NTNU
Innovation and Creativity

Idol Show in China

Gerard Draper Gil

Master of Science in Communication Technology
Submission date: August 2006
Supervisor: Poul Einar Heegaard, ITEM
Co-supervisor: Tormod Omholt-Jensen, Boost Communications

Norwegian University of Science and Technology
Department of Telematics

Problem Description

The use of mobile phones for interactivity is growing rapidly. In Norway, we are increasingly given the ability to use SMS for voting in TV-shows etc. As the popularity of such interactive events increases and expands to new markets, there is huge need for highly scalable SMS processing solutions.
Boost Communications AS is developing and selling
interactive SMS solutions. We are about to expand into new markets and need to find server/ network topologies suitable for high load SMS processing.
Assuming for example that Boost Communications could be chosen as the provider of the SMS voting system for the Chinese «Idol»-show, we would need an SMS processing system capable of handling in the order of one billion users.
A scalable SMS processing system will typically consist of a set of processing servers, a set of database servers, switches and load balancer(s). To receive and send SMS messages, the system will be connected to one or more operators.
The student should investigate existing highly scalable processing solutions. In addition Boost will present a couple of proposed network/server topologies. The student should perform scalability analysis and simulations to detect possible bottlenecks in the proposals. Network traffic and access to shared resources such as databases, will be possible causes of bottlenecks.
Based on the analysis, simulations and research, the student should propose ways to improve the scalability and hence the performance, of the system. Improvements can include both topology changes and change of technology (for instance change of communication protocols used between processing and database servers).

Assignment given: 20. February 2006
Supervisor: Poul Einar Heegaard, ITEM

Contents

List of Figures

Preface

The following project has been developed during my Erasmus in Norway, at NTNU. The project has been supervised by Poul Heegard, from NTNTU, and Tormod Omhold-Jensen from Boost Communications. I would like to thank both of them for the time they have dedicated to this project and to me, to answer my questions and doubts.

Objectives

Our objective is to test the performance of a real SMS Processing System. We will work with Boost Communications, a norwegian company that is developing a SMS Processing System to sell interactive SMS solutions. With our test, we want to be able to put the system performance into numbers, and to identify and eliminate possible bottlenecks.

In order to achieve our objective we will use different techniques. First, we wil build an analytic model, a simple one, an we will use It to try to identtify the main weak points on the Architechture. Once built, we will run a few tests on the Architecture to identify our modeled parameters values. The second step will be to build a simulation model using the knowledge we will had achieved while doing the analityc model and the first tests.

At the end, we will have to be able to point the bottlenecks and offer solutions to them. These solutions can be hardware based, or software based.

With this work, we will try to approach the educational world and the labor world, that sometimes seem to be so far away one from the other. On the labor world, due to timing or economic issues, the solution adopted to fix a problem is not the one we will cal ”optimal” on the educational world.

Part I

Introduction

Chapter 1

Introduction

The mobile phones penetration level is increasing year by year. In some countries this level has even raised above the 100, as we see in Figure 1.1(Source: OECD ICT Key Indicators [www.oecd.org/sti/ICTindicators]). As this level increases, the number of services offered by the mobiles are also growing: MMS, Videoconference, E-mail, Internet, SMS.... In particular, the use of the SMS to provide a wide different variety of services has increased quickly.

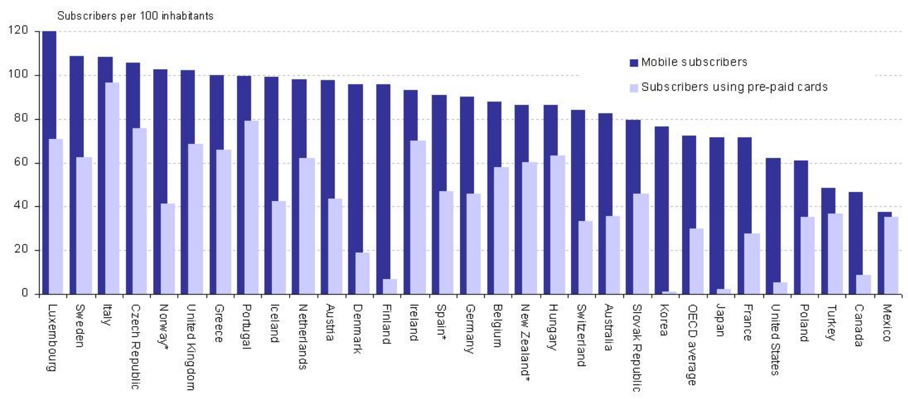


Figure 1.1: Mobile Phone penetration level

1.1 Televoting

One of the most popular SMS services is the ”televoting”. This service creates an interaction, a feedback, between the mobile phone user and the client that hire the service (TV-Shows, radio, magazines,. . .). Its most popular application, perhaps, It is the use that TV Shows gave to It. Besides the many ways in that can be applied, the foremost important thing is that all the audience became a potencial user of this service. With this huge potencial market, the ”televoting” has became an important source of incomes.

This service generates bursts of requests, mostly concentrated during de show duration. So a show like ”American Idol” that can have an audience of 35.5 million viewers, is potentially able to generate an incoming traffic of 35.5 million messages in 2 hours!, or about 10.000 messages per second, during a short period of time.

But ”televoting” is not the only service offered by SMS, there are many others, some related and some unrelated to the TV-shows. Services like ringtones downloading, news subscriptions, etc . . . Services that have a lower incoming traffic needs.

”televoting” systems must process large amount of messages in a short period, meanwhile other services needs a lower traffic. So we need a flexible architechture for our SMS services platform.

1.2 This Project

Let us assume that China (population aprox. 1.300 Millions), achieves a penetration level close to 100%. If a national television plans to run a reality show, like Idol Show, and they use a SMS ”televoting” System , this System will have to support a large number of incoming messages. So our System must be prepared. We have to be able to scale our System to feed the needs of our customer.

The starting point for this project was the assumption that Boost Communications could be chosen to process the SMS messages generated by a reality show, like Idol Show, in China. So if we assume that China reach a penetration level close to 100%, with a population of 1.300 Millions aprox.,we can expect that our SMS processing system will have to process a huge amount of messages. In order to be able to satisfy the client needs, Boost SMS processing System must be able to increase its performance. In this project we pretend to push the Boost system until we find its limit, and why its limit happends.

1.2.1 General View

However, this is a problem that affects not only to the SMS applications. Any on-line service must be ready to increase its performance very fast because they have a potenciall market of millions of users . Every day new services appears and every day some of them disappear. The difference between a succesful or a failure can be the lack of users or the excess of them, the ”slashdot effect”. If we have too many request and our System is not able to deal with them, our service can go off-line or delay the users. This lack of service can make the users to choose another service provider.

Chapter 2

Previous Work

2.1 Computer Based Models

The scalability of a system is not an exclusive problem of Boost Communications. Predict how our system will respond to a traffic increase or hardware changes, will help Boost into cover their clients needs. A good way to do It is through simulations. If we can build a computer based simulation of Boost Communications Architechture, we will be able to predict its behaviour. The problem is to find a valid simulation model. The concern about if the simulatino model is correct or not is resolved through the validation and verification of the model. The verification concerns to the implementation of the model. Meanwhile, the validation is concerned to the bulding of the right model.

- To verificate the model we have to follow the same steps as when we build a software application.
- Many documents have been written on how to validate a simulation [Sar99]. It is not an easy work, moreover, we can find papers where the autor accepts the fact that the simulation may never be 100%, that It is enough with a certain level fo confidence.

2.2 Scalable Systems

An intuitive definition of what is scalability is given by the Free On-Line dictionary of Computing [FOL06]: "How well a solution to some problem will work when the size of the problem increases". It is good enough to understand the concept, but when we want to apply It, we find this definition too subjective, incomplete. Many authors have tried to give a formal definition of scalability or define and measure It [Luk94] and Its properties [Bon00]. In some cases the author even discuss the necessity of a system being scalable [Hil90].

We will asume that the scalability is a desirable and a necessary propertie of our system, moreover, that in fact our system is scalable. As a scalable we will mean that we can increase the maximum incoming traffic λ that our system is able to process, by making hardware (adding processors, HD, ...) or software (change or tuning database, number of Threads, ...) changes at a reasonable cost.

The problem is that the ”reasonable cost” is a subjective value. From the academic point of view we can think that our system is scalable (as easy as duplicate the system!), but from Boost point of view It is possible that the system seems not scalable.

Chapter 3

SMS Processing Solutions

Like an E-mail Service or a Web Service, a SMS Service has Its own particularities. In this chapter we will try to expose them and explain how they affect to our System. After this, we will give a description of the solution that Boost has implemented.

3.1 SMS Services

In a mobile communications system we can distinguish between acces providers and content providers.The acces providers would be, for example: Orange, Telenor, Deutsche Telecom, . . . and the content providers would be the ones that offer services as ringtone downloads, televoting, ... So every time you send a SMS you have to pay to the acces and to the content provider independently. So if your SMS content provider is off-line, by the time you send the message, this message will be lost, but you will have to pay to the network acces provider.

This fact has recently caused a polemic in Spain, where the reality show called "Operacion Triunfo" experimented technical problems during the las program of the last season. In that show, the program received about 1.5 million messages in 2 hours. The complain comes from the fact that a lot of people could not vote for his candidate because of "network problems", but the sms messages were charged.

3.2 Boost Network Architecture

A scheme of the Boost Network Architecture is shown on Figure 3.1. We will build an analytic model of It, based on this scheme. The Boost Network Architechture has 5 main modules: Receiver, Dispatcher, Product, Message Sender and Message Queue.

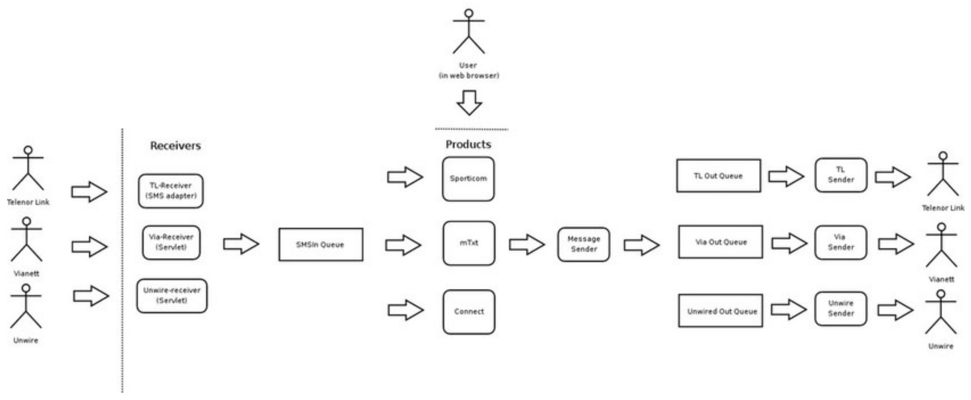


Figure 3.1: Boost SMS Processing Solution

3.2.1 Modules

Receiver

The Receiver is the entrance door to the system. It is connected to the aggregator, the link between the mobile operators and the Boost network. The aggregator sends the request to the Receiver using the http protocol. The incoming messages are stored in a database using the JMS $JavaMessageService$. The JMS server allow the user to choose how to implement the queue, the options are using memory, using a file or using a database. Boost has decide to implement the queue as a database to avoid the losing of received messages that have not been proceesed yet in case of a system failure. We have been told that the receiver and the aggregator have some kind of feedback, but not specified how, to avoid loosing messages. It happends that if the receiver is not able to process more messages, the aggregator will lower the incoming traffic level (Figure 3.2).

Dispatcher

The Dispatcher is a Message-Driven Bean. It reads the messages from the incoming queue and send them to the Product module. It works in two different ways: as a load balancer and as a distribution module.Once a message has been processed, It deletes It. At the scheme shonw on Figure 3.1 we see 3 product modules. This 3 product modules can be the same one so the dispatcher would work as a load balancer, 3 different ones so the dispatcher would work as a distribution system, or a mixture so the dispatcher would distribute the different messages into the different products and would balance the incoming traffic between the redundant modules (Figure 3.2).
.

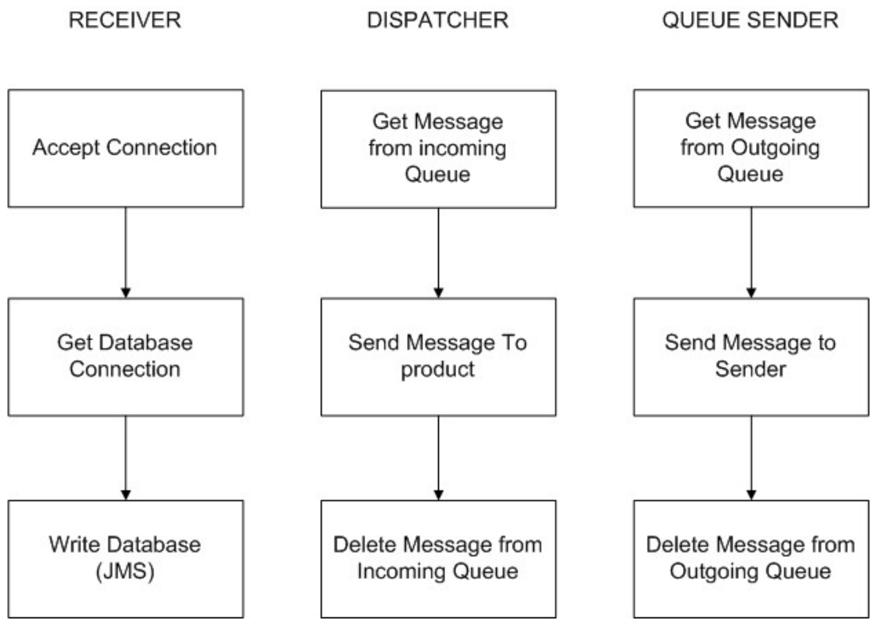


Figure 3.2: Modules 1

Product

The Product module is an EJB. There is one for every different product: televoting, massive sms sending, news, etc . . . During the chain of events triggered by a message: *recepcion* $\longrightarrow$ *distribution* $\longrightarrow$ *process* $\longrightarrow$ *response* It will be the main Service Time consumer in average (from different products). A tipical Product process It is show on figure fig:Diagrama2 . The "Find Price", "Create Account Transaction" and "Create and Send Response" will be the main Service Time consumers within the Product module. The two first ones involve many DataBase transacions: checking client accounts, updating payments, . . . and the third one involves Processor time to create the message or messages to send, and the writing of It to the outgoing queue using JMS, as the Receiver module does

Message Sender

The Message Sender acts the same way than the Dispatcher. It reads the messages from the outgoing queue using JMS, and send them to the proper Message Queue module. The Queue Module is chosen depending on the cost fo sending the message. Once the message has been sent It deletes It from the outgoing queue (Figure 3.2). .

Message Queue

The Message Queue is the exit door from the System. It is connected to the aggregator. We have different Message Queue modules, so we can choose the cheapest way to send a message depending on its destination (we use different network operators)(Figure 3.3).

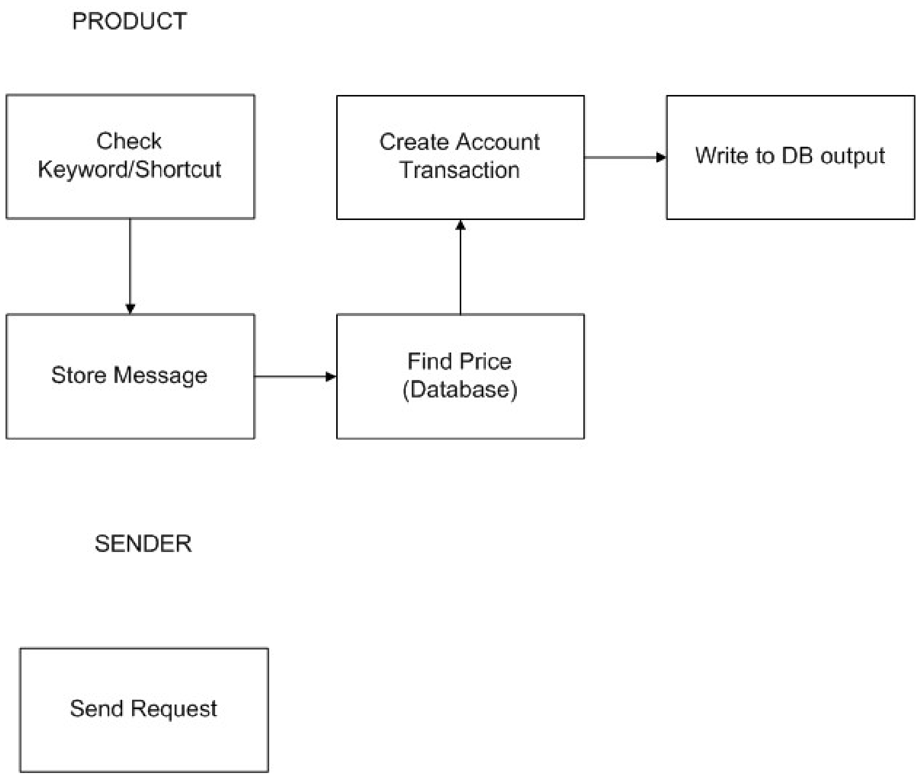


Figure 3.3: Modules 2

3.2.2 Database

We have 3 different databases: One to store the incoming messages, one to store the outgoing databases and another one used by the product module, containing all the information about customers and services. This 3 databases can run on one Database or can be separated, since they do not have any link between them.

Incoming Messages Database

As we explained earlier, the Receiver and the Dispatcher use the JMS to communicate each other. The JMS use this database to store the incoming messages.

Outgoing Messages Database

The product uses the JMS to send the messages to the Message Sender. As It happends with the incoming queue, the JMS use a database to store the messages, this is the Outgoing Messages Database.

Product Database

This is the main database. All the information related to customers, products, etc . . . is stored here. The product module use this Database to process the messages. There are different products. Each one has to perform different actions to process one message. All the information related to this process, like whatever is stored in this database. It is the largest one

3.2.3 First Analysis

After learning how the Boost System is implemented, we can ask ourselves few questions and we can give an early report on the main weak points or possible bottlenecks.

JMS-Database

Is all this writing/deleting messages into database (incoming/outgoing) necessary? If the answer is yes, we can expect that the database acces will be one bottleneck. If it is no, we will have to find an alternative to It. In case we find that this system is absolutely necessary we will have to find the way to improve the performance of this modules. We will have to find a way to improve the write-delete of a database. This operations will involve the Hard Drive configuration: single or RAID system, SCSI?,Serial ATA?, even the brand can make a difference. The Database itself (Boost is using PosgreSQL) will be also a main issue, It must be configured properly (this can be very difficult) and different Database software will have different performance.
Finding an alternative to the JMS-Database may not be that easy. First thought will be eliminate the database as the way to store the messages by the JMS. This way we will move the bottleneck from a write-delete database to a write-delete Memory (much faster).
The write-delete from database gives to Boost a very robust system agains failures, avoiding to loose messages: the messges written in the database will not be lost if the system is restarted or shut down, meanwhile if we store them into the RAM memory they will be erased. But we should think how often does a main failure happens? can we recall messages from the

aggregator? It also gives to the system a nearly unlimited queue size due to the Hard Drive size.

CPU consumption

The Product module is expected to be the one that cosumes more CPU time. We will use this assumption whe we build our analytic model. Despite the operations that realizes every module we also have the CPU consumed by the Database, and in some cases It can require a lot of resources.

Dispatcher-Product

The dispatcher threads are binded to the product ones. It is, the dispatcher reads a message, call the product, and when the product is finished It deletes It. The reason why It happens this way, as we were told by Boost, is that sometimes the sms process by the product may fail. In this case, the dispatcher will send the message to the product again. But, first question: How often does It happens? This relation provoques that dispatcher thread is locked doing nothing, consuming resources (memory,. . .)until the product have finished, very inefficient.

