POLITECNICO DI MILANO

FACOLTA DI INGEGNERIA DELL'INFORMAZIONE

MASTER OF SCIENCE IN ENGINEERING OF COMPUTING SYSTEMS

OPTIMAL LOAD ALLOCATION ALGORITHM FOR LARGE DISTRIBUTED SYSTEMS

Supervisor: Prof. Giuseppe SERAZZI

Co-Supervisor: Prof. Marco GRIBAUDO

Tesi di Laurea di .

Ashanka DAS, Matricola: 737135

ANNO ACCADEMICO 2010

Abstract

This master thesis presents, a family of algorithms for load allocation in a large scale computing infrastructures, using the concept of resource saturation in a multi-class environment.

New request admission policies and routing algorithms have been developed to improve the performance metrics based on the concept of bottleneck in a network. The underlying idea is to selectively admit in the system a job of class that maximize the utilization of the resources unused by the job in execution.

The algorithm have been coded and simulated, in JMT, 'Java Modelling Tool' a suite of applications developed by Politecnico di Milano and released under GPL license, to evaluate the gain achieved when the new admission policy is used.

For modeling a large computing network, the technique of Queuing Network are used where a Queuing Station represents any given Computing Resource with a given service time, capable of serving user requests. Several simulation experiments were performed on the above model for proper calibration of the admission policy to perform at optimum level.

Results are in agreement with the theoretical predictions and show a significant improvement over prevailing static routing algorithms. The work presented also has several implications for future studies of Green IT since it can also help solving the problem of energy consumption in large computing centers.

Keywords(Routing Algorithm)(Tuning)(Load balancing)(Provisioning)(Optimal resource allocation)

Sommario

La tesi presenta un nuovo algoritmo per la ripartizione del carico in sistemi o reti di sistemi informatici avanti dimensioni molto elevate. Il concetto di base quello di valuatare i livelli di contesa delle risorse ed in base alla.

Gli algorithmi sono stati codificati e simulati, in JMT, Java Modelling Tool, una suite di applicazioni sviluppato dal Politecnico di Milano e rilasciata sotto licenza GPL, per vantaggi, ottenuti, attraverso la nuova politica di ammissione.

I risultati sono ulteriormente discussi nella tesi. I risultati mostrano un accordo parziale con i risultati teorici, ed un miglioramento significativo rispetto agli algoritmi di routing statico. Il lavoro qui presentato ha anche profonde implicazioni per studi futuri di Green IT e possono anche contribuire ad affrontare il problema del consumo di energia in grandi centri di calcolo.

Acknowledgements

To Prof. Serazzi for his immense support, encouragement and believe in me, throughout the thesis and my masters at Politecnico. Prof. Marco Gribaudo for his valuable time, comments and ideas.

My family and friends who have helped me and supported me in this experience.

Contents

List of Tables

List of Figures

Chapter 1

Introduction.

1.1 Background

In the modern world, the amount of computing resources available to users are limitless. Computers are becoming large, distributed in nature with huge amount of storage and computing power. Such large number of components, interconnected with each other through intranet or internet and running multiple layer, increases the complexity of system architecture. With many applications running on the same hardware through virtualization, the challenges to manage them effectively and critical issues related to daily operation, increases.
In this complex scenario, a system must be able to

1. self configure

2. self optimize

3. self recover

based on the evaluation of the actual network conditions or state of the system. The dimensions of the solution space are huge and possibly hundreds of thousands of different configurations have to be evaluated in order to take the best possible decision. Modern data centers has to be designed to be scalable and adaptive.
Cloud computing [2], is one of the revolutionary approach for very large data center or computing center, with infinite scalability and tremendous computing power. The designers of such large scale computer infrastructures face a new family of problems related to the ever increasing complexity of the architectures. The large number of components and interconnected networks, the heterogeneity of the applications, the service levels required, the intensity and the unpredictable fluctuations of the traffic, and the multi-layered architecture are some of the critical issues that must be addressed.

In a very large system, many applications can be deployed and run at the same time. The advantage gained is that, many customers can simultaneously use the same underlying physical hardware. This avoids the need to own their own physical infrastructure by renting and sharing resources. This allows the users to reuse initial costs on hardware, software and services by using the idea of pay on use basis. They consume resources as a service and pay only for what they use. As shown in Figure 1.1. Consumption is billed on a utility basis(i.e. resources consumed) or subscription basis (time-based) with little or no upfront cost.
Other benefits of this approach are low barriers to entry, shared infrastructure and low management overhead, and immediate access to a broad range of applications. In general, users can terminate the contract at any time (thereby avoiding return on

