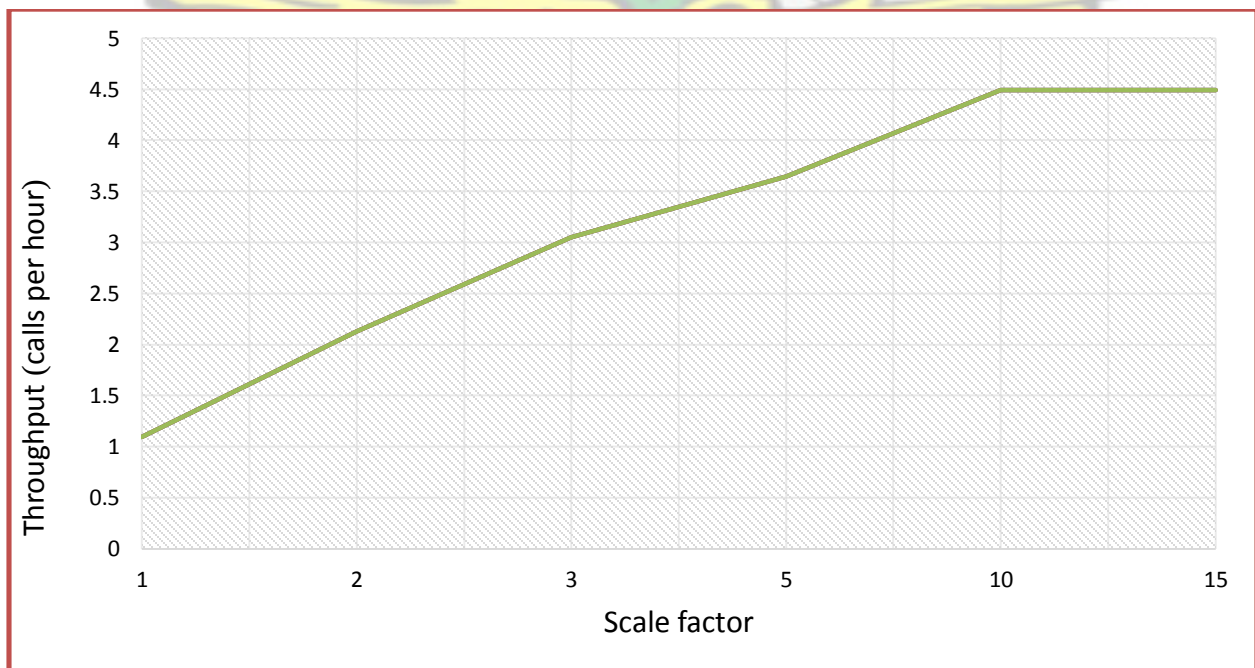


Fig. 4.4b A Graph of throughput vrs Scale factor

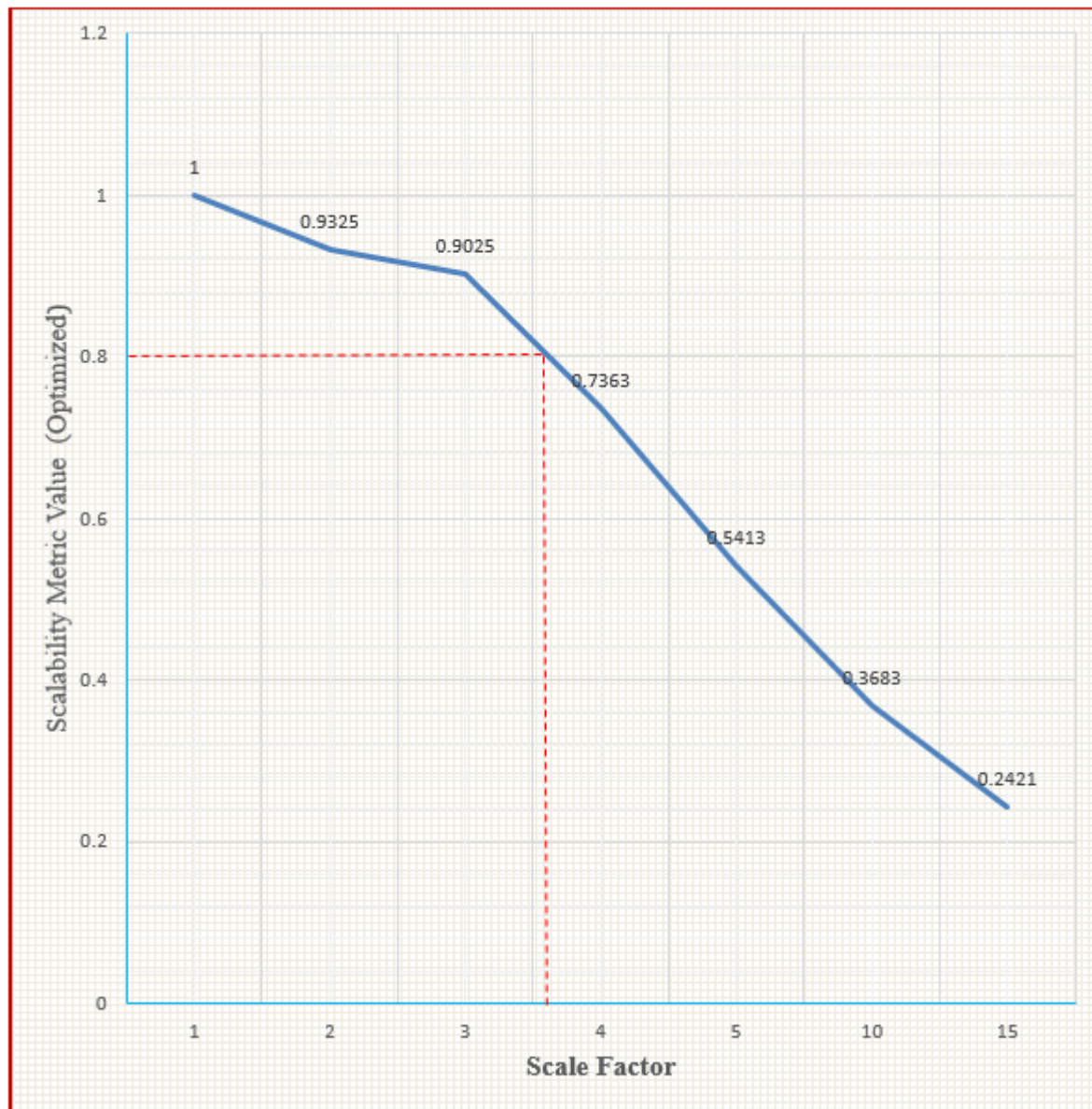


Fig. 4.4c A Graph of Scalability vrs Scale factor

CHAPTER FIVE

CONCLUSION AND RECOMMENDATIONS

5.1 Conclusion

Asymptotic bounds and balanced system bounds becomes easier to calculate once you know D and D_{max} since it requires simple arithmetic operations. Because of this, they are a quick way to obtain a fair insight of the behavior of a system. They shows the critical influence of the bottleneck service center and hence, changes to the system that do not affect the bottleneck center do not alter the heavy load bounds on performance. Therefore, throughput curves for all systems with bottleneck demand D_{max} are constrained to lie below the line $1/D_{max}$. If one needs to improve the performance further than this limit, then it is necessary to reduce the demand at the bottleneck center somehow. The metric derived in this thesis is a strategy based scalability metric that generalizes the well-known metrics for scalability of parallel computations to describe heterogeneous distributed systems. It is worthy to note that, in these systems, a uniform increase in all components types is ideally not a reasonable scaling strategy. What makes this derived metric unique is the fact that, it separates the impact of throughput and response time on the metric, formalizing the notation of a scaling strategy, introducing QoS evaluation and more also, introducing formal scalability enablers which are optimized at each scale factor. The derived metric is the ratio of the system's productivity in a scaled version to the productivity of a base case. It also gives a reasonable results for a large collection of idealized and well understood systems nodes which is in the form of queueing modes suitable for distributed systems unlike the other metric like fixed size speedup, where the scaling strategy is to use k processors, throughput is the inverse of completion time, cost is k and quality of service function is given as $F = 1$. More also, the scalability that deals with