Part II

Analytic Model

Chapter 4

Queueing Theory

In this chapter we will have a brief description of the teorical knowledge necessary to develop the project. The descriptions will be accompanied by references with detailed descriptions.

4.1 Poisson Traffic Model

The first thing we have to decide is how to model the incoming traffic. We have chosen to use the Poisson Traffic Model due to Its analytical and simplification properties:

a. Process without memory. Interarrival time is an exponential distribution.

b. the average number of arrivals within T seconds $=\lambda \cdot T$

c. $\sum_{j=1}^{N}$ Poisson Processes with λ_i =Poisson Process with$\lambda = \sum_{j=1}^{N} \lambda i$

d. if we have a Poisson proces with

$$\lambda = P_1 \cdot \lambda_1 + P_2 \cdot \lambda_2 + \ldots + P_N \cdot \lambda_N$$

will be a Possin process

Also, It has been demostrated that the sequence of times at wich telephone calls are generated in the telephone network is a Poisson process.

4.2 M/M/1-PS

The Processor-Sharing model is a system where the server resources are shared by all the incoming requests. The incoming requests are served continuously, and each one get a proportional part of the resources:

$$\begin{array}{l} 1request \rightarrow \mu = \mu \\ 2request \rightarrow \mu = \frac{\mu}{2} + \frac{\mu}{2} = \mu \\ \vdots \\ Nrequest \rightarrow \mu = N \times \frac{\mu}{N} = \mu \end{array}$$

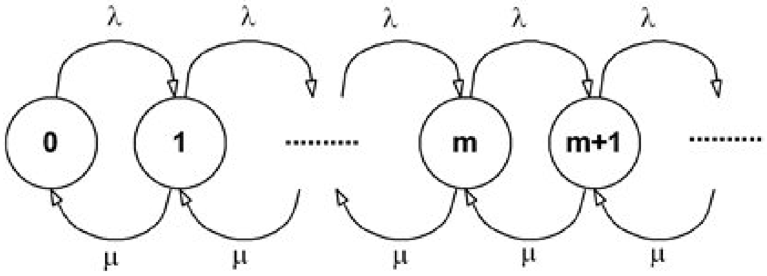


Figure 4.1: M/M/1-PS State Probabilities Diagram

On Figure 4.1 we can see the state probabilities diagram of the M/M/1-PS model.

$$\left\{ \begin{array}{l} P_0 = \left(1 - \frac{\lambda}{\mu}\right) \\ \\ P_k = P_0 \left(\frac{\lambda}{\mu}\right)^k \end{array} \right. \left| \begin{array}{l} \frac{\lambda}{\mu} = \leq 1 \end{array} \right.$$

4.3 Jackson's Network Theorem

In 1957 Jackson J. R. published a paper where he develops a theory known as Jackson's Network Theorem [Jac57]. It is a very powerful and easy to use tool for analyzing networks of queues. It asumes that we have a network of single server queues, where every node comply with the followiong statements:

a. Each node k has an average service time $\frac{1}{\mu_k}$,with Poisson departure process.

b. The messages arrives to the network according to a Poisson process with intensity λ_k

c. The messages processed at node j, will go to node k with probability P_{jk}, or will leave the network with probability:

$$1 - \sum_{k=1}^{N} P_{jk}$$

On his paper, J.R. Jackson demostrates that if we have a network of queues satisfying the previous statements, we can model our network using the following equations:

a. For each node k, the average arrival intensity Δ_k is calculated with the following equation:

$$\Delta_k = \lambda_k + \sum_{j=1}^{N} \Delta_j P_{jk}$$

b. As the state probability for each node is independent, the state probabilities are given in a product form:

$$P\left(i_1, i_2, ..., i_k\right) = \prod_{k=1}^{K} P_k\left(i_k\right)$$

Chapter 5

Boost Model's Architecture

5.1 Modeling Parameters

The first thing we need to do in order to build an analityc model of the Boost arquitecture is to identify and "put into numbers" the main variables. To build an effficient and easy model, we will have to make many assumptions and simplify some processes. The resultant model may differ from the real system behaviour, but It will give us an early vision of the system behaviour.

5.1.1 Incoming Traffic Model

Population

Our starting point to develop all this project was the "televoting" SMS application. Since this application has a potencial number of users about milions (see 1.1), we will assume that we have an infinite population.

Distribution Model

We will use a Poisson model as a traffic distribution model (see 4.1).

5.1.2 Service Time Distribution

We will assume that our Outgoing Traffic Model follows also a Poisson distribution model 4.1.

5.1.3 System Capacity

Since our System will not loose any message (there's some kind of feedback between the receiver and the aggregator), and our Queues (Incoming and

Outgoing message queues) are limited only by the Hard Drive size. We will assume that our system has an infinite capacity.

5.2 First Approach

With the description given by the Figure 3.1, our fist attempt to build a model of the Boost Network can be seen at Figure 5.1. We just made an analogy between modules and queue-servers.

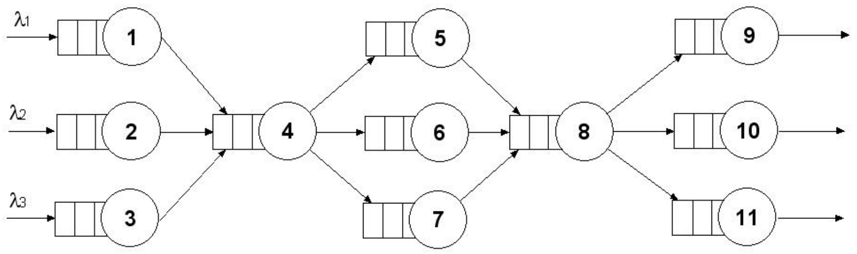


Figure 5.1: First Approach

This model can be simplified. First thing, we assume that the processing time from the product module will be much larger than the processing time from the dispatcher and the sender modules together. We also know that the dispatcher threads, are "binded" to the product threads. When the dispatcher gets a message from the incoming queue, It calls a product thread (depending on the message type) and waits for the process thread to finish. Once the product has finished It deletes the message from the queue.

5.3 A Simple Model

With the information given at the end of the previous section, we can make a simplification on the "Firts Approach" model to end with what It will be our model for the Boost System: Figure 5.2.

With this last simplification, our model will be like the Figure 5.2. The model also fits into the load balancing server distribution: All the incoming requests are sended to a common node. This node will distribute the traffic between the different servers.

Once we have a description on how our model will look like, we have to determine its equations. Using the theory from 4.2 and the Jackson's Network Theorem 4.3, we can easily have a mathematical description for our network.

A M/M/1-PS model will fit into our model: We have a markovian input and output and we will assume that we have a multiprocessor machine. This

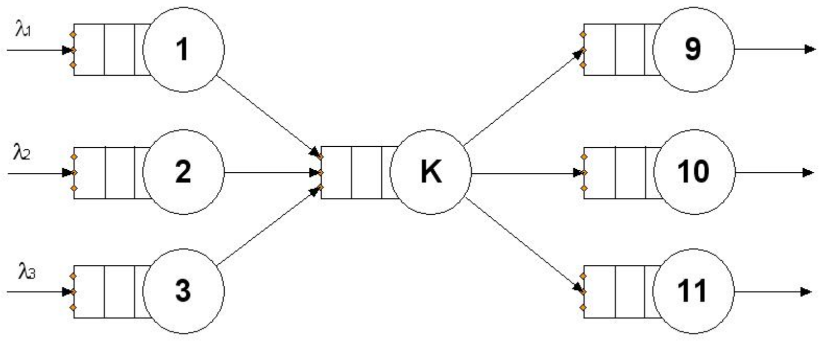


Figure 5.2: Boost Network Analityc Model

model will help us to comply with the Jackson's theorem statements since Its ouptut distribution is a Poisson

The Jackson's theorem Statemens:

a. First: Every node, as a M/M/1-PS model will have an average Service Time $\frac{1}{\mu_k}$, and the departure process will be Poisson, since the incoming is Poisson too.

b. Second: The incoming traffic model will be Poisson.

c. Third: The messages will go from node i to node j with a certain probability, P_ij.

Now we are able to write down the equations for the Boost Model:

a. For each node k, the average arrival intensity Δ_k is calculated with the following equation:

$$\Delta_k = \lambda_k + \sum_{j=1}^{N} \Delta_j P_{jk}$$

b. As the state probability for each node is independent, the state probabilities are given in a product form:

$$P\left(i_1, i_2, ..., i_k\right) = \prod_{k=1}^{K} P_k\left(i_k\right)$$

Our final model has 3 nodes. Each one can be represented by this equations:

$$\begin{cases} \Delta_k = \lambda_k + \sum_{j=1}^{N} \Delta_j P_{jk} \\ \\ P_k\left(m\right) = \left(1 - \frac{\Delta_k}{\mu_k}\right)\left(\frac{\Delta_k}{\mu_k}\right)^m ; \frac{\Delta_k}{\mu_k} \leq 1 \end{cases}$$

Where Δ_k = Average arrival intensity for each node, P_k = Delay probability for each node and m = number of threads running.

The value $\frac{\Delta_k}{\mu_k}$ is also known as ρ = Busy Probability. This value only makes sense if we have a stable system. That is, if $\rho \leq 1$ or $\lambda_{IN} = \lambda_{OUT}$.
Which is the desired value for this parameter? We would like to have a system that is 99,99% of the time busy to make our system as profitable as possible. The problem is that there is a relation between the busy probability and the delay probability: the more busy our system is, the worse delay probability we have, see Figure 5.3.
The number of threads will also influence the delay probability value: as much threads we have better Delay Probability we have. So, how many threads will we be running? on a real system our number of threads will be limited by the CPU power and the system memory. Sometimes adding a new thread will affect the system performance in a negative way. So, contrary to the graph, we will have to choose a maximum number of threads and this will limit our delay probability.

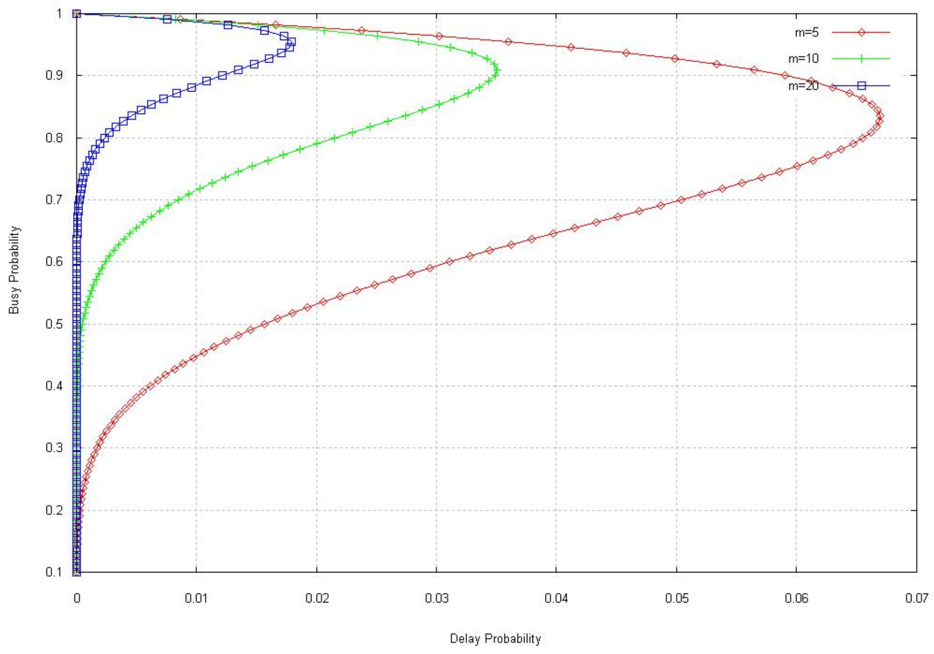


Figure 5.3: Busy Probability vs Delay Probability

We can also calculate the "Average time spent in the system" or system time by a message, that is, the time from a message enter the system until It leaves the system. We will use *Little's Law* [Kle57] to do It:

$$N = \lambda \cdot T \ \longrightarrow\ T = \frac{1/\mu}{1-\rho}$$

Where N = Average number of units in the system and T = Average time spent in the system

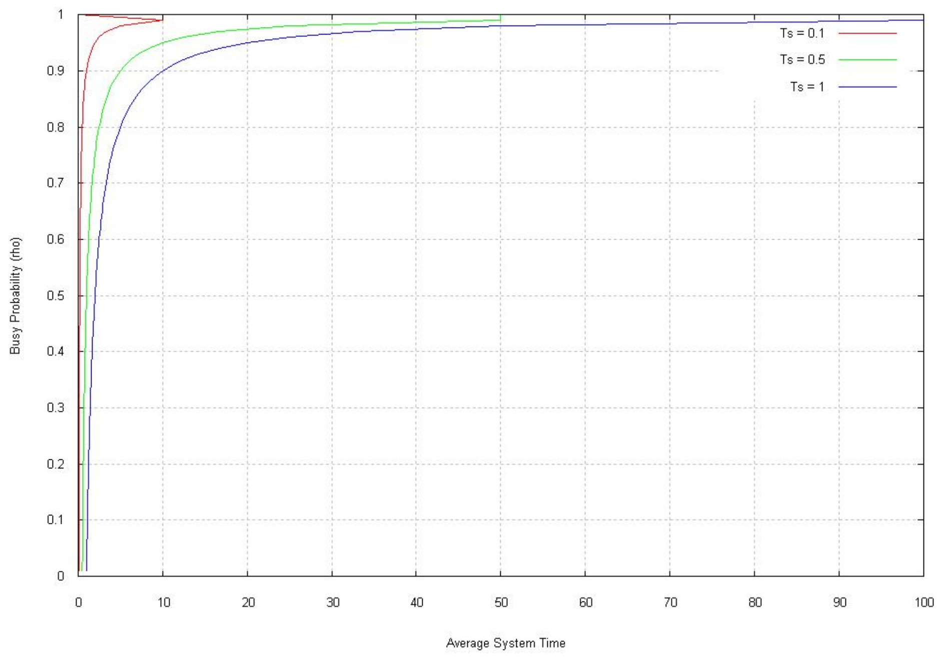


Figure 5.4: Average Time spent in the System

On Figure 5.4 we can see a graphic representing the Average Time spent in the system. Notice how this value increase exponentially at certain ρ value. Even when we have a Service Time of 0.5 ms, we can have that our message can last more than 10ms (20 times more) if our Busy probability is above 90%.

5.4 Test Results

To identify the parameters from the analytic model we ran a few tests on the Boost System. We used a single-processor computer with a software RAID 1 (mirroring) to run the modules and another computer to send messages using a sofware called *jmeter*.To do the tests we only used the dispatcher and the receiver modules, Boost was interested into knowing the capacity of receiving messages.

First unexpected results were related to the database. We noticed that the deletes and the selects took too many time to be executed. The cause was the lack of indexes in the database: design error. Also, another database problem was related to the Postgre SQL itself, the command *vacuum*. This command must be executed periodically to reset some database statistics that are used to decrease the seek time within the database. The problem is that either the command is executed too often or too late, the database performance decreases.
On the other hand, the behaviour related to the number of threads was expected. We made tests with different number of threads on the receiver and the dispatcher, to end with the conclusion that the best performance was obtained with 1 thread on the receiver and 3 threads on the dispatcher side. This result contradicts the analytic model, where increasing the number of threads has allways a positive effect.
On the following table, we have the results obtained during the tests:

RECEIVER			DISPATCHER		
Threads	Proc. Time	Std. Var.	Threads	Proc. Time	Std. Var.
1	6,3086	0,045244414	2	6,3074	0,044971086
1	6,0030	0,013067778	3	6,0020	0,013217778
1	6,0170	0,00249	4	6,0160	0,002448889
1	6,1320	0,012217778	5	6,1320	0,012217778
1	6,3990	0,076498889	10	6,3980	0,075551111
1	6,1120	0,014995556	2	6,1120	0,014995556
2	4,0540	0,176182222	1	6,0110	0,02001
3	3,4580	0,001373333	1	6,0740	0,003471111
4	3,0180	0,002884444	1	5,6160	0,007115556
5	3,8300	0,006911111	5	5,2880	0,007817778
4	3,9270	0,005823333	4	5,2880	0,00484
4	4,9000	0,003333333	8	5,2690	0,005121111
4	5,3187	0,020421344	10	5,3196	0,020334933
4	5,5270	0,009912222	15	5,5270	0,009912222

Part III

Simulation Model

Chapter 6

Simula Language

The SIMULA programming language was created at the Norwegian Computer Centre (NCC) by Ole-Johan Dahl and Kristen Nygaard. Its first version was developed in 1962, called SIMULA I. 5 years later SIMULA I evolved into SIMULA67, by adding among other things the concept of inheritance. At the beginning It was created as a language for discrete events simulation, but turned out in addition to posess interesting properties as a general programming language. SIMULA is known as the first Object Oriented Languaje. Among other things SIMULA introduced important object-oriented programming concepts like classes and objects, inheritance, and dynamic binding. It is not the aim of this project to teach SIMULA. More information about SIMULA: history, links, . . . can be found here [SIM06a] [SIM06b].
To create our simulation, we will use DEMOS. DEMOS is a SIMULA package created to help on the development of discrete events simulations. An introduction to DEMOS can be found at [DEM06b] and [SIM06b] [DEM06a]

Chapter 7

Boost's Simulation Model

The Simulation model has been built from the assumption that the foremost critical resources will be the Processor and the Hard Drive. A sample code can be found at Appendix .1.

7.1 Parameters

7.1.1 Incomming Traffic Model

We will use an exponential function to generate the Incomming traffic interarrival time.

7.1.2 Service Time

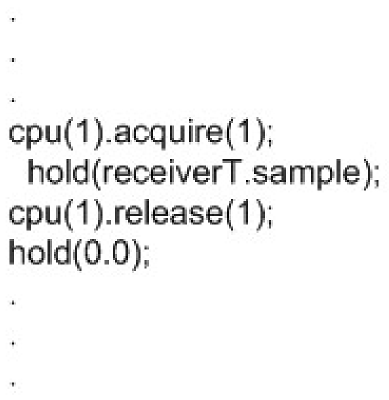
For the analytic model we choosed an exponential service time because It simplified It. But, for the simulation model we have chosen a Normal distribution to represent the service time for every operation, because its sample values are more "stable" within a range. On Apenndix .1 we can see the values we have choosed. For example, the receiver service time for accepting connections is a normal distribution with $\mu = 0.1$ and $\sigma^2 = 0.01$. With this values the probabiliti to have a value between $[\mu - \sigma, \mu + \sigma] \approx = 0.68$ and the maximum value will be μ, meanwhile with an exponential distribution this probability is $\approx$0.038. Using the Normal distribution we can also have a problem if our average value is close to zero, we can have negative samples. To avoid It, we used the function Abs(), that will return a positive value.

7.2 Shared Resources

The main objective of the simulation is to show how the use of shared resources affect our performance. The Shared Resources have been modeled as an array of *RES* classes. We focused on the processor, the hard drive and the threads.