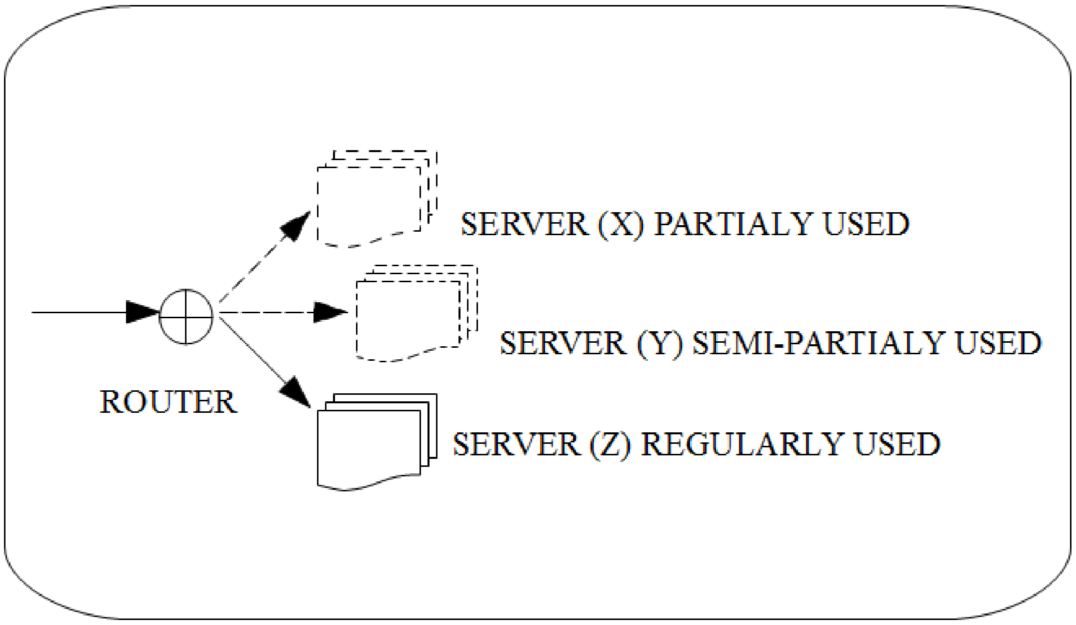


Figure 1.1: Dynamic scaling of very large systems

investment risk and uncertainty), and the services are often covered by service level agreements (SLAs) with financial penalties.

A Service Level Agreement (frequently abbreviated as SLA) is a part of a service contract where the level of service is formally defined. In practice, the term SLA is sometimes used to refer to the contracted delivery time of the service or performance. See figure 1.2. A service level agreement (SLA) is a negotiated agreement between two parties where one is the customer and the other is the service provider. The SLA may specify the levels of availability, service ability, performance, operation, or other attributes of the service.

A possible topic of discusion and research is negotiating an unique SLA between the service provider and the customers, which benefits both, for e.g. customers pay reasonable price for the usage of the resource and service provider is able to efficiently manage the resources. Studies need to be undertaken to know the performance associated with the usage of network resources and workload charecterization to gain insight into parameters which can fluctuate the optimality of the given hardware infrastructure.

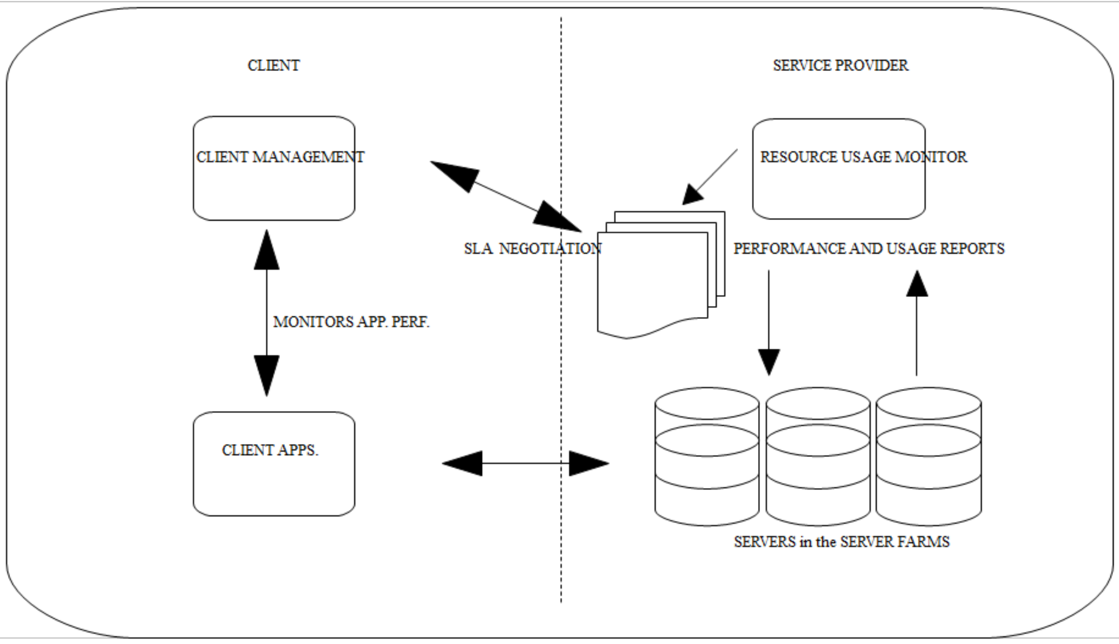


Figure 1.2: SLA negotiation based on usage and performance.

1.2 Motivation

The increasing range of Internet applications and the corresponding growth of user base have led to the need of large, reliable, scalable data centers having hierarchical tier based architecture. Figure 1.3 depicts the Internet as a cloud in network diagrams.

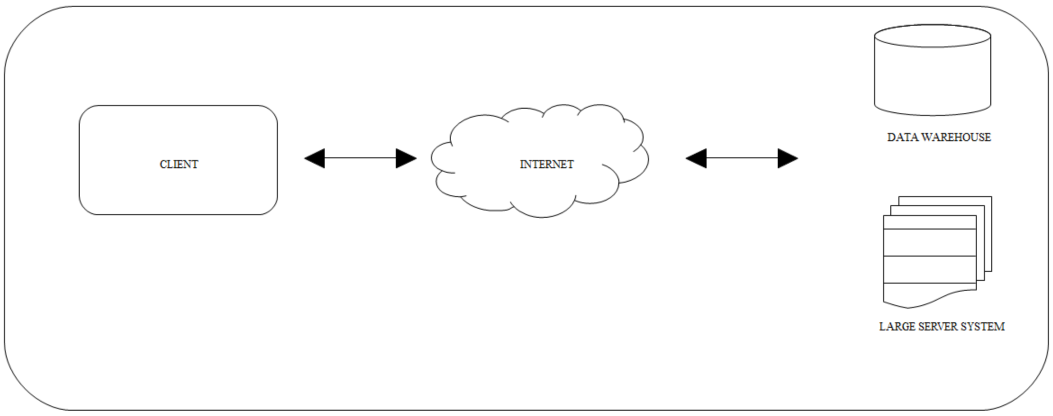


Figure 1.3: Modern Internet

Such modern data centers are typically organized in building blocks, hosting hundreds or thousands of servers tightly packed in racks. In such scenario, the existing infrastructure are enriched with run-time management mechanisms and algorithms that share multiple goals, for e.g., to optimize the run and deployment of applications over the servers, to reduce power consumption, to avoid performance degradation.
In such large data centers normally, algorithms are operating at a coarse-grained time scale which run periodically, to determine which servers must be turned on or off and how applications and services are to be deployed over the servers. Special attention has to be given to sudden and unpredictable peaks in the user demand or work load. These mechanism is not sufficient in dealing with the rapidly changing service access patterns and can not be monitored efficiently with coarse-grained decisions.
Over provisioning of the hardware infrastructure may represent one of the solution

in the short term, but it introduces additional power consumption costs which are economically poor decisions. Load management algorithms are needed to guarantee good performance despite the foreseeable changes in request patterns. Hence, the strategies operating at a coarse grained time scale must be enriched with real-time solutions for handling load management. This motivates to investigate innovative methods based on real-time state of system, which can efficiently deal with varying server load, optimize the resources consumed and keep the performance of the system under given quality and standard.