fixed time speedup also deals with the use of k processors, but it changes the workload (W) to a value which keeps the completion time constant. In that strategy, throughput is constant, cost is k and quality of service function is $F = W$.

5.2 Recommendations

With respect to case study 2, the conclusions reached would not entirely hold in a context of different workload. In the case of call processing system, it can be concluded that the call processing system is scalable up to the scale factor of 4 and at that level, it is capable of supporting roughly 3.3 million calls per hour. If it becomes necessary to scale further than this, then recommendations would be as follows:

1. The software objects organization into concurrent tasks ought to be redesigned in order to make their execution and communication demands more equal.
2. Modification of the database schema can be tried in order to partition the database in various domains instead of replicating it and also, the consistency management overheads can be reduced.

REFERENCES

Agarwal A. Limits on Interconnection Network Performance. *IEEE Transactions on Parallel and Distributed Systems*, 2(4):398–412, October 1991.

Amdahl G. M. . Validity of the Single Processor Approach to achieving Large Scale Computing Capabilities. In *Proceedings of the AFIPS Spring Joint Computer Conference*, pages 483–485, April 1967

Anderson T. E.. The Performance of Spin Lock Alternatives for Shared-Memory Multiprocessors. *IEEE Transactions on Parallel and Distributed Systems*, 1(1):6–16, January 1990.

Andrew Tannenbaum, Maarten Van Steen "Distributed systems: principles and paradigms". Second Edition, 26th February, 2016.