7.2.1 Processor

To model the CPU-Share behaviour we have divided every module in small tasks. Before performing any task we will have to ask for the CPU resource. Once the task is finished we will return the resource. A sample of this is shown on figure 7.1. We use the commnad *hold(0.0)*to "jump" from task to task. When we execute this command the next entity in queue for the resource will get It.



```
.
.
.
cpu(1).acquire(1);
  hold(receiverT.sample);
cpu(1).release(1);
hold(0.0);
.
.
.
```

Figure 7.1: CPU share example

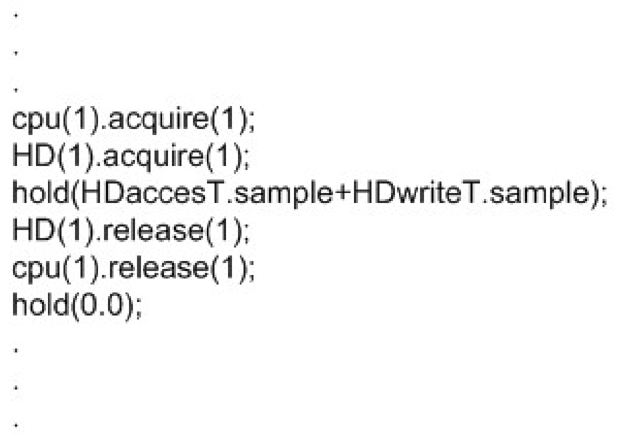
7.2.2 Hard Drive

The Hard drive will be another important resource in our simulation. It has a direct relation with the database, because It is where the data will be stored. We will assume that every action regarding the hard drive is composed by the accessing time to the hard drive and the time needed for the operation itself, like writing and deleting. A sample code is shown on figure 7.2

We will use different Hard Drive configuration in our simulations, from a single Hard Drive to a multiple Hard Drive using RAID.

7.2.3 Threads

The threads will be another shared resource but instead of acquire and free them every task, we will acquire them at the begining of a module: receiver, dispatcher, ... and we will free them when all the tasks in wich have been divided the module are done. On Figure 7.3 we can see wich tasks are needed for every module. This relation tasks-module is not allways that obvious. If we take a look at the Figure 7.3 It seems that the product is part of the dispatcher, but they are two different modules. The problem is



```
.
.
.
cpu(1).acquire(1);
HD(1).acquire(1);
hold(HDaccesT.sample+HDwriteT.sample);
HD(1).release(1);
cpu(1).release(1);
hold(0.0);
.
.
.
```

Figure 7.2: HD share example

that the dispatcher waits for the Product to finish, so we can not release the dispatcher thread until the product is finished.

7.3 Simulation workflow

The Simulation will follow the life cycle of an incoming request. On Figure 7.3 we can see how the simulation runs. We have 3 entities: the Generator, the Incoming Message and the Outgoing Message.

- The Generator creates an Incoming Message entity every $\frac{1}{\lambda}$ ms or 166,67 *messages/second*.
- The Incoming Message entity simulates the operations done by the system from the message reception by the dispatcher to the message delete from the incoming queue by the dispatcher.
- The Outgoing Message entity simulates the system from the moment the Producer module generates the response until this response is delivered.

It is important to realize that the life cycle of an incoming request will not be the add of the life cycle of modules operations, but the add of parts of It. In particular, the message removing from Incoming/Outgoing queues will not be part of It. So we can say that the only modules that will completely be part of the life cycle of an incoming request will be the Receiver, the Product and the Sender. This fact will be important when we will be doing the analysis of the simulation's results.

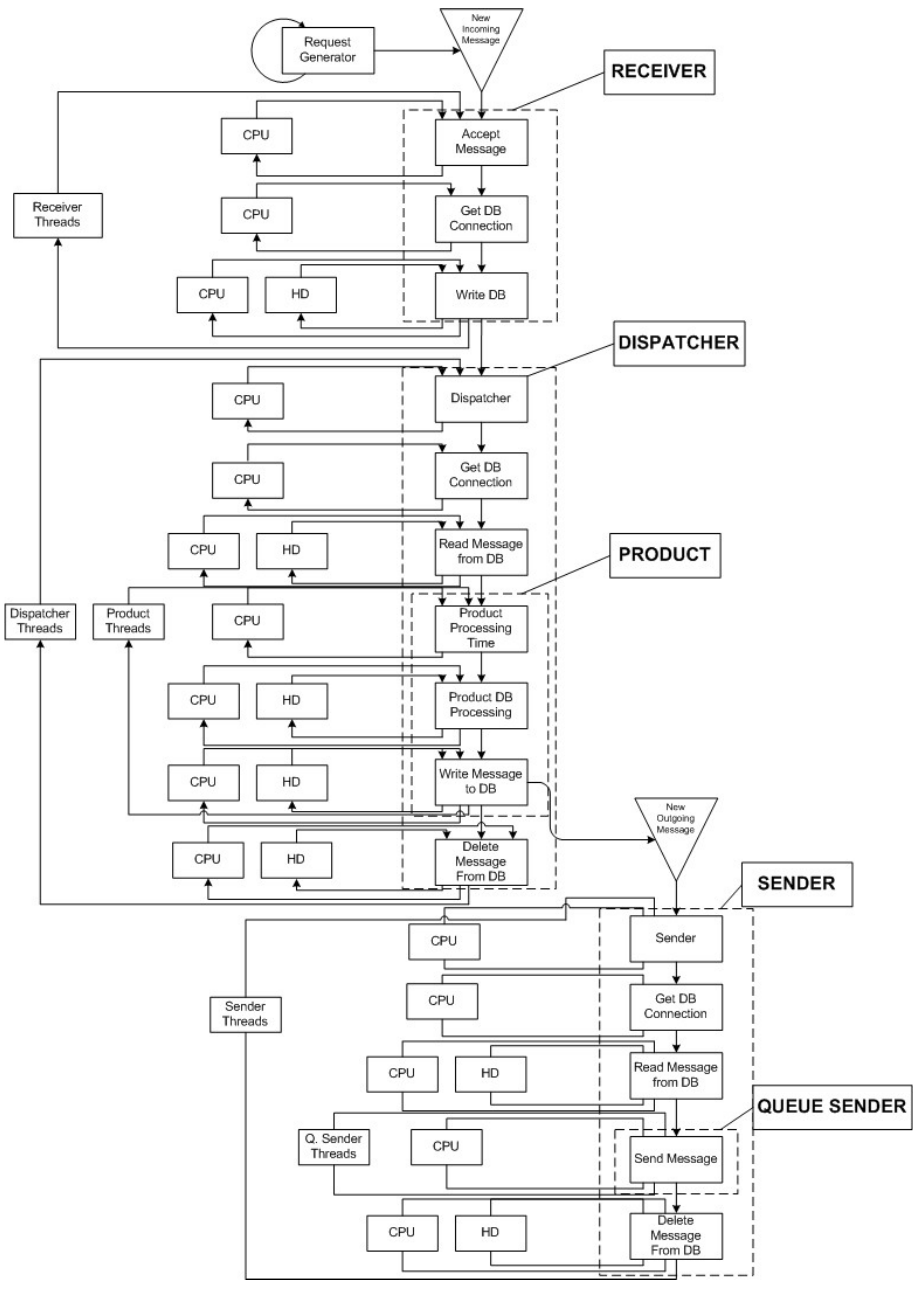


Figure 7.3: Simulation Diagram

7.4 Simulation Models

We will run the simulation "step by step". We will use 3 different configurations, from a simple to a more developed one. This configurations will be made after real systems. Despite the fact that we have 3 different configurations, every configuration will be runed with 3 different Hard Drive processing time. Every Hard Drive operation will require two operations: seeking + operation. The operation can be write, delete or select. We will assume that the seeking time is the same for the three of them, but the operation time will decrease. The operation time will be affected by various parameters, like the use of a RAID system, The communications protocol: SCSI, Serial ATA, . . . , or even the Hard Drive brand. We will call to our reference operation processing time noRAID, then we will have an operation time equal to reference*0.6 (RAIDx2) and the third one will be reference*0.4 (RAIDx3).

7.4.1 Single Multiprocessor Shared Hard Drive

This will be our first simulation. We will also use this one to initialize our service time values, according to the results on section ()(This configuration is the same we used when we tested a real system at Boost.) It will simulate a single multiprocessor machine, from 1 to 10 CPU. The "Shared Hard Drive" means that the 3 databases (incoming, product and outgoing) will be running on the same Hard Drive.

7.4.2 Single Multiprocessor Different Hard Drives

This configuration will also use a single multiprocessor machine, but in this case we will split the databases. We will run them in separate Hard Drives. This will be our first performance improving mesure for the system.

- On Hard Drive 1 will be running the incoming database
- On Hard Drive 2 will be running the product database
- On Hard Drive 3 will be running the outgoing database

7.4.3 Double Multiprocessor Different Hard Drives

This configuration has been made after the Boost specificationsof their system. We will have 2 different multiprocessor machines: CPU1 and CPU2.

- On CPU1 we will run the Receiver module and its database.
- On CPU2 we will be running the dispatcher, product, queue sender, sender and the product/outgoing database. Boost have chosen this

configuration to isolate the incoming requests from the rest of the process, so they can easily control them.

Chapter 8

Running the Simulation

We will start with a model equal to the system we have tested. This is a receiver connected to the dispatcher, but without any product module. We will adjust the values close to the ones we get from the tests and that will be our starting point. From there, we will add the product (a generic one) and we will start the simulation.

8.1 Interpreting the results

We will analyze the simulation results in two steps:

- First we will check our system throughput. With the throughput we will be able to know if our system is stable or not stable. $\lambda_{IN} = Throughput$ means that we have a stable system.

- Second, we will check our latency. To have an idea of the system latency, we will have to look at the system time, or the average time since a message enter the system until It leaves It. Since we do not have any quality values to compare with, we will not be able to qualify the value as good or bad one.

With this two values we can have an idea of our system quality. For example: A train loaded with DVD will be a system with a huge througput, but its latency will be very large. From an Internet user point of view, the 56k modem will be much faster if he tries to open a web-site, use the e-mail,... On the other hand, a user trying to download a DVD, may find a better choice to wait one day for the train.

On Appendix .2 we can see a sample simulation result. The result file has 4 main parts: distributions, counts, tallies and resources.

8.1.1 distributions

In this part we have a list of all the distributions used during the simulation. It shows its main values: reset time, number of observations, type, average value, standar deviation (if we can choose the value) and the seed value. It is important to notice that if we run 2 simulations using the same seed, the result will be exactly the same.

8.1.2 counts

We have only one count variable. It is used to count the number of messages sended. We will use this valu to calculate the system throughput: $\frac{Messagessent}{Simulationtime} = [messages/second]$

8.1.3 tallies

We used the tallies to calculate the threads duration, and the system time. We have timestapms inside the simula code, so que can folow the message inside the system and calculate how long It takes every module to process a message.

8.1.4 resources

On this part we have listed our shared resources, with its utilization, standard deviation, ...

8.2 First Simulation

We will start our simulation with a time between arrivals $1/lambda$=6ms. That was the best result we had during our tests on the Boost System (see section **??**). We will also configure the number of threads according to the results from the tests: 1 Receiver, 3 Dispatchers, 3 Products(dispatcher and product are binded, see section 3.2), 3 Senders, 1 Queue Sender. As we saw of section 5.3, even having a small processing time, the system time can be much bigger depending on the value ρ, the probability of busy sytem. But this time we do not have a poisson distribution output, because our processing time has been modeled as a normal distribution. So to calculate the system time we have to use the pollaczek khintchine formula:

$$T = \frac{\lambda E\left\{x^2\right\}}{2\left(1-\rho\right)} + E\left\{x\right\}$$

Where x is the aleatory variable representing the service time.

We will have 3 different configurations for our simulation:

8.2.1 Single Multiprocessor Shared Hard Drive

On Figure 8.1 we have the throughput for the first simulation. Knowing that our input ratio is $\lambda = 166\ messages/second$, we can conclude that with this configuration our system is unstable, our throughput is lower than λ. To find the bottleneck we have to look at Figure 8.2 and 8.3. From 1 to 4 CPU our bottleneck is the Processing Power, but at the end, what is limiting our system is the Hard Drive, its utilisation level is 100%. No matter how many processors we use, our shared Hard Drive configuration will not be able to handle all the incomming traffic.

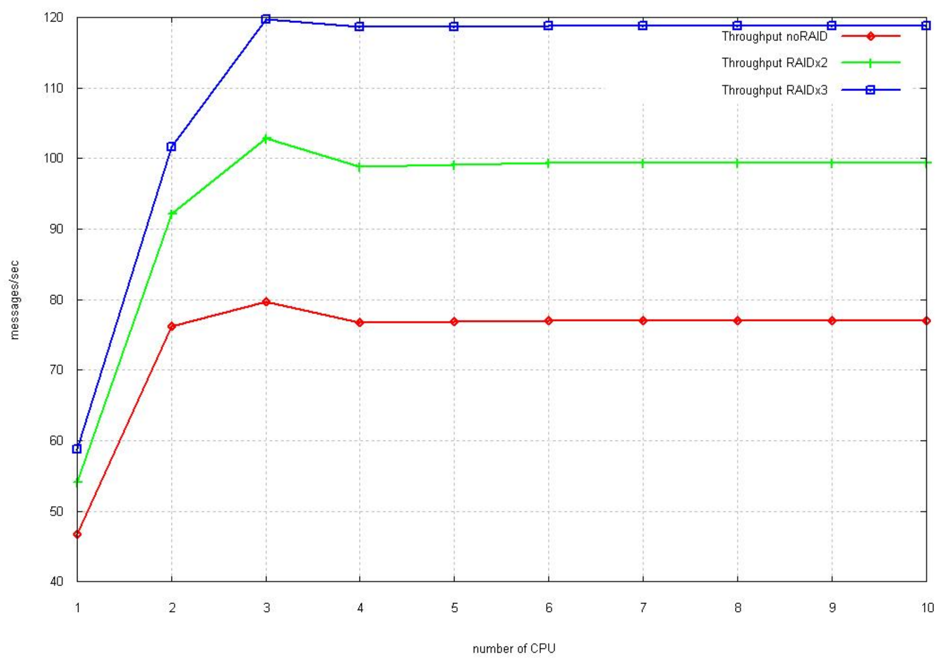


Figure 8.1: Throughput Configuration 1: Single Multiprocessor 1 Hard Drive

8.2.2 Single Multiprocessor Multiple Hard Drive

On Figure 8.4 we have the throughput for our 2nd simulation. What we can see is that separating the Databases into different hard drives we are able to reach the stability point, that is when the throughput is equal to λ, but depending on the Hard Drive speed we will need 3, 4 or 5 CPU. Once we know that our system is stable, we can talk about the latency. To do It we will have to check the system time.
On Figure 8.5 to 8.8 we have the average system time for processing 1 message and its standard deviation. Despite of the Hard Drive configuration we use, we can see that the system time graphs have the same behaviour.

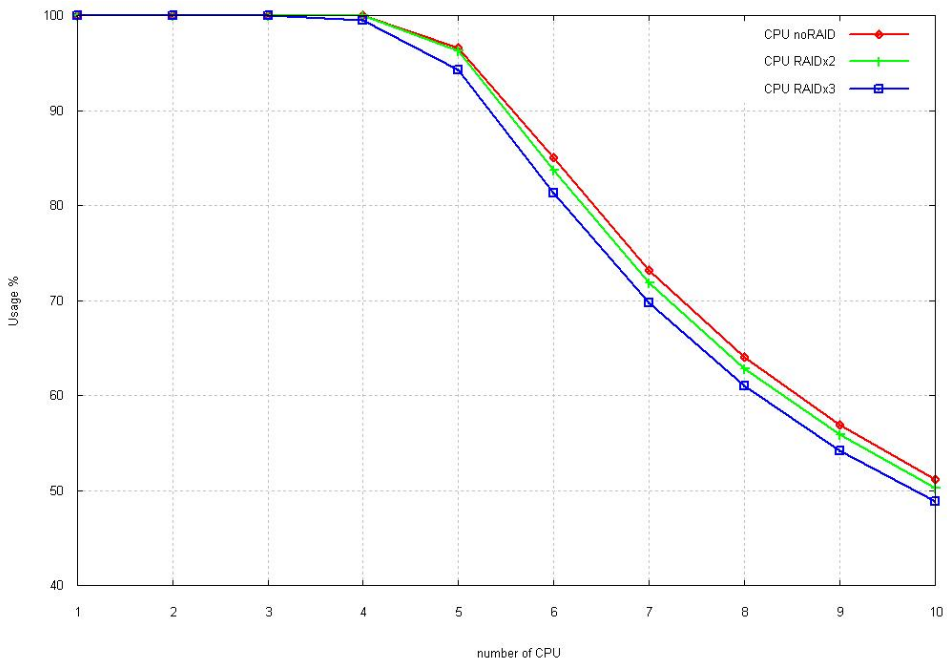


Figure 8.2: CPU usage: Single Multiprocessor 1 Hard Drive

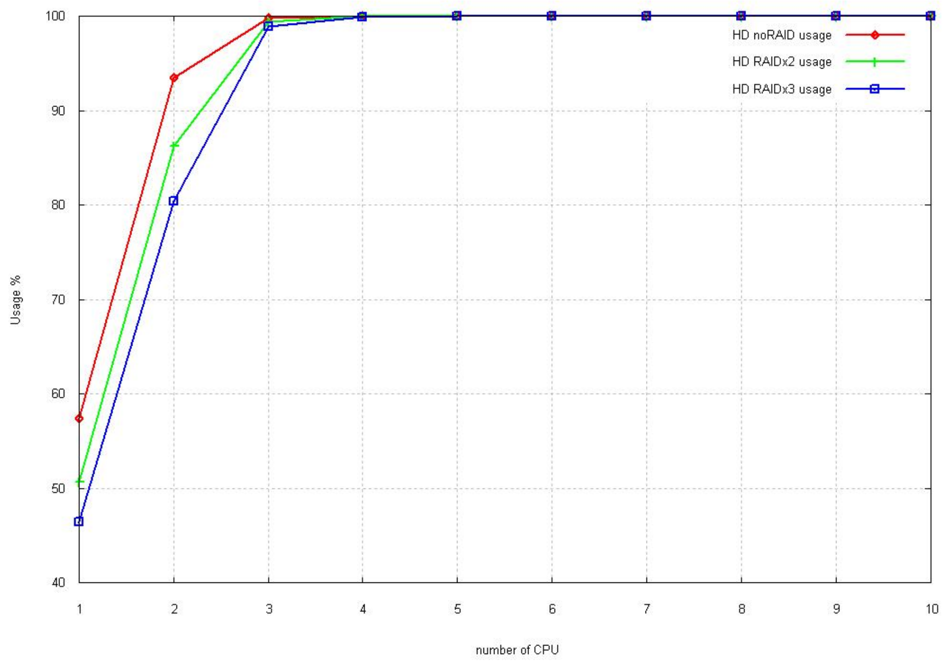


Figure 8.3: Hard Drive Usage: Single Multiprocessor 1 Hard Drive

The system time decreases very fast until It reaches the equilibrium point, then it decreases very slowly for two steps more (2 CPU more) and It remains stable for the rest of the simulation. Yet its final value is much better as faster the Hard Drive is, what means that we have a better (lower) latency. The fact that the graph lines reach a stable value means that at this point our busy probaility ρ has reached a low value, tipically 60 or 70$, or one of the shared resources is limiting It. We can discard the CPU as a limiting resource, because we increase its number up to 10. On the other hand we have the Hard Drive resource, which is not increased. On Figure 8.9 to 8.11 we have the Hard Drive usage. We can see that the Hard Drives for the incoming and the product database reach a level of usage above 80%, for the noRAID configuration. Meanwhile on the other configurations its level is close or below 60%. That explains why the system time is much bigger for the noRAID configuration.To find the cause of this bottleneck, we have to look at Figure 8.12 where we can see that the dispatcher usage is allways above 90%, meanwhile the other two hard drive configurations 8.13 and 8.14 the usage about 70-80%.
If we take our worse value for the Total Time (figure 8.6):$Average = 77ms$, $\sigma = 61ms$, and asuming that we need an average processing time = $1/\lambda$ = 6ms. This means that our latency time is more than 10 times oour processing time. On the other hand, our best system time is about 20 ms Average with $\sigma = 10ms$. The big difference between this two results is due to the usage of the system resources.