1.3 Aim

The focus of this masters thesis is to address the performance issues of the above mentioned large server system without compromising on an agreed minimum Quality of service. The aim is to design efficient techniques based on the knowledge of work-load and their heterogeneous demand on resource consumption, in order to optimize the utilization, throughput and response time.

As the workload is highly dynamic and characterized by the nature of high unpredictability, to design such systems with high performance, they have to be designed as autonomic in nature. An Autonomic computing systems perform self- management including ability to optimize the use of resources, self-tuning and self-reconfiguration based on the current state of the machine. To achieve the above mentioned goals, concepts of *bottleneck migration* and *common saturation sector* in a network (theory to be introduced later in chapter2) has been used to build a new approach to optimize a given large multiclass server system. The objective is to design a *Admission control* *policy*, *Redirection policy* and *Routing policy* for a given computer network having contention of resources to maximize its performance.

Study aims to understand to which extent and under which conditions the integration of predictive models into existing request management solutions can improve the performance of the system. We focus initially on an algorithms that selectively allow the admission of user request based on the class and then focus redirection or routing of user requests among the servers hosted in the data center. The decisions about user request admission and request redirection are local in nature. An admission control algorithm also decides whether a request should be processed in the given node or should be dropped into another node. In the latter case, it also chooses the most appropriate node. Both decisions are usually enforced through the evaluation of the load conditions of the server and, of its immediate neighbors, and through the comparison of these values against a fixed, static threshold.

Effort is given to model the infrastructure through queuing theory specifically M/M/1 or M/G/1 queues and infer the server performance from the mix of classes residing in the service section. The approach using Queueing theory also aims to overcome the complexity of architecture of such large server systems by assuming certain assumptions and obtain results which gives a fair evaluation. To address this issue, we use the server queue length and number of requests getting serviced as a measure of server load. Process queue lengths allow to define a linear relationship between server load and the predicted user request response time, thus simplifying the prediction model. In order to understand the gain achieved we evaluate the performance associated by the redirection or controlled admission of a user request by comparing the throughput obtained with the value obtained by random routing and default admission pol-

icy(First come First). Our study also aims to understand to which extent and under which conditions the integration of predictive models into existing request management solutions can improve the performance of the system.
Aim is to study the results with the help of a detailed system simulator, and demonstrate that the proposed solution outperforms existing, static threshold-based request management strategies. We also strive to show how prediction can help us to achieve results that are stable across different workload scenarios and how it can preserve good performance for different request redirection.

1.4 Thesis Outline

This Thesis document is organized as follows:

- Chapter 2, presents the Theoretical synthesis behind the developed techniques, an introduction about Queueing Networks and concepts on Bottlenecks and Saturation Sector which has mainly motivated this research.
- Chapter 3, presents the techniques of converting the large data centers as Queueing Network models.
- Chapter 4, introduces the techniques and simulation tools used for the development and modeling of the work.
- Chapter 5, discusses the proposed request admission and redirection algorithms.
- Chapter 6, present the experimental setup and the results of the experimental evaluation carried out for different workload scenarios.

- Chapter 7, concludes the thesis with and future work.

Chapter 2

Theoretical Synthesis

2.1 Multi-Class Queueing Network.

Queueing theory is the mathematical study of waiting lines or queues. The theory enables mathematical analysis of several related processes, including arriving at the (back of the) queue, waiting in the queue and being served at the front of the queue. The theory permits the derivation and calculation of several performance measures including the average waiting time in the queue or the system, the expected number waiting or receiving service, and the probability of encountering the system in certain states, such as empty, full, having an available server or having to wait a certain time to be served.

Queueing network modelling, is a particular approach to computer system modelling in which the computer system is represented as a network of queues which is evaluated analytically. A network of queues is a collection of service centers, which represent system resources, and customers, which represent users or transactions.

Multi class queueing networks are used to model large computing networks because they can provide an good description of the system load, which is reflected in more accurate results of performance analysis [3]. Computers are becoming large, distributed in nature and have huge amount of storage and computing power. Such huge number of components, which are interconnected to each other through intranet or internet and running N-tier architectures, also increases the complexity of the model of the system. Also with many applications running on the same system through virtualization, challenges and critical issues related to modeling and designing increase manifold times.

The complexity of modern computer infrastructures makes the application of known solution techniques infeasible for these new environments. For example the exact solution techniques like the convolution algorithm [4] and the MVA [15], are prohibitively expensive in term of computational resource requirements. On the other side, approximate solution techniques (see e.g., [7, 9]) may suffer an uncontrolled decrease in accuracy as the complexity of the model grows [19]. Thus, with computer systems that comprise hundreds of servers, LANs and WANs and workloads consisting of large populations of several heterogeneous customer classes an interesting alternative (in some cases the only one feasible) to both exact and approximate solutions is represented by asymptotic techniques.With a limited effort it is possible to determine asymptotic values of several performance indices such as system response time and throughput, resources utilizations and queue lengths.

2.1.1 Bottlenecks

The performance of any system is limited by the congested hardware resources, also known as bottlenecks. The asymptotic analysis of multi-class queueing networks with distinct bottlenecks have introduced us to concepts of bottleneck migration. A bottleneck is a phenomenon where the performance or capacity of an entire system is limited by a single or limited number of components or resources.