Burkhart H. and Millen R. Performance measurement tools in a multiprocessor environment. *IEEE Transactions on Computer Systems*, 38(5):725–737, May 1989.

Chen D, Su H, and Yew P. The Impact of Synchronization and Granularity on Parallel Systems. In *Proceedings of the 17th Annual International Symposium on Computer Architecture*, pages 239–248, 1990.

Crovella M. E. and LeBlanc T. J. Parallel Performance Prediction Using Lost Cycles Analysis. In *Proceedings of Supercomputing '94*, November 1994.

Cypher R., Ho A., Konstantinidou S., and Messina P. Architectural requirements of parallel scientific applications with explicit communication. In *Proceedings of the 20th Annual International Symposium on Computer Architecture*, pages 2–13, May 1993

Eggers S. J. and Katz R. H.. The Effect of Sharing on the Cache and Bus Performance of Parallel Programs. In *Proceedings of the Third International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 257–270, Boston, Massachusetts, April 1989.

Fortune S. and Wyllie J. Parallelism in random access machines. In *Proceedings of the 10th Annual Symposium on Theory of Computing*, pages 114–118, 1978.

Gustafson J. L., Montry G. R., and Benner R. E. Development of Parallel Methods for a 1024-node Hypercube. *SIAM Journal on Scientific and Statistical Computing*, 9(4):609–638, 1988

Helmbold D. P. and McDowell C. E. Modeling Speedup (n) Greater than n . *IEEE Transactions on Parallel and Distributed Systems*, 1(2):250–256, April 1990.

Jamieson L. H, Gannon D. B., and Douglas R. J., editors, *The Characteristics of Parallel Algorithms*, pages 65–100. MIT Press, 1987.

Jogalekar P.P. and Woodside C.M., "A Scalability Metric for Distributed Computing Applications in Telecommunications," *Proc. 15th Int'l Teletraffic Congress Teletraffic Contributions to the Information Age*, pp. 101-110, 1997.

Karp A. H. and Flatt H. P. Measuring Parallel processor Performance. *Communications of the ACM*, 33(5):539–543, May 1990.

Kumar V. and Rao V. N. Parallel Depth-First Search. *International Journal of Parallel Programming*, 16(6):501–519, 1987

Kung H. T. The Structure of Parallel Algorithms. *Advances in Computers*, 19:65–112, 1980. Edited by Marshall C. Yovits and Published by Academic Press, New York.

McDonald J.C, Fundamentals of Digital Switching. New York: Plenum Press, chapter 4, 1990.

Mellor-Crummey J. M. and Scott M. L. Algorithms for Scalable Synchronization on Shared-Memory Multiprocessors. *ACM Transactions on Computer Systems*, 9(1):21–65, February 1991

Nussbaum D. and Agarwal A. Scalability of Parallel Machines. *Communications of the ACM*, 34(3):57–61, March 1991.

Pan A. M, "Solving Stochastic Rendezvous Networks of Large Client-Server Systems with Symmetric Replication, " master's thesis, Dept. of Systems and Computer Eng., Carleton Univ, Ottawa, Sept. 1996

Pfister G. F. and Norton V. A. Hot Spot Contention and Combining in Multistage Interconnection Networks. *IEEE Transactions on Computer Systems*, C-34(10):943–948, October 1985.

Rothberg E, J. Singh P, and Gupta A. Working sets, cache sizes and node granularity issues for large-scale multiprocessors. In *Proceedings of the 20th Annual International Symposium on Computer Architecture*, pages 14–25, May 1993. "Generic Functional Architectures for Transport Networks," Int'l Telecommunications Union Recommendation no. G. 805, Nov.1995

Sivasubramaniam A, Singla A, Ramachandran U, and Venkateswaran H. An Approach to Scalability Study of Shared Memory Parallel Systems. In *Proceedings of the ACM SIGMETRICS 1994 Conference on Measurement and Modeling of Computer Systems*, pages 171–180, May 1994.

Sivasubramaniam A, Singla A., Ramachandran U, and Venkateswaran H. A Simulation-based Scalability Study of Parallel Systems. *Journal of Parallel and Distributed Computing*, 1994. To appear.

Sun X-H. and Gustafson J. L. Towards a better Parallel Performance Metric. *Parallel Computing*, 17:1093–1109,1991

Trantafiliou P. and Taylor D. J, "The Location-Based Paradigm for Replication: Achieving Efficiency and Availability in Distributed Systems, " *IEEE Trans. Software Eng.*, vol. 21, pp. 1-18, Jan. 1995..

Von Eicken T, Basu A., Buch V., and Vogels W., "U-Net: A UserLevel NetworkInterface for Parallel and Distributed Computing, "Proc. 15th ACM Symp. Operating Systems Principles, pp. 1-14, Dec. 1995.