8.2.3 2 Multiprocessor Multiple Hard Drive

On the 2 Multiprocessor graphs we can see that we have many lines, look like a mess. To know how many CPU are we using, we have to look at the x axis where we find the number of CPU2, and the number of CPU1 is represented by the multiple lines of the same color.

On Figure 8.15 we have the througput for the third configuration. First thing we notice is that It does not help to achieve the stability point the fact that we can add CPU to the receiver, with 1 CPU It is enough, except for the noRAID configuration, when we need 2. This means that the Receiver and the *incoming messages database* CPU consumtion is equal or less than one. On the other hand, we have the number of CPU2 needed to reach the stability point, that is 3, no matter how many CPU1 we have or wich HD configuration we are using. On Figure 8.16 to 8.19 we have the average system time for processing 1 message and its standard deviation. On the previous section we saw that the system time graphs had the same behaviour, the only differenece despite the value was that the graph was "delayed" one step (CPU), because the stability point was reached in different moments depending on the CPU configuration. In this case the stability point is reached at the same time 3-CPU2 for every configuration. This re-

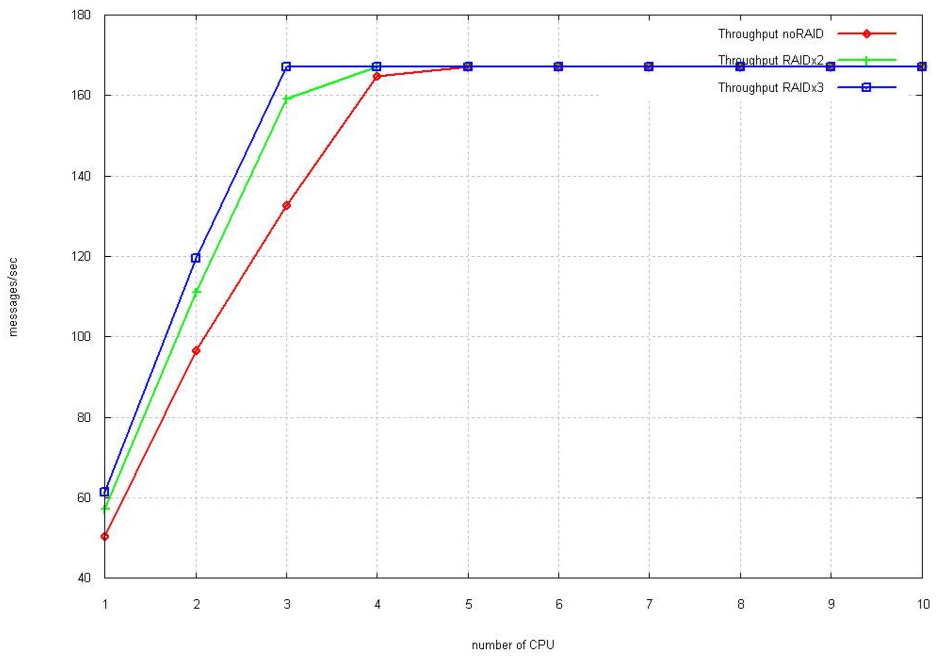


Figure 8.4: Throughput Configuration 2: Single Multiprocessor 3 Hard Drive

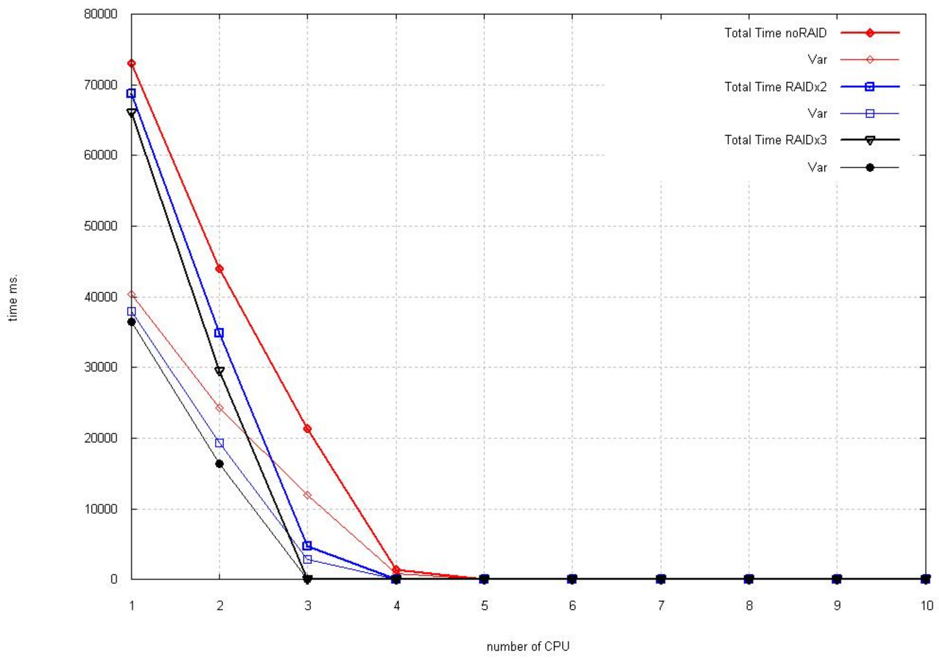


Figure 8.5: System Time for Configuration 2: Single Multiprocessor 3 Hard Drive

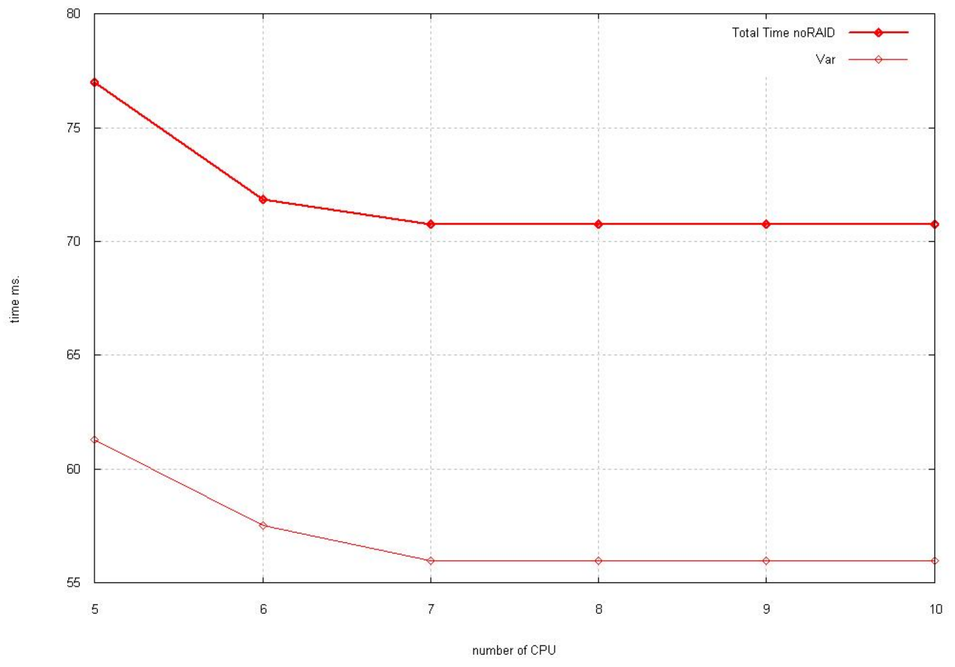


Figure 8.6: Zoom System Time Configuration 2: noRAID

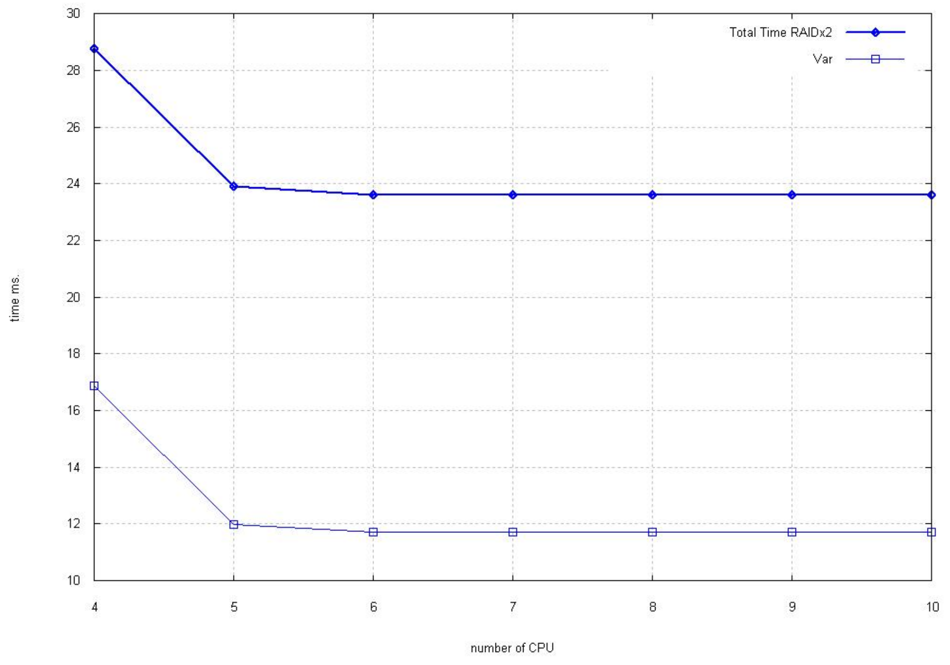


Figure 8.7: Zoom System Time Configuration 2: RAIDx2

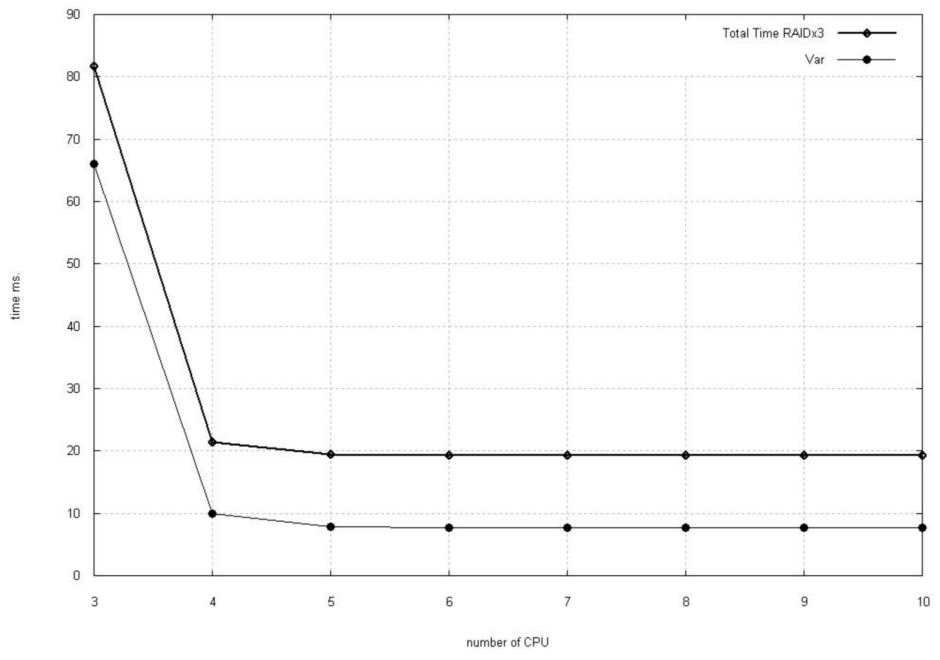


Figure 8.8: Zoom System Time Configuration 2: RAIDx3

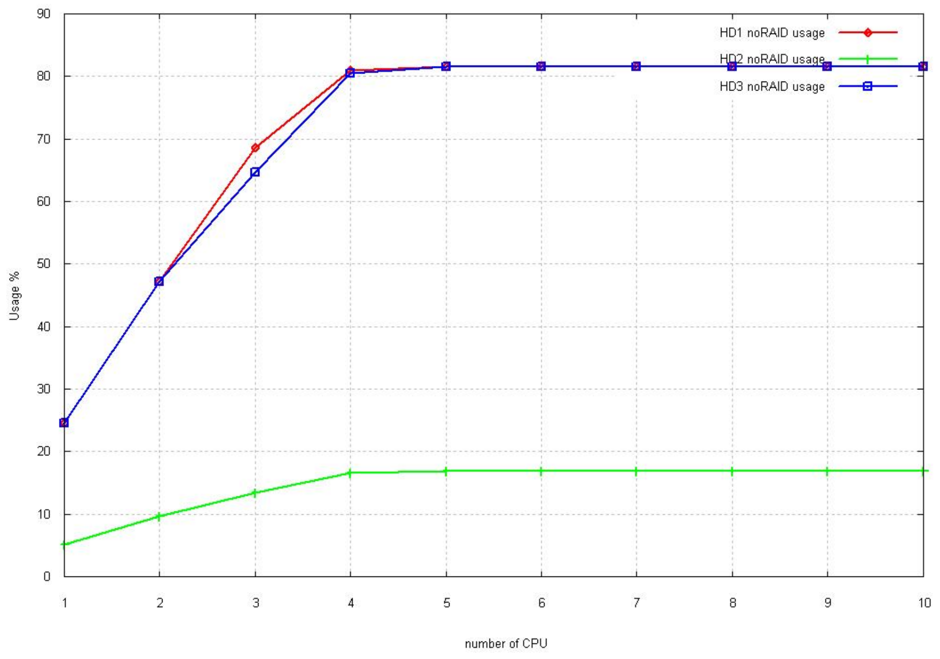


Figure 8.9: Hard Drive Usage Configuration 2: NoRAID

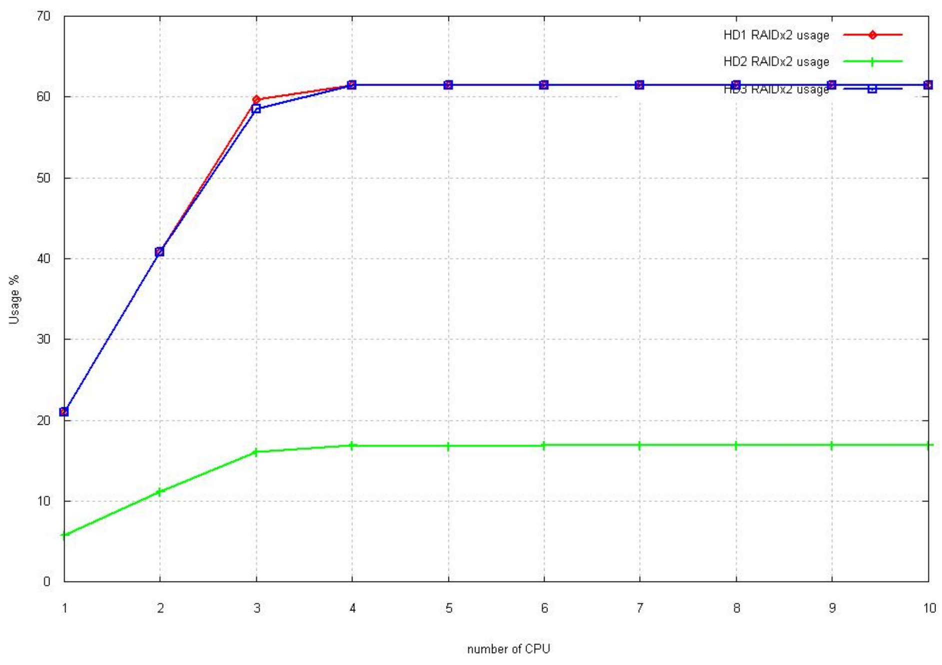


Figure 8.10: Hard Drive Usage Configuration 2: RAIDx2

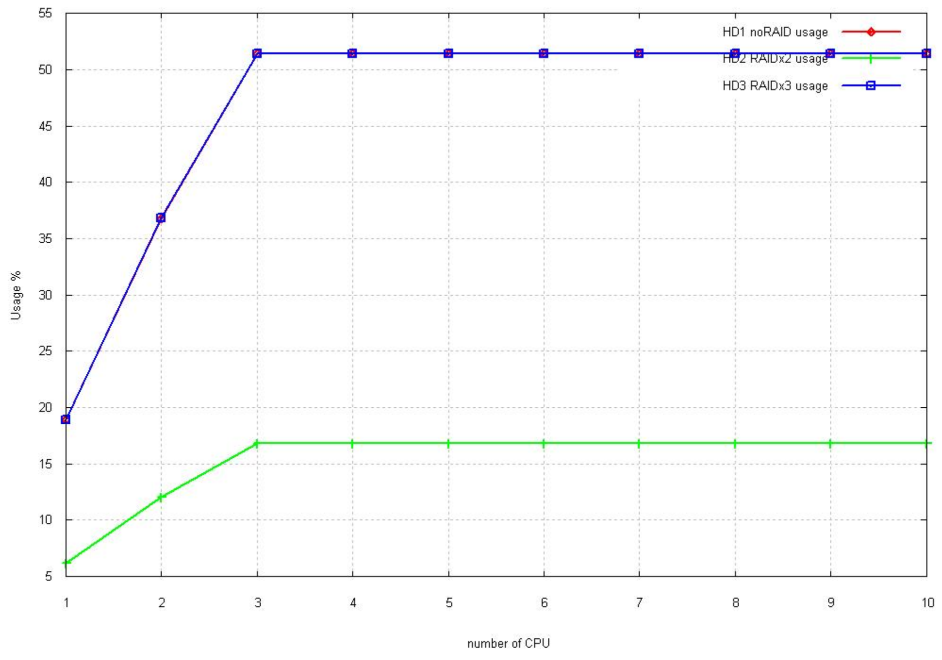


Figure 8.11: Hard Drive Usage Configuration 2: RAIDx3

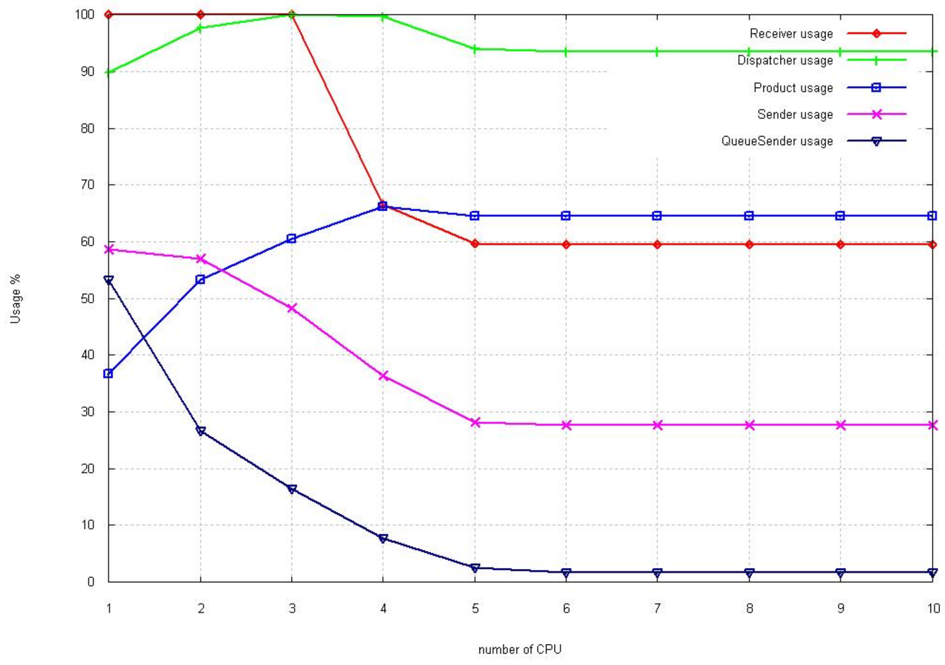


Figure 8.12: Threads Usage Configuration 2: noRAID

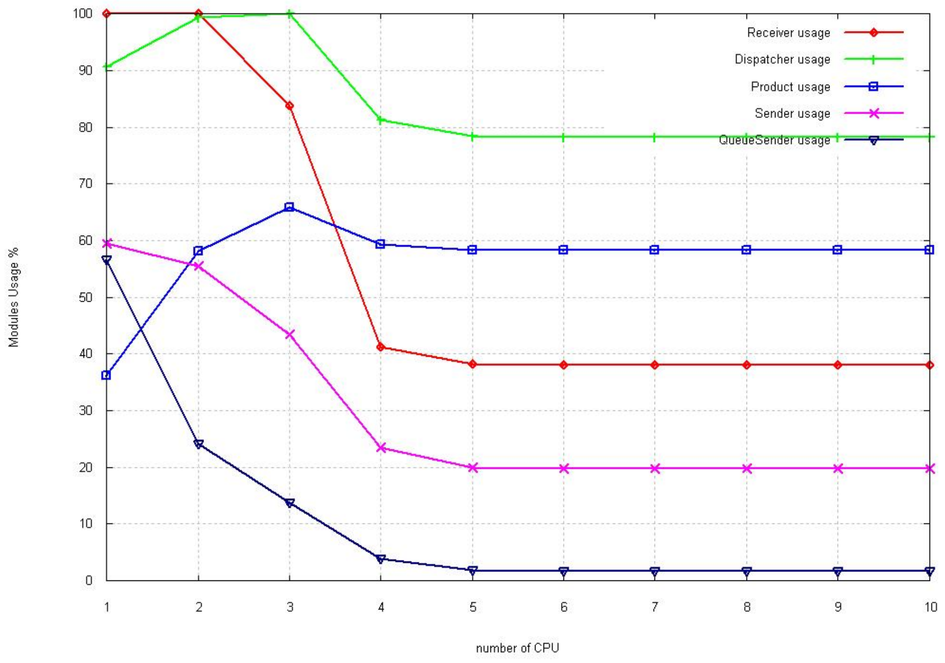


Figure 8.13: Threads Usage Configuration 2: RAIDx2

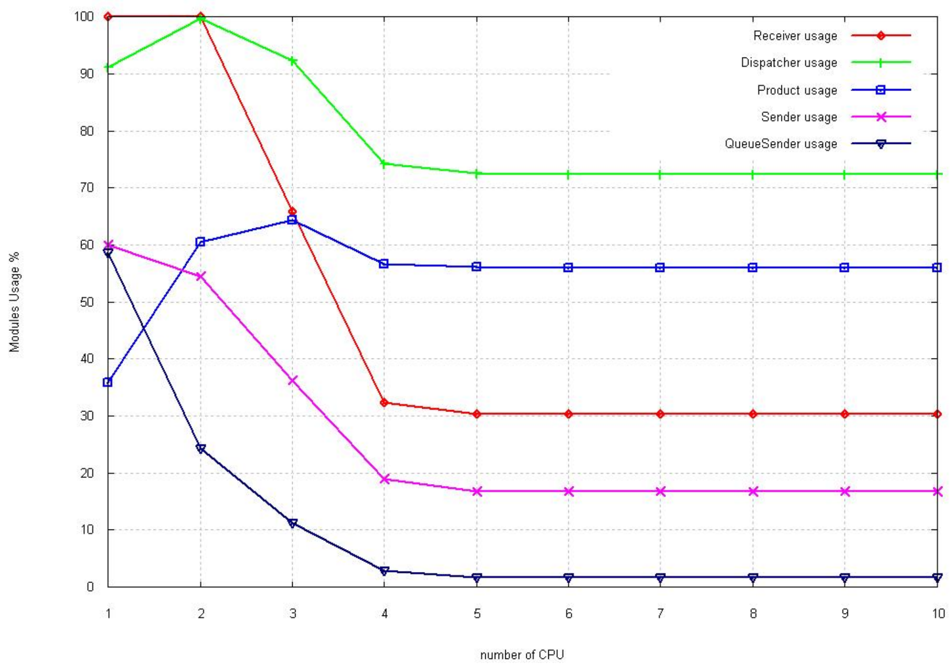


Figure 8.14: Threads Usage Configuration 2: RAIDx3

sult is translated into the evolution of the graphics: The time decreases very fast until the equilibrium point, It decreases slowly to the next step and It remains stable. The influence of the CPU1 can be seen through the fact that we have 2 different values related to the number of CPU1, one for 1CPU1 and another one for 2 to 10 CPU1. The Hard Drive speed will have influence in the system latency, as faster the hard drive is, less latency we will have. If we have a look at Figures 8.20 to 8.22 we will see that we are in the same situation than the previous configuration, with one multirpocessor machine. For the noRAID configuration we have a HD utilisation above 80% for the product and the incoming database. It is an expected result, since we found out that the resource that was limiting our system was the Hard Drive not the CPU. Again, if we have a look at the threads utilisation Figure 8.23, we find that the dispatcher is above 90% for the noRAID configuration, and .Figure 8.24 and Figure 8.25 the maximum usage is about 70-80%.

8.2.4 First Simulation Conclusion

With this first simulation we have seen that an obligated choice if we want to be able to handle an icoming traffic ratio $\lambda \geq 166\ messages/second$ is to separate the databases into different hard drives. We have also seen that the Hard drive speed will have a big influence in the quality of the system, we will need less CPUs and we have a better latency. Based on