As shown in [3] that multi-class models can exhibit multiple simultaneous bottlenecks. The dependency of the bottleneck set on the workload mix is therefore derived. In an enterprise system there are normally different classes of jobs and the class mix can change at run-time. This suggests that there might be several bottlenecks at the same time and bottlenecks can shift from one server to another server over time. The concepts derived in the [3] will revisited in the following section of Optimal mixes which will guide us through the mathematical derivation of common saturation sector.

2.1.2 Optimal Mixes

Interesting properties of two-class queueing networks are investigated. The theory presented is valid for two-station networks only, while others can be extended to models with a higher number of stations and/or a higher number of classes. In a general setting, we consider that each customer class has a favorite station which it utilizes the most. Such station is different for each class. For this type of workloads, we show that in a two-station network there exists a customer population in which the proportion of components of each class is such that both stations are equally utilized for all population sizes. It is shown that the equiutilization population provides an optimal operational point of the system where the system power (or, equivalently, the sum of the station utilizations) is maximized.

This means that the system operates at the best performance since the global utilization of the stations is maximum and so is the ratio of throughput to response time. Furthermore, the equiutilization population belongsto a set of populations, called common saturation sector, such that both stations saturate when the population size increases.

2.1.3 The Equiutilization Point

The contents of this section has been described in details in [3]. Let **N** be a closed queueing network with two fixed rate single server stations (denoted by indices `i` and `j`), two classes of customers (denoted by indices `r` and `s`), and total number of customers **K**. Let $\underline{\beta} = (\beta_1,\beta_2)$ be the population mix, where $\beta_r = K_r$ / **K** and K_r is the total number of class r customers in the network (thus, $K_1 + K_2 =$ **K** and $\beta_1 + \beta_2 = 1$). Therefore, the network population is given by $\underline{\mathrm{K}} = (\beta_1$ **K**, β_2 **K**). Let $L_{ir} = V_{ir}\ S_{ir}$ be class `r` loading of station i, where V_{ir} is the average number of visits of class `r` customers at station `i` and S_{ir} is the average service time of class `r` customers at station `i`. Thus, L_{ir} is the total service demand of a class `r` customer on station `i`. As an example, consider the following loading matrix

$$L_{ir} = \begin{bmatrix} 60 & 40 \\ 30 & 70 \end{bmatrix} \tag{2.1}$$

According to matrix 2.1, class 1 service demands on stations 1 and 2 are 60 and 30 time units, respectively. Class 2 service demands on stations 1 and 2 are

40 and 70 time units, respectively. 2.1 describes one of the load of a model that will be used as a running case in the thesis to illustrate the results presented. In a network **N** there exists a population mix, i.e., a vector β^*, such that both stations are equally utilized for any number **K** of customers in the network. Such a mix is called equiutilizationpoint. The equiutilization point identifies the population mix β^* = (β_1^*, β_2^*) such that

$$U_1(K^*) = U_2(K^*) \tag{2.2}$$

for all K, where K* = $(\beta_1^*$ K,β_2^* K). Station i's utilization can be composed from the steady state as probability `p(k)`.

$$U_i(K) = 1 - \sum_{k:k_i=0} p(k) = 1 - \sum_{k:k_i=0} \frac{1}{G(K)} \prod_{j=1}^{2} F_j(k) \tag{2.3}$$

where k = (k_1 , k_2) is the state vector that describes the population in the network, k_i = (k_{i1}, k_{i2}) is the vector describing the population at station i, and G(K) is the normalization constant of the product-form solution. The functions `F` in Eq. 2.3 can be written as

$$F_i(k) = \prod_{r=1}^{2} L_{ir}^{k_{ir}} = L_{i1}^{k_{i1}} L_{i2}^{k_{i2}} \tag{2.4}$$

as we consider only fixed rate stations. By substituting Eqs.2.3 and 2.4 in Eq.2.2, we obtain

$$L_{21}^{\beta_1 K} L_{22}^{\beta_2 K} = L_{11}^{\beta_1 K} L_{12}^{\beta_2 K} \tag{2.5}$$

which yields the coordinates of the equiutilization point:

$$\beta^* = \left(\beta_1^* = \frac{log\frac{L_{22}}{L_{12}}}{log\frac{L_{11}L_{22}}{L_{12}L_{21}}}, \beta_2^* = 1 - \beta_1^* \right) \tag{2.6}$$

The derivation above applies also to networks with any number of customer classes. As the number R of classes increases, the number of dimensions of the population space increases equally. In general, in an R-dimensional space the equiutilization point is a hyperplane.

The following loading matrix has been used,

$$L_{ir} = \begin{bmatrix} 60 & 40 \\ 30 & 70 \end{bmatrix} \tag{2.7}$$

The theoretical calculation shows us that Equiutilization in such a given network is (0.44, 0.56), i.e. in a given network when class1 is 44% and class2 is 56%, the system will perform at optimum level. To validate the theory, the graph was plotted with various mix of class1 and class2 jobs is a network and corresponding system throughput and system response time was plotted. In the figure 2.1 represents the system throughput for various population mix in execution, which clearly shows that when population mix is 0.40-0.50, the system throughput is maximum. In the figure 2.1 represents the system response time for various population mix in execution and as seen that when population mix is 0.40-0.50, the system response time is minimum.

The figures 2.4 and 2.3 show us that the system throughput and system response time for each class switches at the equiutilization point and the global throughput and response time obtained is optimum.