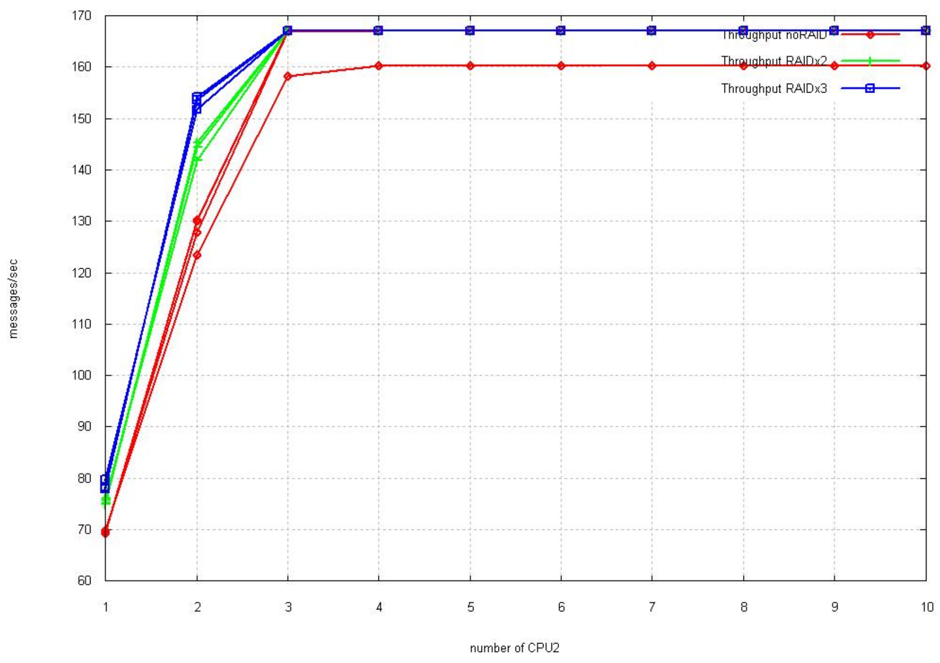


Figure 8.15: Throughput Configuration 3: Two Multiprocessors 3 Hard Drive

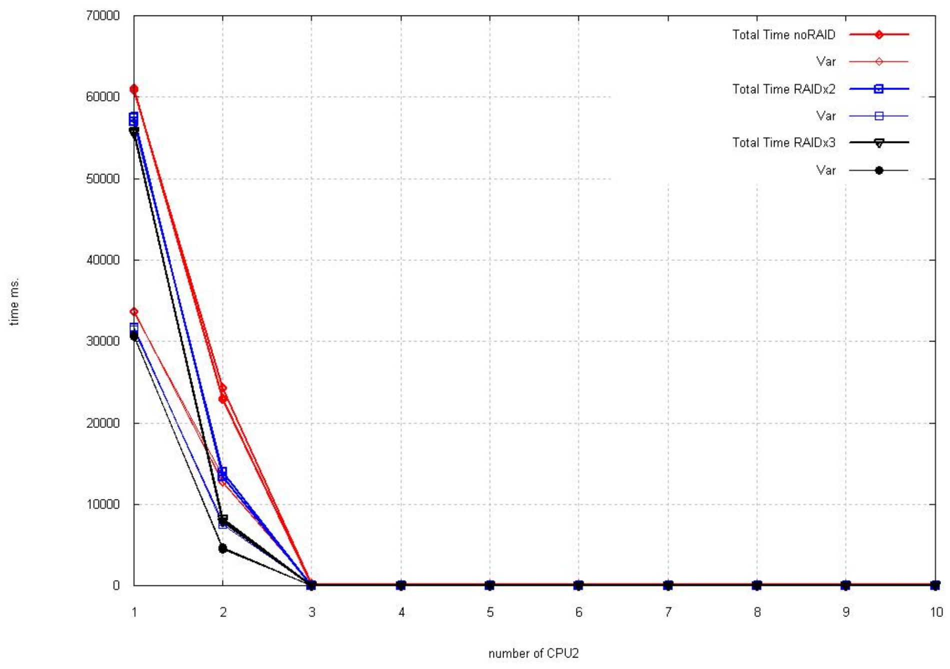


Figure 8.16: System Time for Configuration 3: 2 Multiprocessors 3 Hard Drive

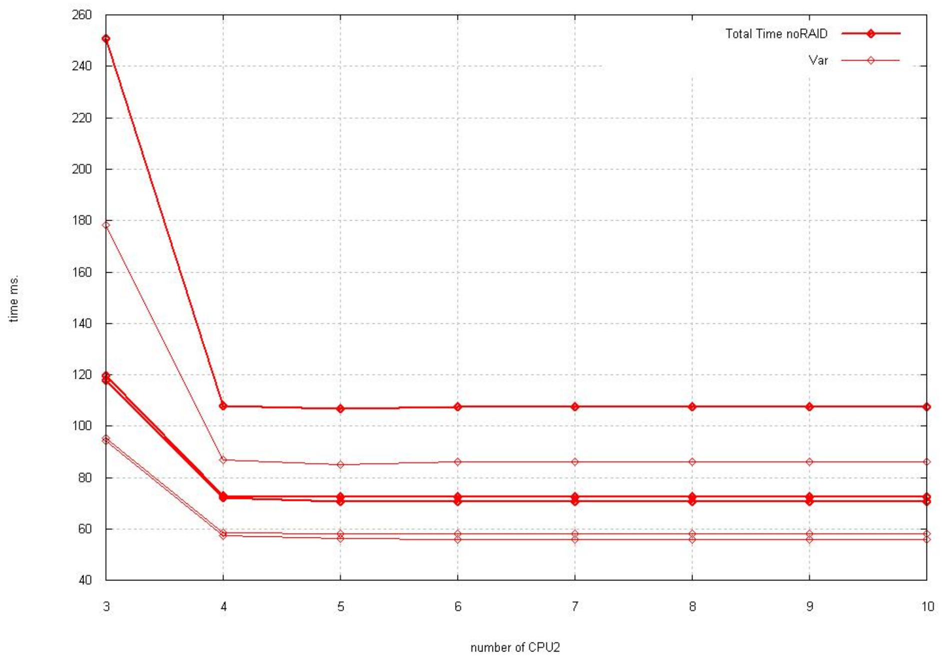


Figure 8.17: Zoom System Time Configuration 3: noRAID

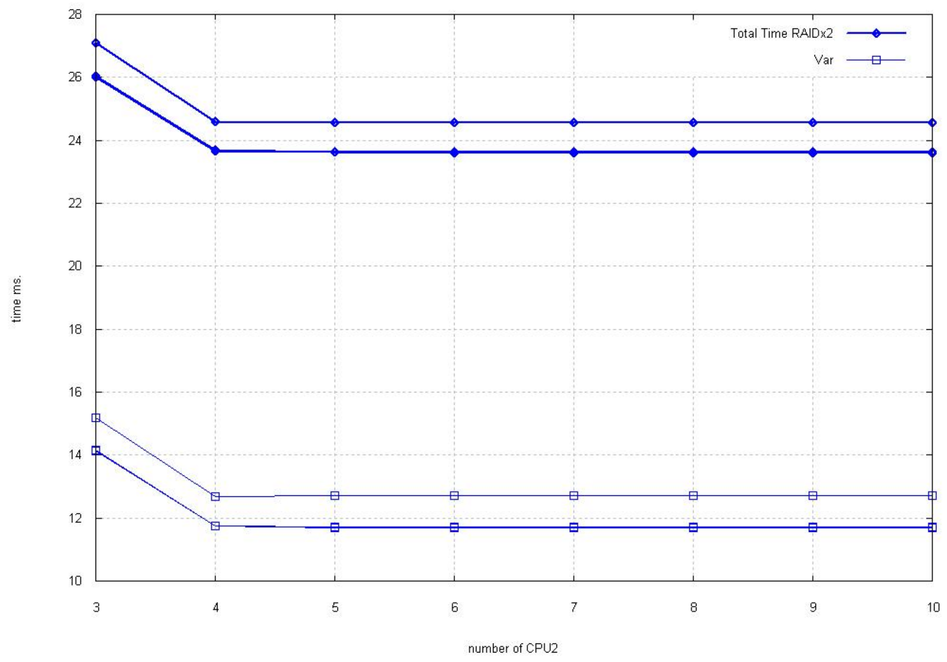


Figure 8.18: Zoom System Time Configuration 3: RAIDx2

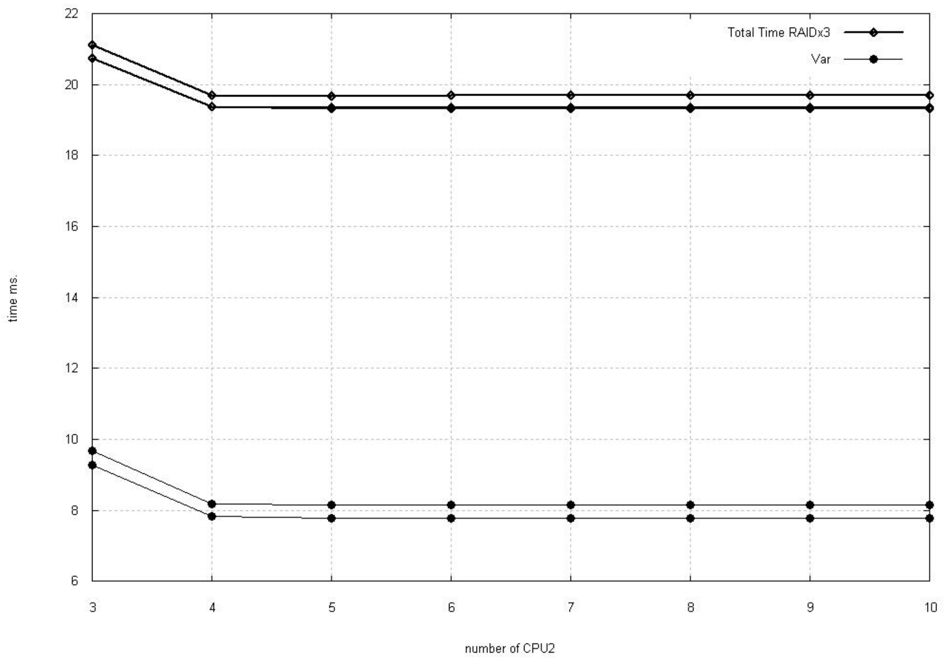


Figure 8.19: Zoom System Time Configuration 3: RAIDx3

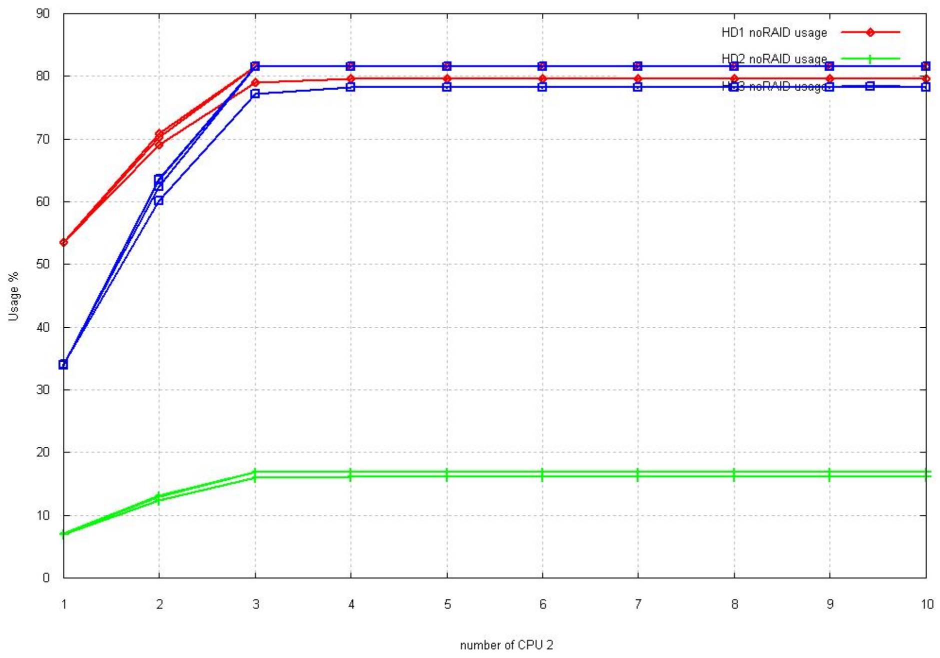


Figure 8.20: Hard Drive Usage Configuration 3: NoRAID

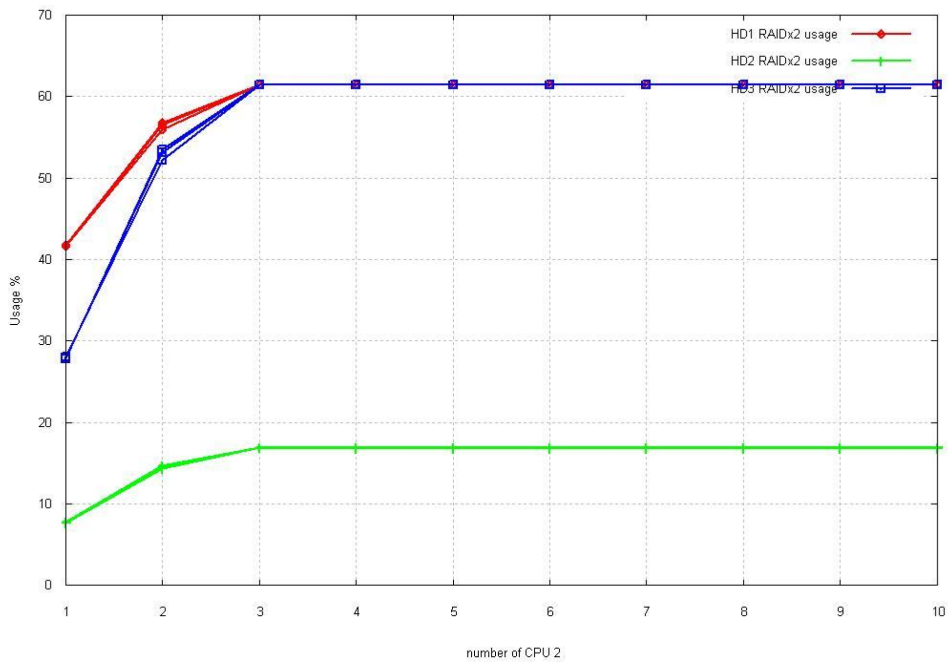


Figure 8.21: Hard Drive Usage Configuration 3: RAIDx2

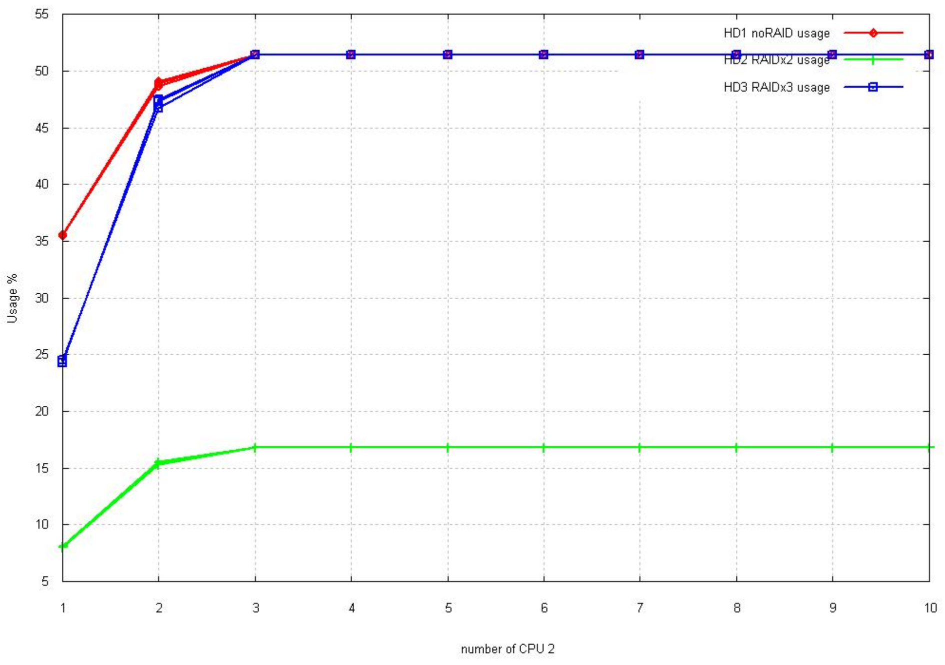


Figure 8.22: Hard Drive Usage Configuration 3: RAIDx3

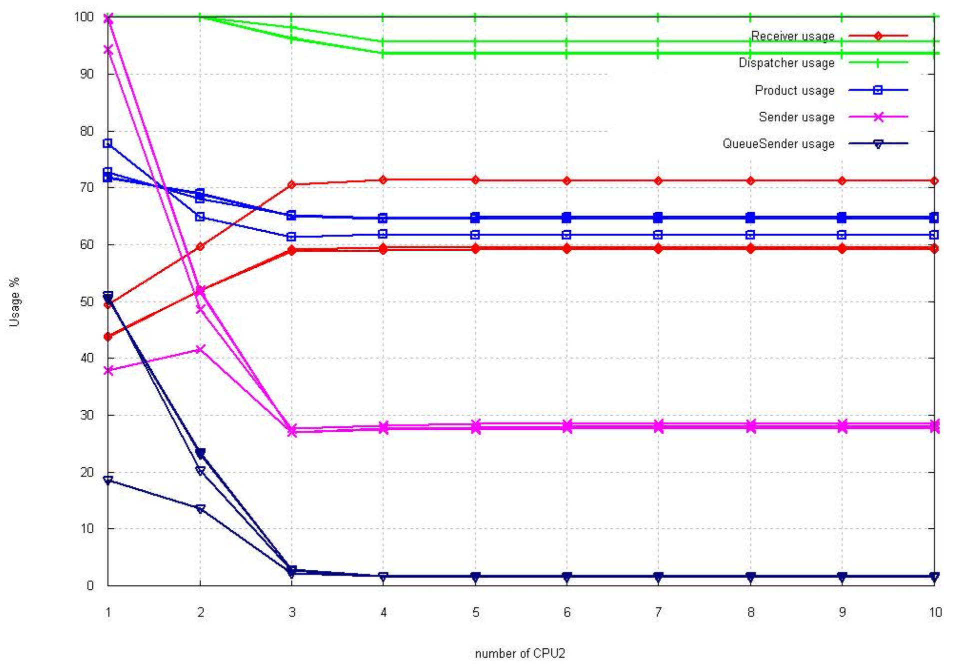


Figure 8.23: Threads Usage Configuration 3: noRAID

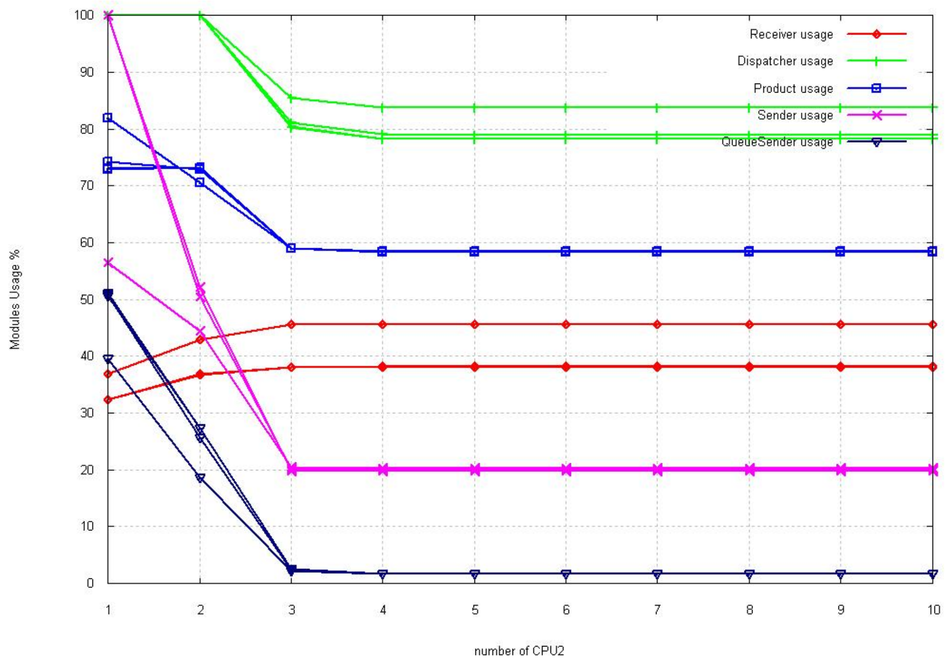


Figure 8.24: Threads Usage Configuration 3: RAIDx2

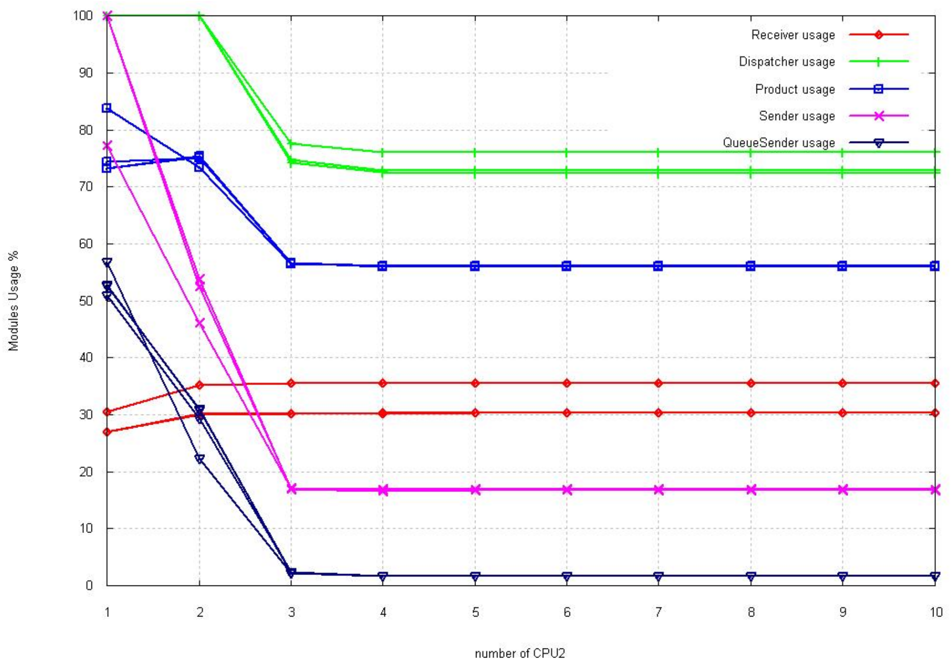


Figure 8.25: Threads Usage Configuration 3: RAIDx3

perfromance criterium the 2 multiprocessors configuration is not needed. We can obtain the same results using only one multiprocessor machine. But using criteriums of security or system availability It is a matter that can be discused.

8.3 Increasing Interarrival Time

On the previous section we tested our simulation with a $\lambda = 166 messages/second$, and we saw that only the 2nd and 3rd configuration were able to handle that traffic.
In this section we are going to push our system harder, we are going to double the traffic $\lambda = 333\, messages/second$ and see how our configurations handle It.

8.3.1 Doubling Incoming Traffic

On Figure 8.26 and 8.26 we have the throughput for our 2nd and 3rd configuration with $\lambda = 333$. We can see that we are not able to reach the equilibrium point in any case. For our single multiprocessor configuration, we also see that our CPU needs have increased from the previous simulation, at least up to 5, when our throughput remains stable. Meanwhile, on the double multiprocessor configuration we see that the CPU needs have increased only on the CPU1, the receiver, while the CPU2 needs have slightly increased by one.

If we check the Hard Drive utilisation, Figures 8.28 to 8.33, we will find out that in both cases, the Hard Drive utilisation for the noRAID configuration reach levels beyond the 90% of usage, what means that the system is limited by this resource. But, on the other 2 Hard Drive configurations, and in both cases, the utilisation is close to 80%. This means that we are not using all our system resources. So the RAIDx2 and RAIDx3 throughput is limited not by the sytem resources, is limited by the use we make of them: we can add more threads.

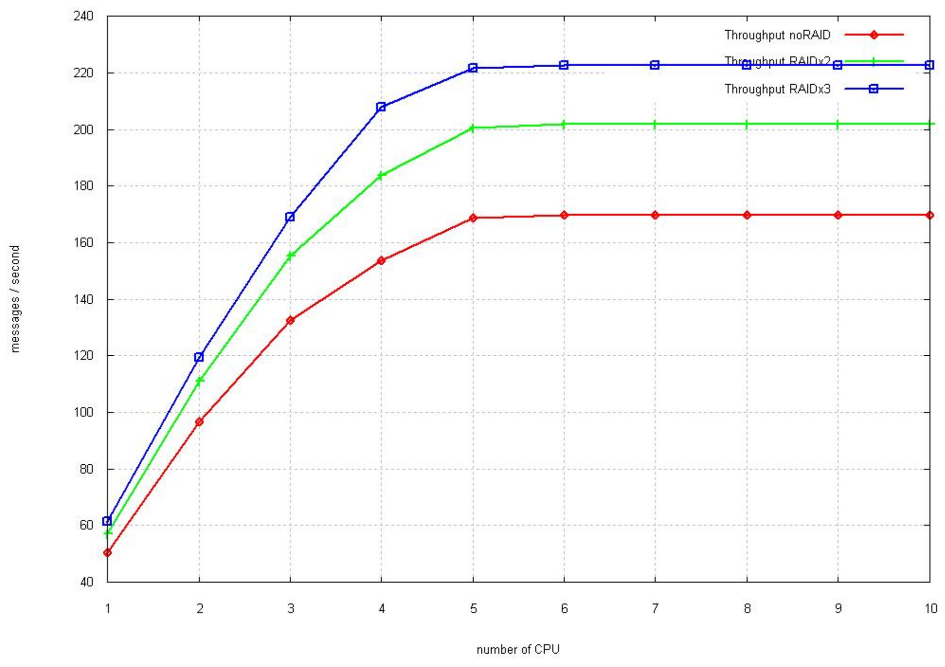


Figure 8.26: Throughput Configuration 2: Single Multiprocessor 3 Hard Drive

8.3.2 Doubling the threads number

On the previous section we found that we can increase our throughput by increasing the number os threads, because our resources utilisation were below 100%. On Figure 8.34 and 8.35 we have the throughput for a simulation with twice number of threads, that is: 2 receiver, 6 dispatcher, 6 product, 6 sender, 2 queue sender. Again, we find that we can not reach the throughput objective: 333 messages / second. To find the limitation resource we have to look to Figure from 8.36 to 8.41 where we have the

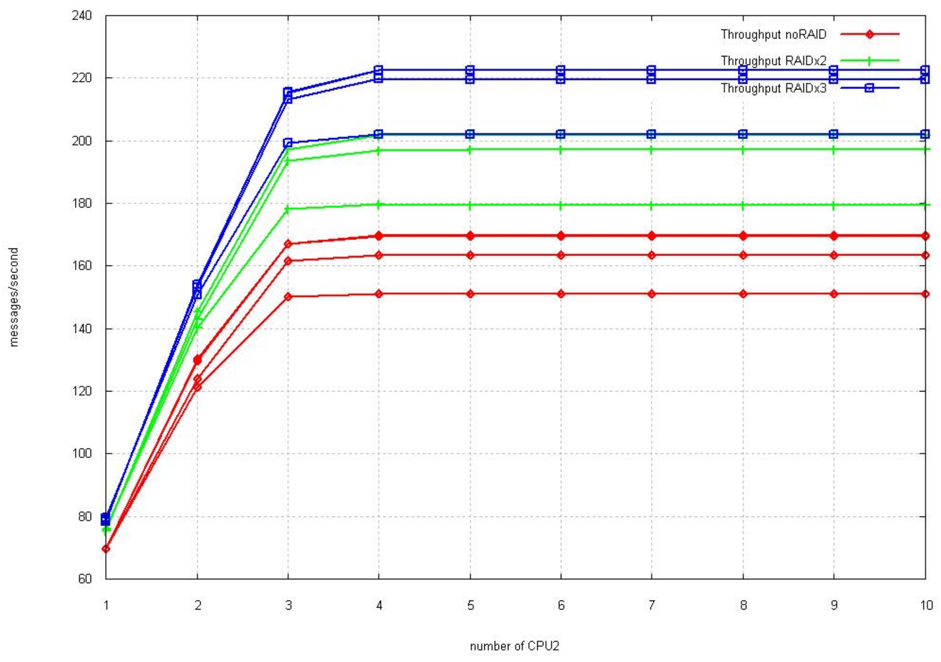


Figure 8.27: Throughput Configuration 3: Two Multiprocessors 3 Hard Drive

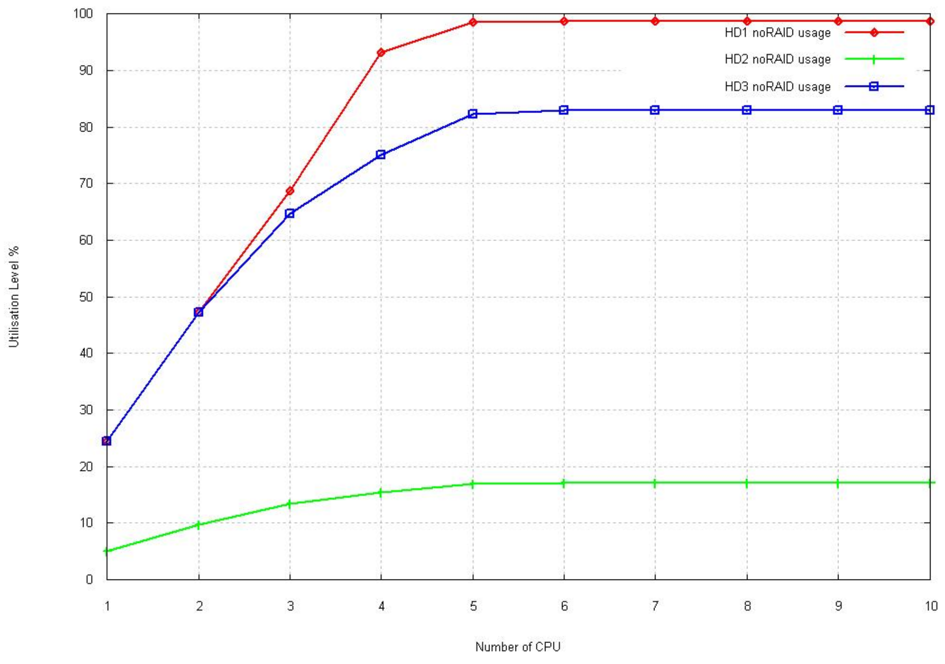


Figure 8.28: Hard Drive Usage configuration 2 noRAID: double traffic

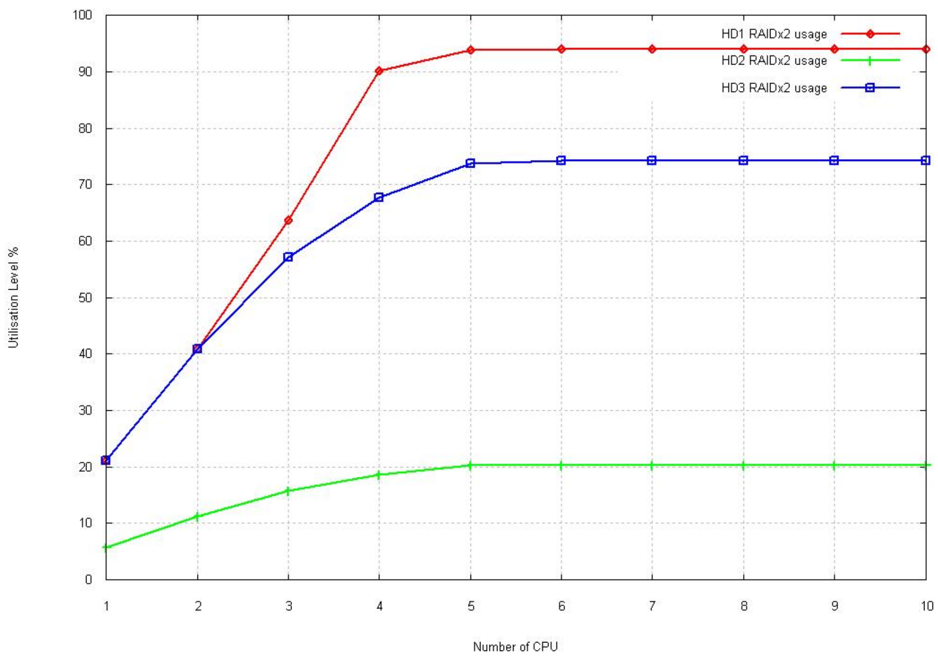


Figure 8.29: Hard Drive Usage configuration 2 RAIDx2: double traffic

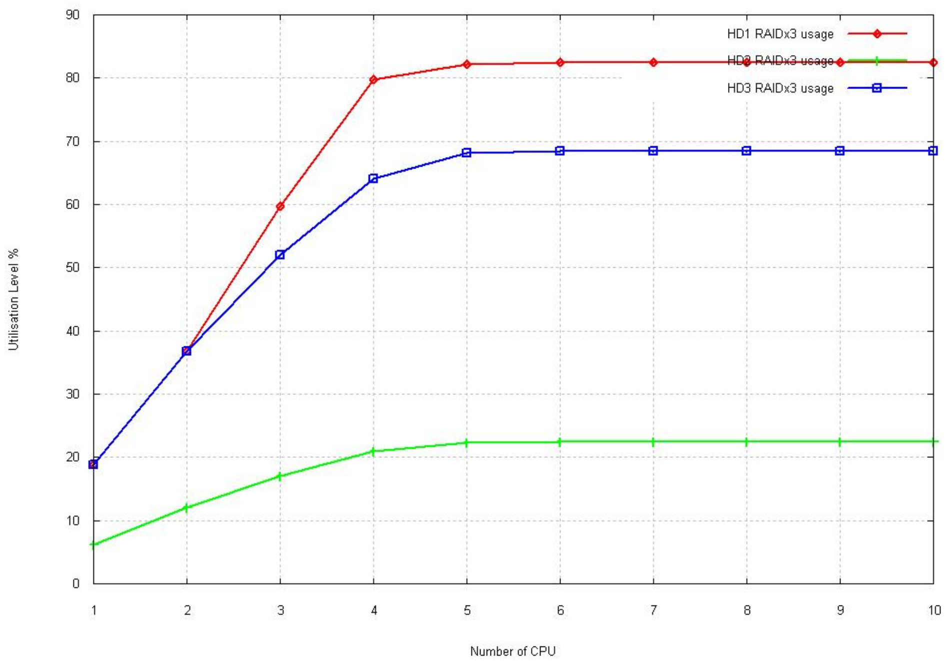


Figure 8.30: Hard Drive Usage configuration 2 RAIDx3: double traffic

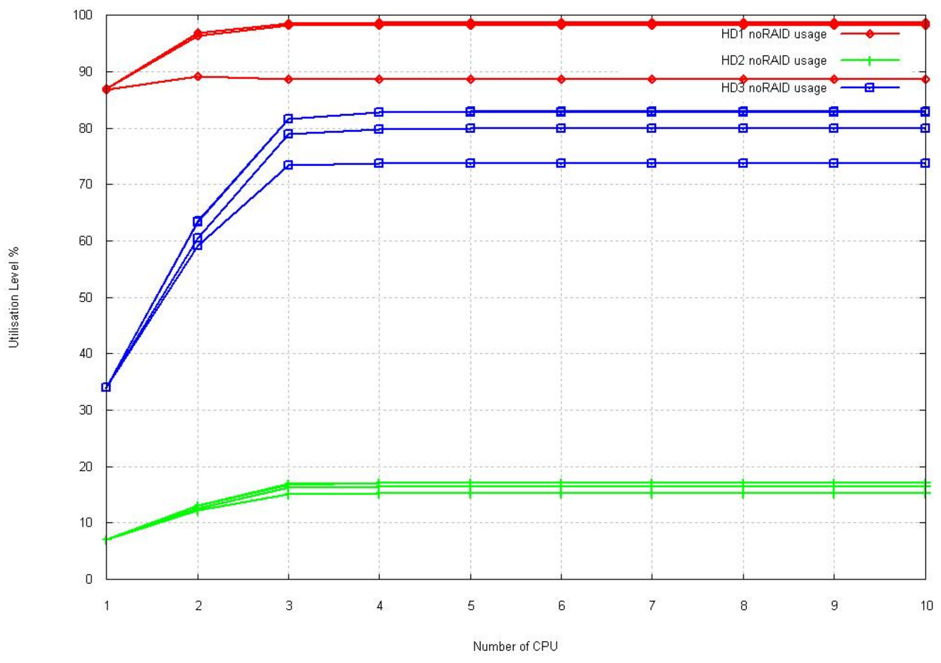


Figure 8.31: Hard Drive Usage configuration 3 noRAID: double traffic

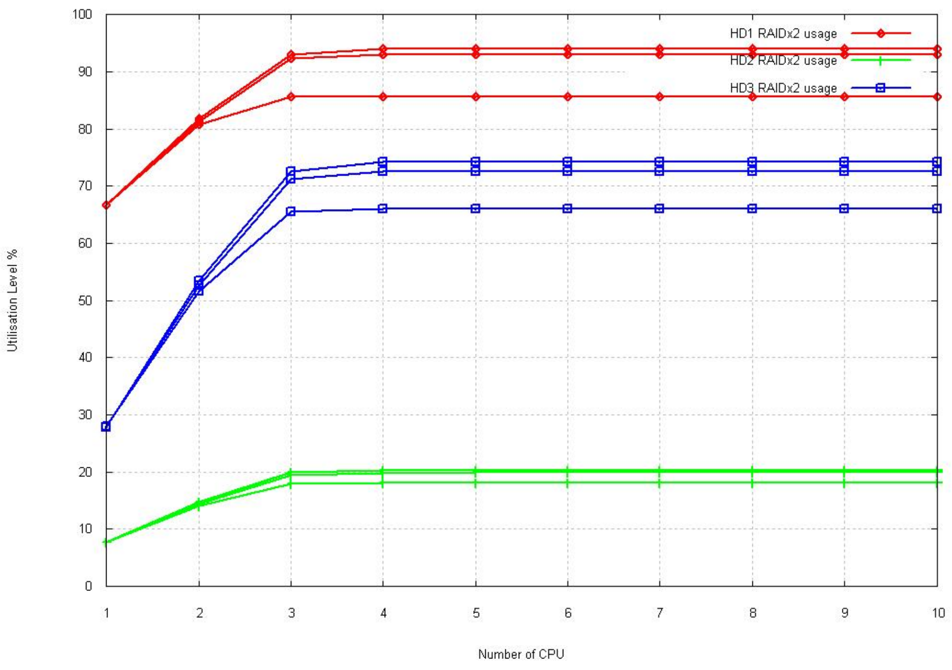


Figure 8.32: Hard Drive Usage configuration 3 RAIDx2: double traffic

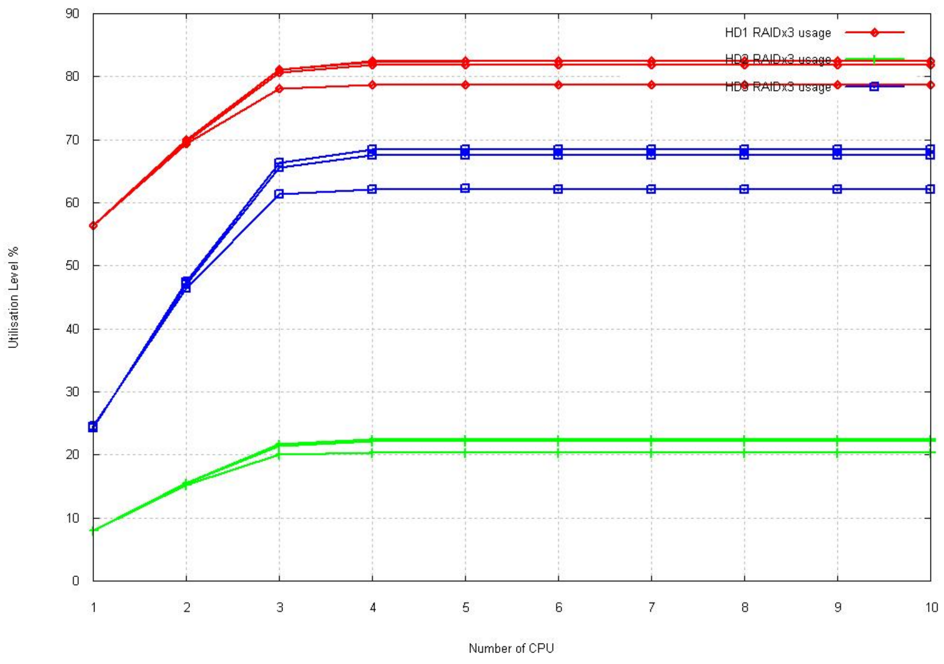


Figure 8.33: Hard Drive Usage configuration 3 RAIDx3: double traffic

hard drive configuration for both configurations. We can see that, again, our limitation resource is the Hard Drive.