The properties of multi-class product-form closed queueing networks with two classes

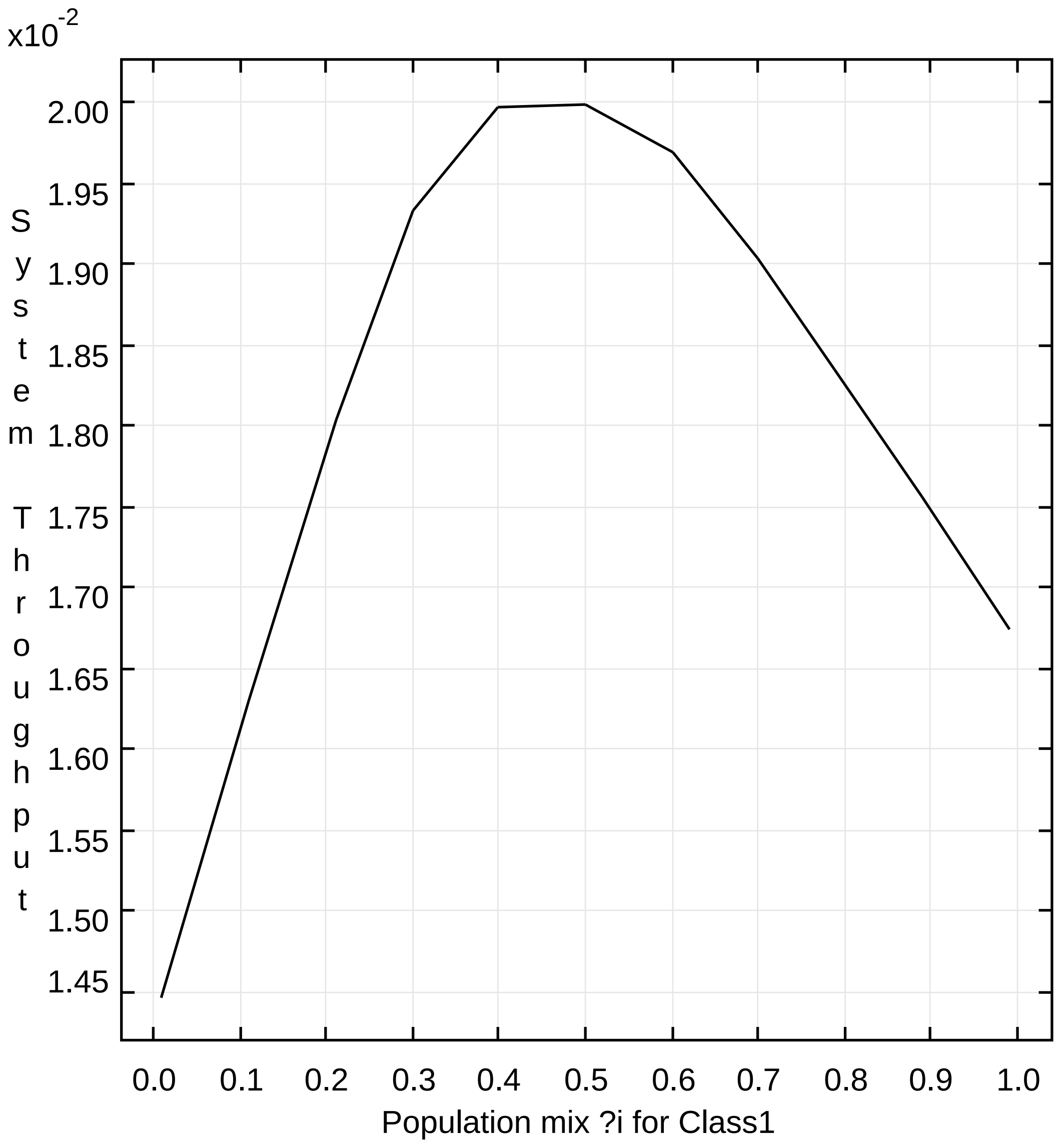


Figure 2.1: System throughput