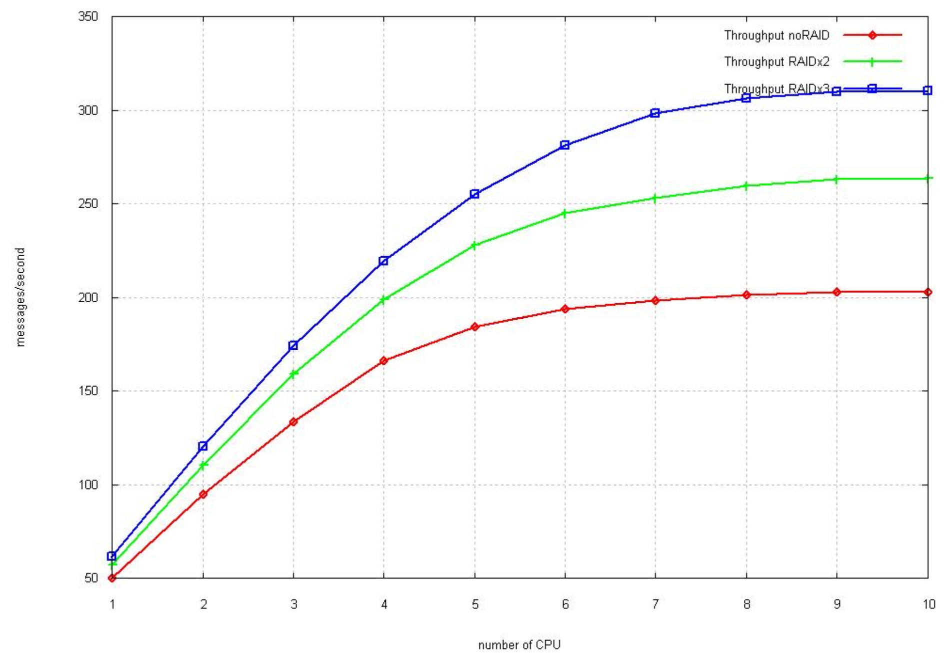


Figure 8.34: Throughput configuration 2: increasing threads

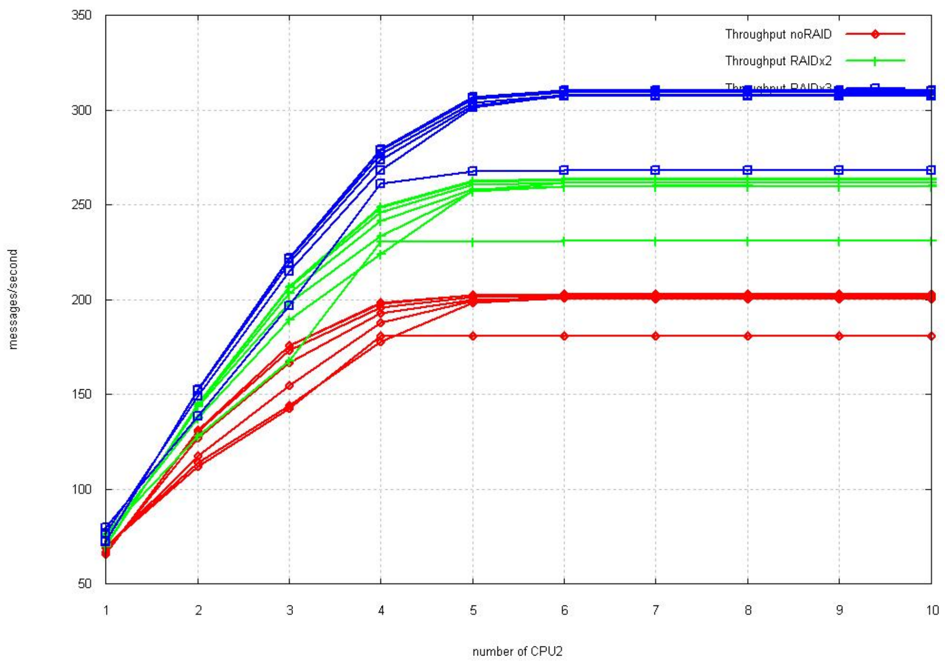


Figure 8.35: Throughput configuration 3: increasing threads

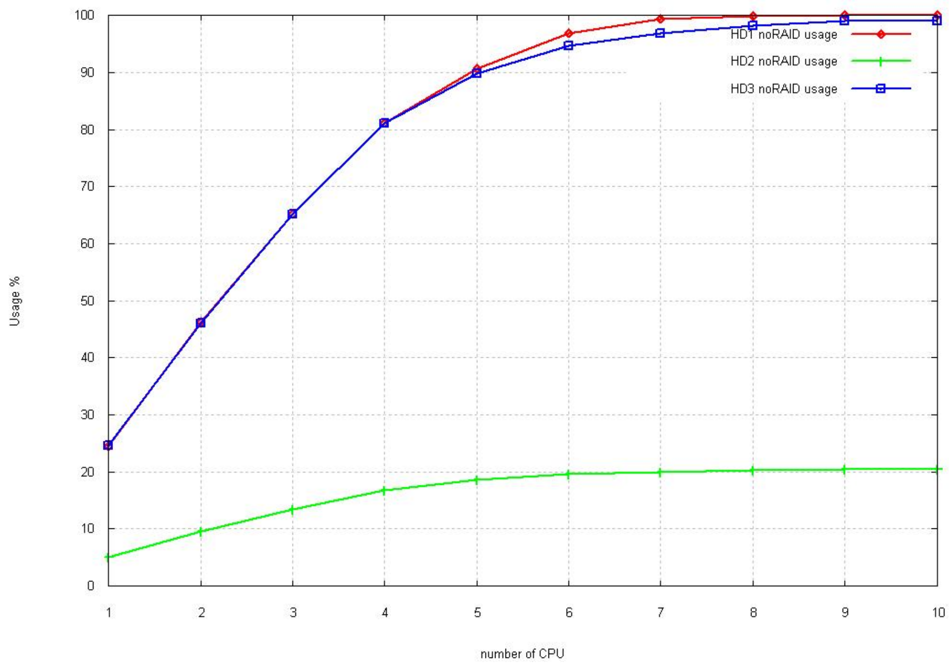


Figure 8.36: Hard Drive Usage configuration 2 noRAID: increasing threads

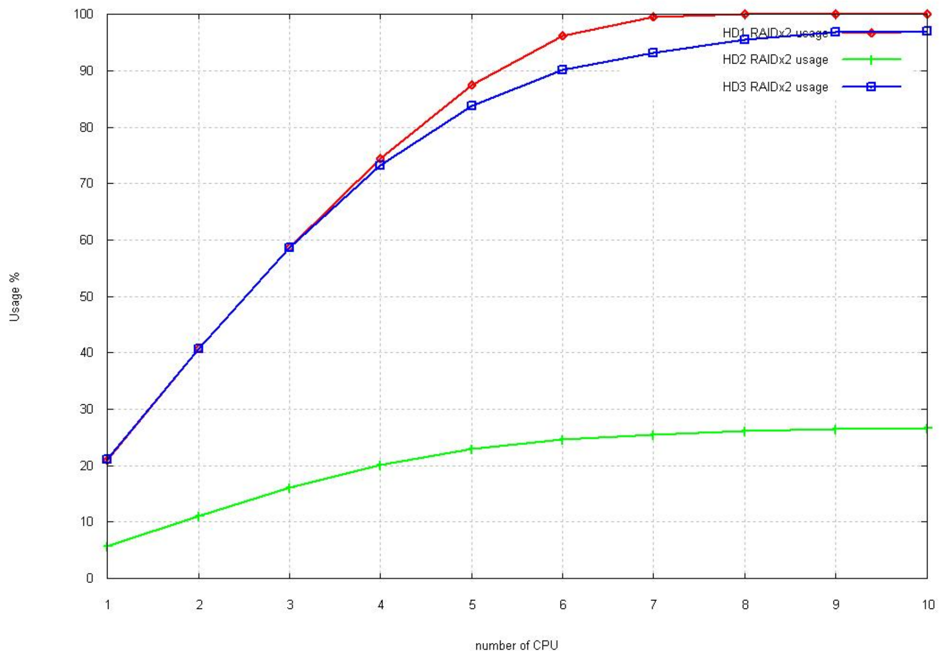


Figure 8.37: Hard Drive Usage configuration 2 RAIDx2: increasing threads

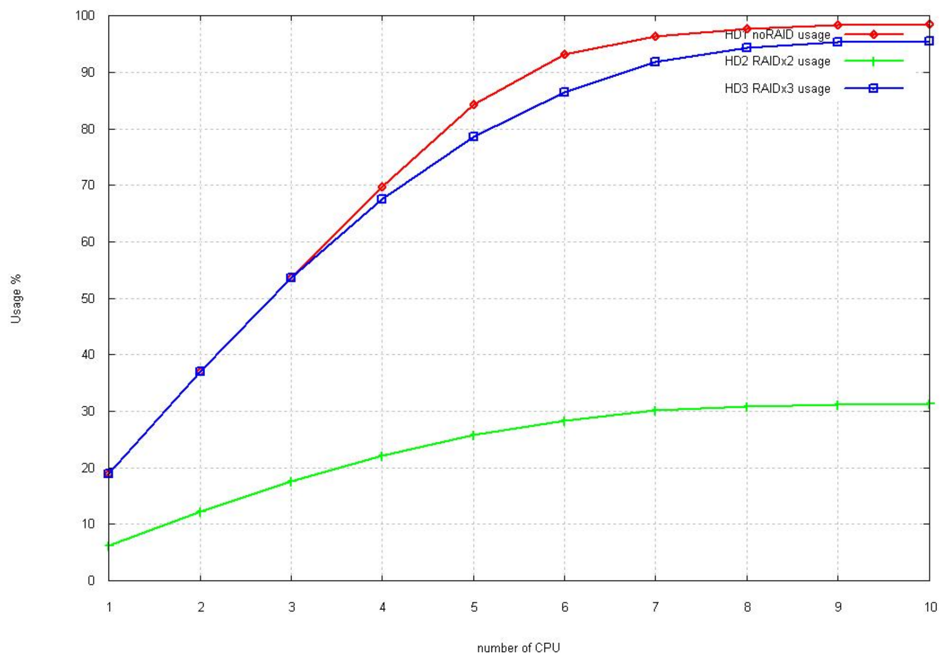


Figure 8.38: Hard Drive Usage configuration 2 RAIDx3: increasing threads

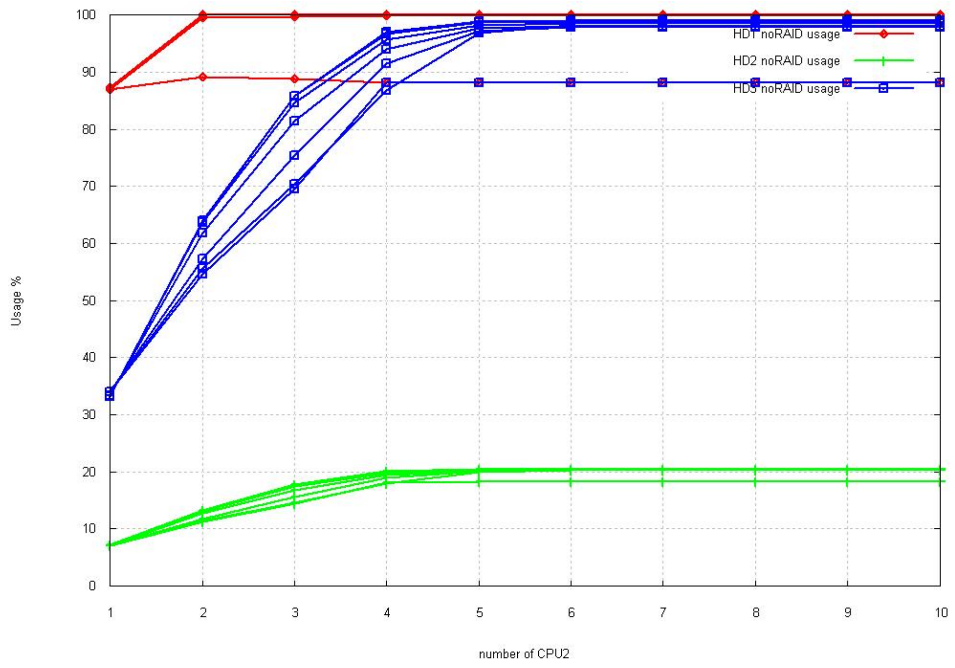


Figure 8.39: Hard Drive Usage configuration 3 noRAID: increasing threads

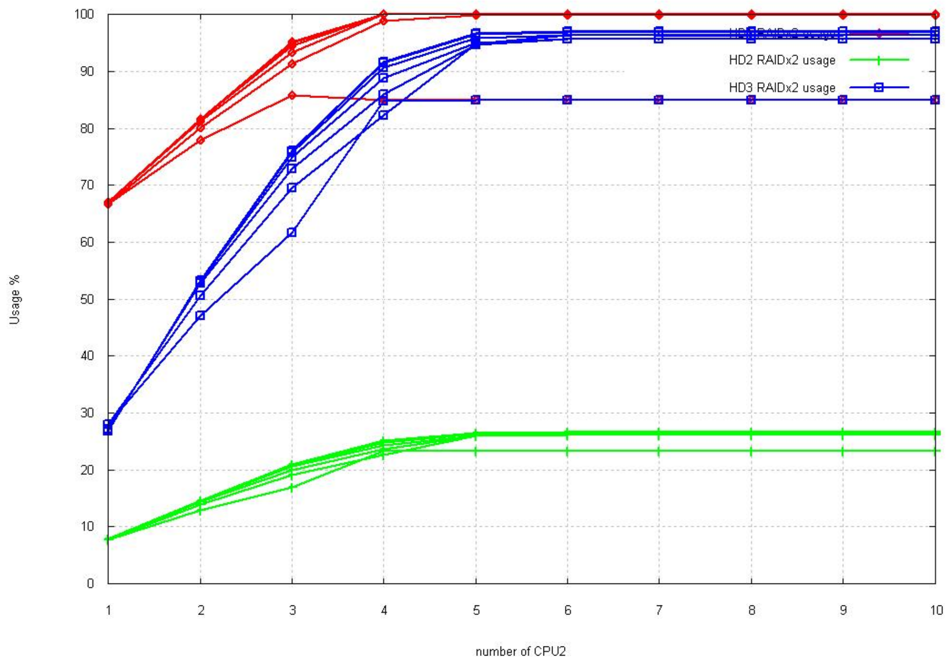


Figure 8.40: Hard Drive Usage configuration 3 RAIDx2: increasing threads

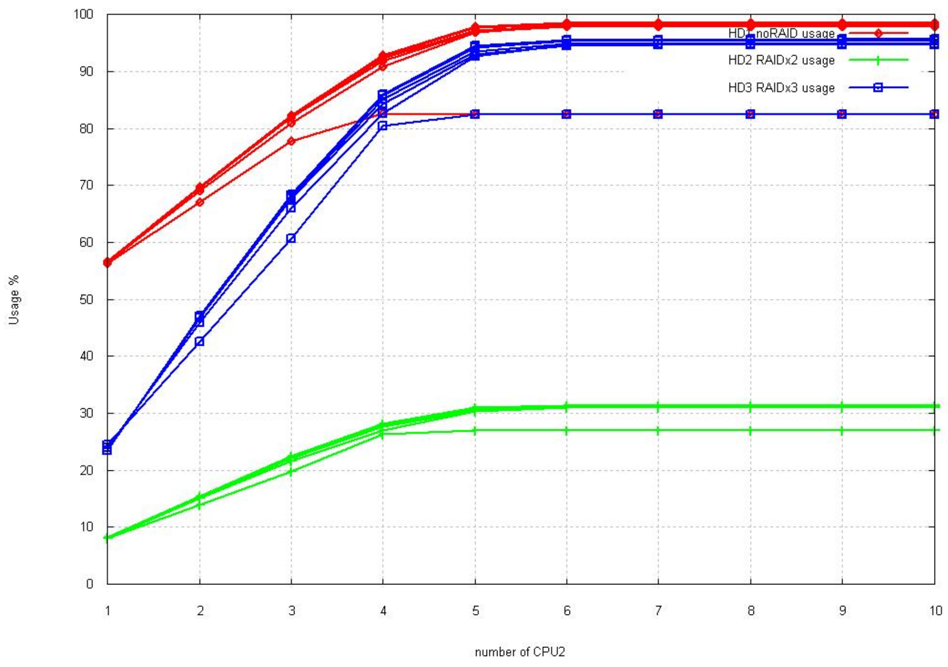


Figure 8.41: Hard Drive Usage configuration 3 RAIDx3: increasing threads

Part IV

Conclusions

Chapter 9

Conclusions

After the analityc and the simulation model we are in a position to point to the weak points of this architechture and give a few possible solutions. We found out that our main bottleneck was the use that this architechture make of the database. We do have 3 databases.The product one is necessary, the other 2: the incoming an outgoing database are open to discussion, from our point of view. To solve this problem, we offer many possible options:

- **Break the write/deletes sequence**: With the hard drive sizes that we have, we do not need to delete the messages after process them. We can delete all the messages at night, for example, as a mainteinance work. This solution has a problem: A table with 100.000 rows will have a select time much bigger than one with 1000 (as more rows, more time will take the operations). So the solution could be rename the table every 1000 messages, for example, and create a new one with the same name.

- **Use multiple incoming databases**: The incoming database do not have a direct relation with the other databases. So, we can have N incoming databases, every one in a different hard drive. To do this we will also need to have different dispatchers, related to the different databases.
  The dispatcher is a message-driven bean, It is binded to a JMS queue. When this queue register an entry, the JMS service will call the message-driven bean binded to this queue.
  We can use the same solution on the outgoing database.
  This modifications will need software modifications: the receiver needs to include an algorith to decide in wich queue (database) send the message, and configuration modifications: we need one Mesasge-Driven bean for every queue, at least. See Figure 9.1 for details.

- **Eliminate the incoming database** This is a more extreme solution.

This solution has more impact than just improving the performance. It also affects security and availability issues. We can change the JMS configuration to base the incoming queue in RAM memory instead of in a database. The I/O operations with RAM memory are much faster than the I/O operations with the hard/drive. Adopting this solution, will make the amount of RAM memory a critycall resource.

Besides all this changes, we also have to think on the database performance. Perhaps PostgreSQL is not the best one, so we recommend to test the architechture with different databases: MySQL, Oracle, SQL Server, ... We can neither forget the database configuration [POS06]. Find the proper database configuration can be a very difficult work.

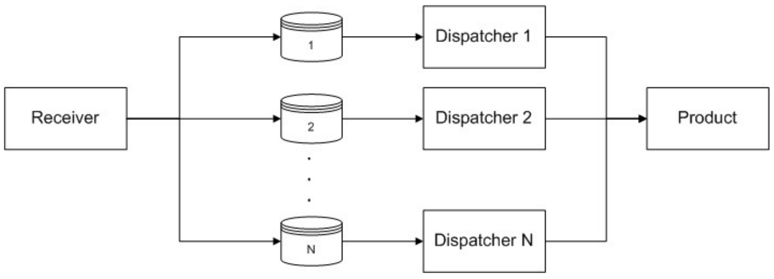


Figure 9.1: Architecture modifications: multiple incoming databases

But the problem do not ends here. If we succeed on eliminating the incoming database bottleneck, we will find ourselves with another one. This time It will be the product database. This problem is far more complicated, since this database is needed for the system to operate, we can not eliminate It. The solution here will be using a clustering or a multiple database solution.

- The clustering solution can be more difficult to implement, from the database point of view, but It will not add extra functions to our system.
- The multiple database solution will add extra funtions to our system. The objective of this functions will be maintain the data integrity and maintain It updated. If we have a user with a pre-paid credit, we need to be able to acces to its current balance from any of the databases.

To be able to choose a solution at this point we need to study the database design, and the use of the system.

- Do the clients have multiple accounts, one for each product? in this case we could try to split the database in different databases, one for each product.

- Is there a client with special needs? If a client is generating the most traffic, we can think in use a database for him.
- In this project we have not thought in the mobile operators performance, can we have this traffic level in one location? or we do have to have many servers distributed in different geographical locations?. This will finally influence in our choice between clustering or multiple databases.

Another possible improvement will be "unbinding" the dispatcher from the product. This way, we will need less dispatcher threads, what means less system resources. The problem with the wrong processed messages can be solved if the product itself, once the process have failed, sends the message to the incoming database again, as if It were new. The reason why the dispatcher and product modules are binded can be the way that they are implemented: the dispatcher is a Message-Driven Bean and the product is a EJB. A possible solution could be create a listener where the dipatcher will send the messages. This way, the dispatcher will not need to wait until the product finishes. The listener will call the proper EJB-product depending on the message type.

Furtherwork

In this project we have looked into Boost Communications architechture. We have build and tested an analytic and simulation model looking for possible bottlenecks or weak points. Finally we have presented possible solutions to the problems.
A future work following this line, should be focused on the possible solutions. Test the availability of the solutions, and its impact on the performance through simulations, and if It is possible making tests on a real environment.

Appendix

.1 Simula Code for Boost Network

```
begin
EXTERNAL CLASS demos=”d:/cim/demos/demos.atr”;
demos
begin

    integer M = 1;
integer N = 1;
integer K = 1;
integer J = 5;

    ref(res) array cpu(1:M);
ref(res) array HD(1:N);
ref(res) array Threads(1:J);

    ref(rdist) lambda,receiverT,dispatcherT,selectT,HDaccesT;
ref(rdist) HDwriteT,HDdeleteT,senderT,queueT,getConnectionT,productT;
ref(bdist) cache;
ref(tally) incoming,dispatch,sender,total,product,average,Qsender;
ref(count) quant;

    integer i;
integer semilla,cpus;

    entity class Generator;
begin
loop:
hold(lambda.sample);
NEW IncomingMessage(”IP”,time).schedule(0.0);
repeat;
end;

    entity class IncomingMessage(creationTime);
```

```
real creationTime;
begin
real receiverTime;
real dispProdTime;
real productTime;

    Threads(1).acquire(1);
receiverTime := time;
receiver:
cpu(1).acquire(1);
hold(Abs(receiverT.sample));
cpu(1).release(1);
hold(0.0);
getConnection1:
cpu(1).acquire(1);
hold(Abs(getConnectionT.sample));
cpu(1).release(1);
hold(0.0);
write:
cpu(1).acquire(1);
HD(1).acquire(1);
hold(Abs(HDaccesT.sample)+Abs(HDwriteT.sample));
HD(1).release(1);
cpu(1).release(1);
Threads(1).release(1);
incoming.update(time-receiverTime);
hold(0.0);

    Threads(2).acquire(1);
dispProdTime := time;
dispatcher:
cpu(1).acquire(1);
hold(Abs(dispatcherT.sample));
cpu(1).release(1);
hold(0.0);
getConnection2:
cpu(1).acquire(1);
hold(Abs(getConnectionT.sample));
cpu(1).release(1);
hold(0.0);
select:
cpu(1).acquire(1);
if cache.sample then
begin
```

```
HD(1).acquire(1);
hold(Abs(selectT.sample+HDaccesT.sample));
HD(1).release(1);
end
else
hold(Abs(selectT.sample));
cpu(1).release(1);
hold(0.0);

    Threads(3).acquire(1);
productTime:=time;
productProcessor:
cpu(1).acquire(1);
hold(Abs(productT.sample));
cpu(1).release(1);
hold(0.0);

    productDataBase:
cpu(1).acquire(1);
HD(1).acquire(1);
hold(Abs(HDaccesT.sample)+Abs(selectT.sample)+Abs(HDwriteT.sample));
HD(1).release(1);
cpu(1).release(1);
hold(0.0);
cpu(1).acquire(1);
HD(1).acquire(1);
hold(Abs(HDaccesT.sample)+Abs(HDwriteT.sample));
HD(1).release(1);
cpu(1).release(1);
Threads(3).release(1);
product.update(time-productTime);
hold(0.0);

    NEW OutgoingMessage("OUT",time,creationTime).schedule(0.0);

    delete:
cpu(1).acquire(1);
HD(1).acquire(1);
hold(Abs(HDaccesT.sample)+Abs(HDdeleteT.sample));
HD(1).release(1);
cpu(1).release(1);
Threads(2).release(1);
dispatch.update(time-dispProdTime);
hold(0.0);
```

```
end;

    entity class OutgoingMessage(creationTime,totalTime);
real creationTime,totalTime;
begin
real senderTime,queueTime;

    Threads(4).acquire(1);
senderTime:=time;
QueueSender:
cpu(1).acquire(1);
hold(Abs(queueT.sample));
cpu(1).release(1);
hold(0.0);
getConnection:
cpu(1).acquire(1);
hold(Abs(getConnectionT.sample));
cpu(1).release(1);
hold(0.0);
select:
cpu(1).acquire(1);
if cache.sample then
begin
HD(1).acquire(1);
hold(Abs(selectT.sample+HDaccesT.sample));
HD(1).release(1);
end
else
hold(Abs(selectT.sample));
cpu(1).release(1);
hold(0.0);

    Threads(5).acquire(1);
queueTime:=time;
send:
cpu(1).acquire(1);
hold(Abs(senderT.sample));
cpu(1).release(1);
total.update(time-totalTime);
Threads(5).release(1);
quant.update(1);
sender.update(time-queueTime);
hold(0.0);
```

```
    delete:
cpu(1).acquire(1);
HD(1).acquire(1);
hold(Abs(HDaccesT.sample)+Abs(HDdeleteT.sample));
HD(1).release(1);
cpu(1).release(1);
Threads(4).release(1);
Qsender.update(time-senderTime);
hold(0.0);
end;

    cpu(1) :- New res("CPU1",1);

    for i:= 1 step 1 until N do
HD(i) :- New res(edit("HD",i),1);

    Threads(1) :- New res("Thread1",1);
Threads(2) :- New res("Thread2",3);
Threads(3) :- New res("Thread3",3);
Threads(4) :- New res("Thread4",3);
Threads(5) :- New res("Thread5",1);

    incoming :- New tally("ReceiverT");
dispatch :- New tally("DispatcherT");
product :- New tally("ProductT");
Qsender :- New tally("QSendT");
total :- New tally("TotalT");
sender :- New tally("SenderT");
average :- New tally("AverageT");

    quant :- New count("quant");

    semilla:=sysin.inint;
setseed(semilla);

    lambda :- NEW NegExp("lambda",1/6);
receiverT :- NEW normal("receiver",0.1,0.01);
dispatcherT :- NEW normal("dispatcher",0.2,0.02);
productT :- NEW normal("product",7.5,0.75);
queueT :- NEW normal("queueSender",0.2,0.02);
senderT :- NEW normal("Sender",0.1,0.01);
getConnectionT :- NEW normal("getConnection",0.1,0.05);
selectT :- NEW normal("select",0.5,0.25);
HDaccesT :- NEW normal("acces",0.75,0.1);
```

```
HDwriteT :- NEW normal(”write”,1.25,0.5);
HDdeleteT :- NEW normal(”delete”,1.75,0.75);

   cache :- NEW draw(”cache”,0.3);

   NEW Generator(”GenIP”).schedule(0.0);

   hold(5000);
reset;
i:= time;
hold(200000);
average.update((time-i)/quant.obs);

   end demos;
end;
```

.2 sample SIMULA Result

```
                       CLOCK TIME = 2.050\&+005
**********************************************************************
*                                                                    *
*                            R E P O R T                             *
*                                                                    *
**********************************************************************



                      D I S T R I B U T I O N S
                      *************************

TITLE       /   (RE)SET/   OBS/TYPE     /        A/        B/     SEED
lambda         5000.000  33373 NEGEXP        0.167                 36855
receiver       5000.000   9357 NORMAL        0.1001.000&-002  16300085
dispatcher     5000.000   9357 NORMAL        0.2002.000&-002  64487931
product        5000.000   9357 NORMAL        7.500     0.750  36742265
queueSender    5000.000   9357 NORMAL        0.2002.000&-002    218568
Sender         5000.000   9357 NORMAL        0.1001.000&-002   3378000
getConnectio   5000.000  28071 NORMAL        0.1005.000&-002  26530315
select         5000.000  28073 NORMAL        0.500     0.250    160441
acces          5000.000  52450 NORMAL        0.750     0.100   8292919
write          5000.000  28072 NORMAL        1.250     0.500  64192707
```

```
delete          5000.000  18714 NORMAL        1.750      0.750  26388359
cache           5000.000  18716 DRAW          0.300               2136727


                             C O U N T S
                             ***********

                    TITLE       /   (RE)SET/   OBS
                    quant           5000.000   9357


                            T A L L I E S
                            *************

TITLE       /   (RE)SET/   OBS/  AVERAGE/EST.ST.DV/ STD.ERR./  MINIMUM/  MAXIMUM
ReceiverT       5000.000   9358    21.374     1.6751.731&-002    15.446    27.742
DispatcherT     5000.000   9357    56.520     2.7292.821&-002    47.709    66.917
ProductT        5000.000   9358    23.377     1.7471.805&-002    17.372    29.963
QSendT          5000.000   9357    38.010     2.2202.295&-002    29.034    45.980
TotalT          5000.000   9357 75017.958 41530.736   429.340  3606.7211.473&+005
SenderT         5000.000   9357    10.606     1.0591.095&-002     6.371    14.602
AverageT        5000.000      1    21.374--------------------    21.374    21.374


                          R E S O U R C E S
                          *****************

TITLE       /   (RE)SET/   OBS/ LIM/ MIN/ NOW/  \% USAGE/ AV. WAIT/QMAX
CPU1            5000.000 140358    1    0    0  100.000     6.302    5
HD 1            5000.000  52451    1    0    1   57.379     0.000    1
Thread1         5000.000   9358    1    0    0  100.000 74987.56324591
Thread2         5000.000   9357    3    0    0   88.145     0.000    1
Thread3         5000.000   9358    3    1    2   36.457     0.000    1
Thread4         5000.000   9357    3    1    1   59.278     0.000    1
Thread5         5000.000   9357    1    0    1   49.618     0.000    1
```

Bibliography

[Bon00] Andre; B. Bondi, *Characteristics of scalability and their impact on performance*, WOSP '00: Proceedings of the 2nd international workshop on Software and performance (New York, NY, USA), ACM Press, 2000, pp. 195–203.

[DEM06a] *Demos reference manual*, `http://www.iro.umontreal.ca/~vaucher/DEMOS/Demos.html`, July 2006.

[DEM06b] *Introduction to demos and discrete events simulation*, `http://www.dcs.shef.ac.uk/~graham/research/`, July 2006.

[FOL06] *Free on-line dictionary of computing*, `http://www.foldoc.org`, July 2006.

[Hil90] Mark D. Hill, *What is scalability?*, ACM SIGARCH Computer Architecture News **18** (1990), no. 4, 18–21.

[Jac57] J.R. Jackson, *Networks of waiting lines*, Operations Research (1957), no. 5, 518–521.

[Kle57] Leonard Kleinrock, *Queueing systems, volume i: Theory*, Wiley-Interscience, 1957.

[Luk94] E. Luke, *Defining and measuring scalability*, 1994.

[POS06] *Postgre tuning configuration*, `http://www.varlena.com/GeneralBits/Tidbits/perf.html#basic`, July 2006.

[Sar99] Robert G. Sargent, *Validation and verification of simulation models*, WSC '99: Proceedings of the 31st conference on Winter simulation (New York, NY, USA), ACM Press, 1999, pp. 39–48.

[SIM06a] *Compiling simula*, `http://www.ifi.uio.no/~cim/sim_history.html`, July 2006.

[SIM06b] *Montreal simula site*, `http://www.iro.umontreal.ca/~simula/`, July 2006.