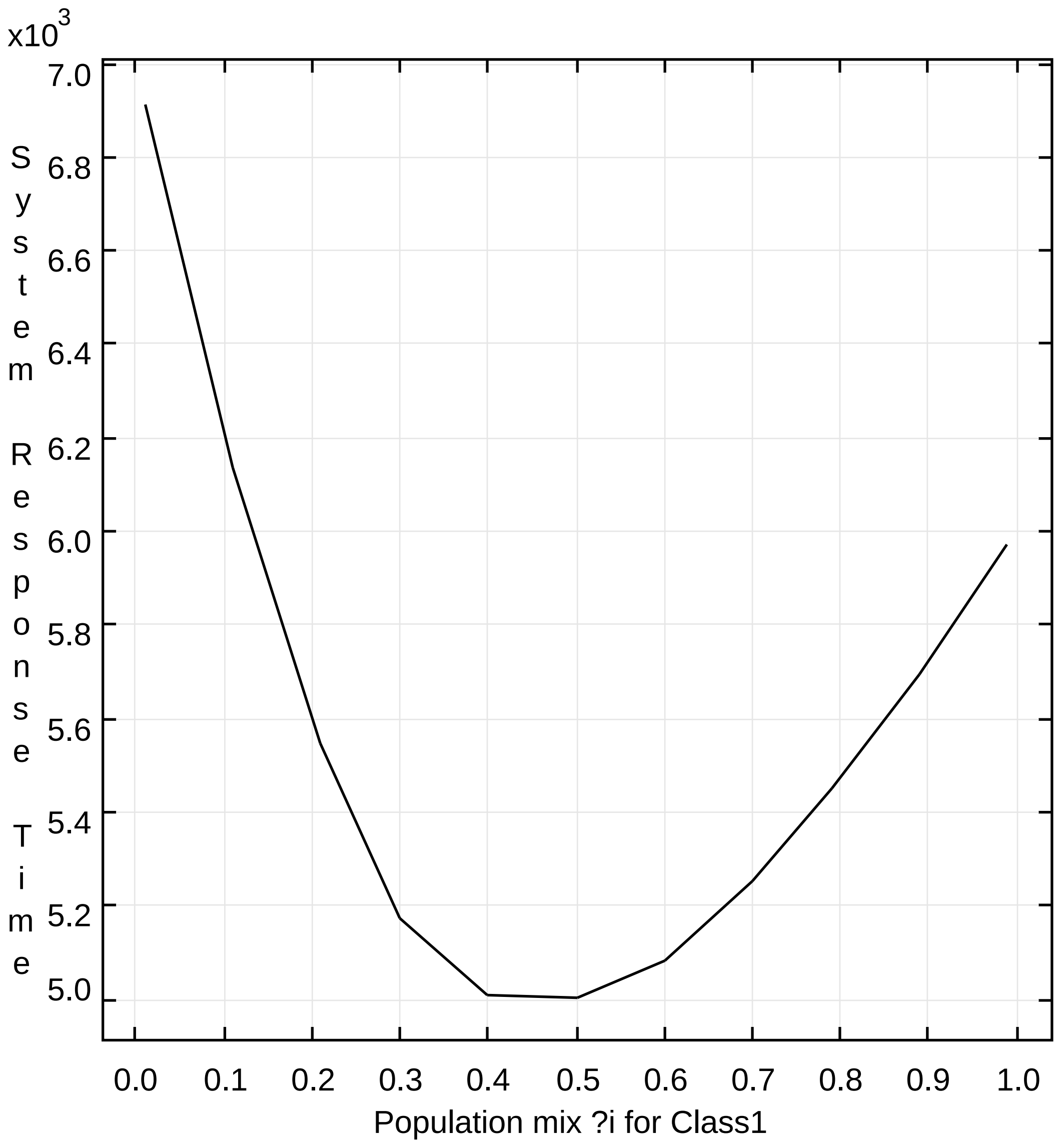


Figure 2.2: System Response time

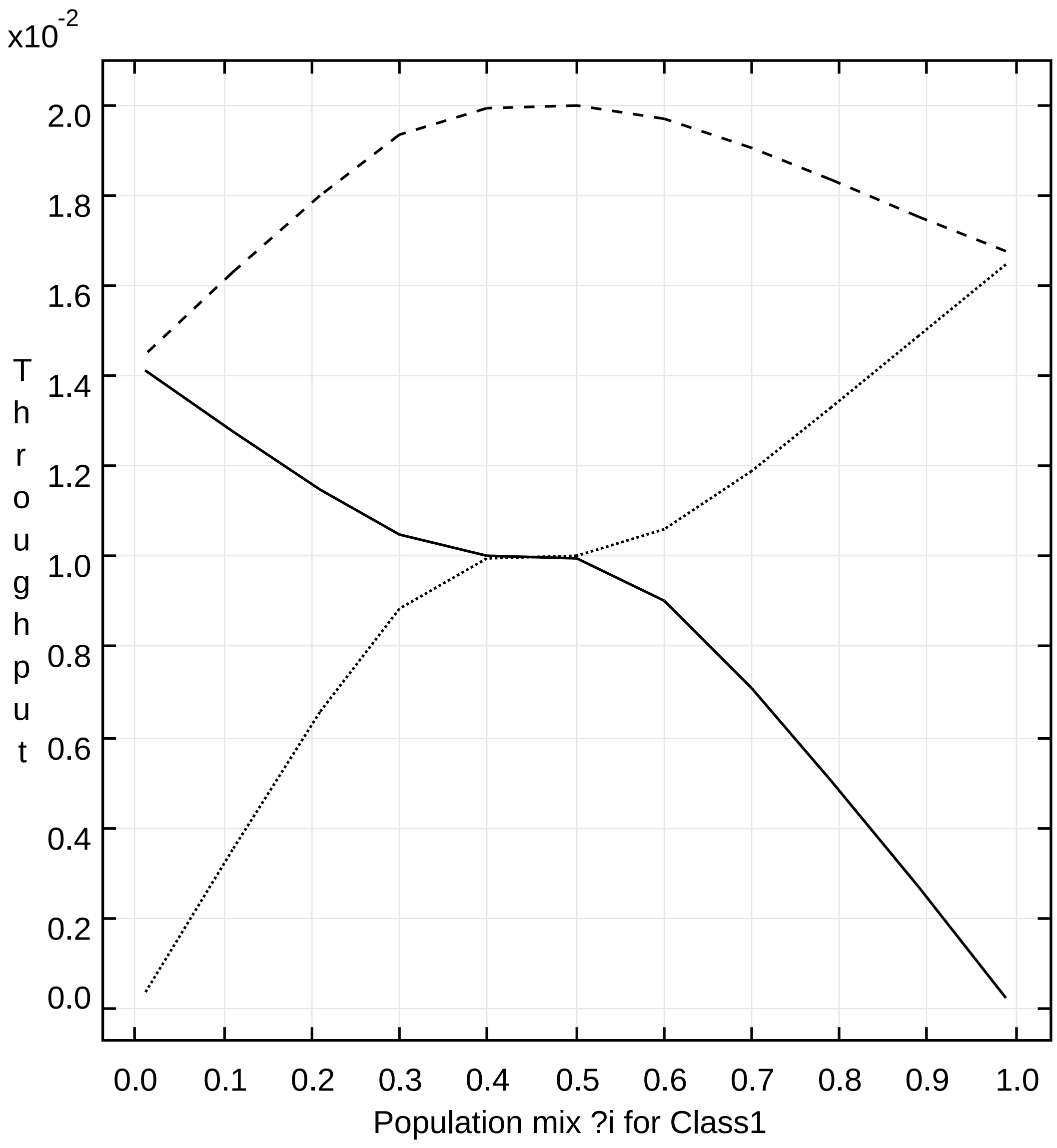


Figure 2.3: System Throughput per class (switch)

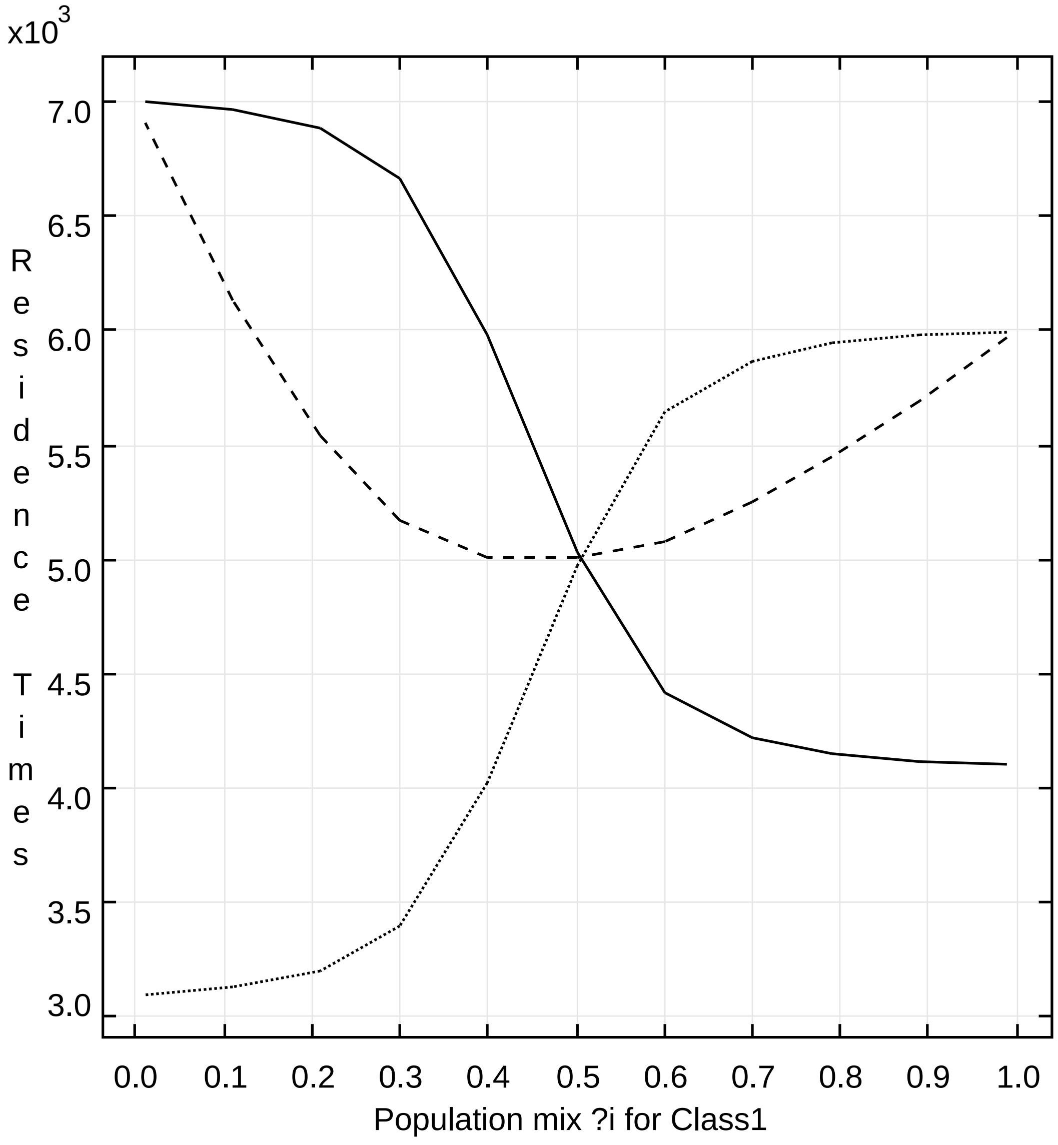


Figure 2.4: System Response time per class (switch)

of customers as presented will be used later in the thesis to calculate equiutilization point for a given Queueing Network model. In particular, as seen in a network with two stations there exists a combination of customer classes such that the stations are equally utilized regardless of the population size. Such an equiutilization point is shown to be an optimal operational point at which the system operates at maximum power.

2.1.4 Common Saturation Sector

The equiutilization population belongs to a set of populations, called common saturation sector, such that all the stations in a given queueing network saturate when the population size is large enough. When the workload mix changes, the bottleneck in the system can change as well and there may be many simultaneous bottlenecks in the system at a given time. Such a set is called common saturation sector.
Balbo and Serazzi [3] showed that in multiple workload mixes, multiple resources systems, changes in workload mixes can change the system bottleneck. The points in the workload mix space where the bottlenecks change are called crossover points, and the sub-spaces for which the set of bottlenecks does not change are called saturation sectors. The same authors, in the same paper, showed analytical relations between the workload mixes and utilization at the saturated bottlenecks as well as analytical expressions for asymptotic (with saturated resources) response times, throughput, and utilization within the saturation sectors.
As shown in the figure 2.5 we can see that for 2 station 2 class network, with mentioned service loads. The figure 2.6 shows that when the class 1 is dominating and

more than 60% in execution in the queueing network, then the station 1 is the bottleneck. When the class 2 is dominating and more than 70% in the queueing network, then station 2 is the bottleneck. But for a range when the Class 1 is between (30% - 60%) and Class 2 is between (70% - 40%) then both the stations are bottleneck. Also in this sector the Equiutilization point lies where the system has optimum performance.

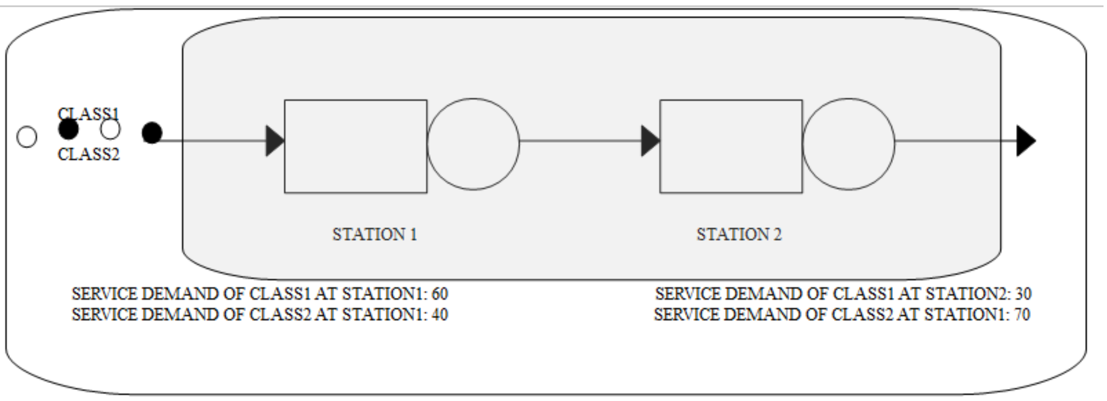

CLASS1
CLASS2
STATION 1
STATION 2
SERVICE DEMAND OF CLASS1 AT STATION1: 60
SERVICE DEMAND OF CLASS2 AT STATION1: 40
SERVICE DEMAND OF CLASS1 AT STATION2: 30
SERVICE DEMAND OF CLASS2 AT STATION1: 70

Figure 2.5: 2 class 2 station

To summarzize the mathematical equations and concepts regarding the equiutilization point, bottleneck migration and common saturation sector form the core principles and foundations of the presented thesis.

2.2 Workload Classification

Workload characterization has significant impact on performance evaluation. Understanding the nature of the workload and its intrinsic features help us to interpret

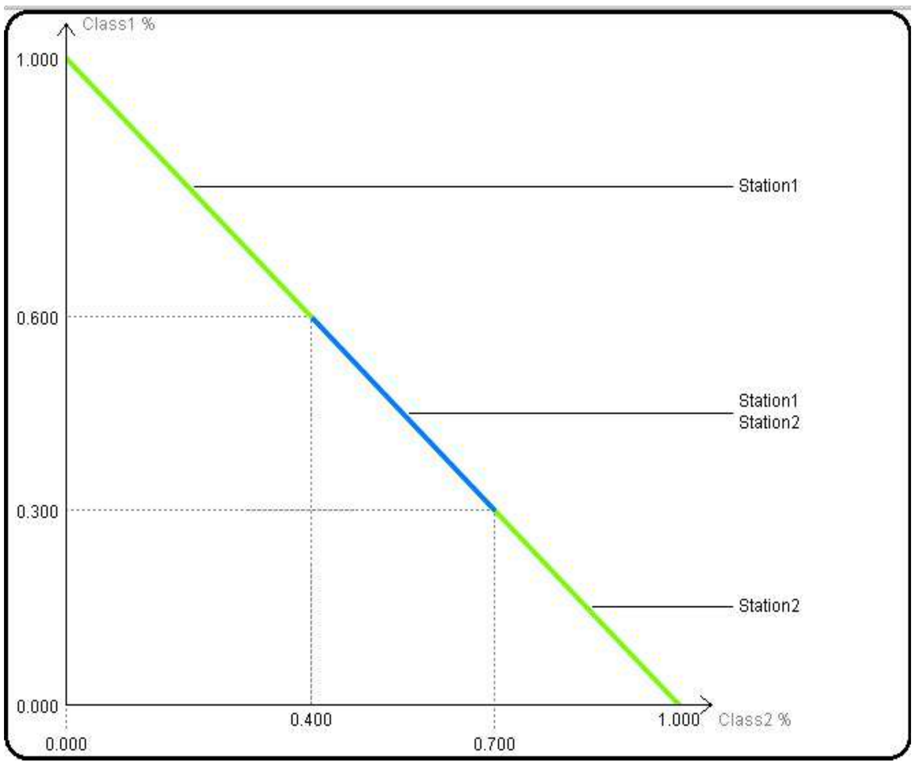


Figure 2.6: Common Saturation in 2 class 2 station

performance measurements and simulation results. Identifying and characterizing the intrinsic properties of an user request in terms of its access of servers, locality, control flow behavior etc. can eventually lead to program a model of computer system, which can be used for analytical performance modelling.
As the theory of common saturation sector and migration of bottleneck is currently based on two classes, it is important to analyse and understand how a given workload can be classified in different classes and what techniques are used to achieve the same. Workload can be characterized at various level and classified at each level depending on the perspective. Levels of Workload characterization can be Physical Level, Logical Level and Functional Level.

2.2.1 Physical Level

At Physical Level any workload can be broadly classified into two major classes. **I/O bound** and **CPU bound**. At physical level the designers of a particular application are able to interpret whether the developed tool is computation oriented or data manipulation oriented. This helps system architects to allocate right hardware resources resulting in good performance.

2.2.2 Logical Level

At Logical level the workload can be broadly classified as **response** oriented or **request** oriented. At this level the network performance managers can design the internet network to be upload intensive or download intensive.

2.2.3 Functional Level

At Functional level the workload can be classified as to which application the request is directed. For example if a network hosts three web application then, request bound for application `A` can be classified as class-`A` and so on. Such functional classification helps the service provider to negotiate SLA with clients by monitoring the usage of network resources by a given class of request.

2.2.4 Techniques

Clustering techniques are widely recommended tools for workload classification. The k-means algorithm is widely accepted as the standard technique of detecting workload classes automatically from measurement data. [14] The thesis does not go too much deep into the actual methods like web crawling and request monitoring mechansisms. Validity of the randomly generated workload classes, when the system and workload is analyzed by a queueing network model and mean value analysis.

2.2.5 Summary

Workload classification [12] provides a sound understand of how users are using the network. The resulting information can be used to simplify troubleshooting tasks and help in making decisions regarding network design and optimization. In addition, characterization data, placed in the right format, produces excellent tools for management and non-technical management like SLA negotiation.

Within the confines of a network, workload is the amount of work done by, a server, or internetwork in a given time period. Workload characterization is the science that observes, identifies and explains the phenomena of work in a manner that simplifies the understanding of how the network is being used.

Chapter 3

System model

Internet service providers usually use several server pools to host different web applications, to ensure smooth system management and minimum interference between applications. Internet applications can be modelled as multi-class queueing networks, with each queueing station corresponding to a server.

The advantage of using an queueing network model is that performance metrics can be easily computed, and potential system bottlenecks can be identified without running the actual system. In this thesis, a queueing netwoek model is used for simulation to assist dynamic resource allocation in server pools. The main components of a modern web application is

- **WS** Web server.
- **AS** Application server.
- **DS** Database server.
- **C** Incoming customer request.

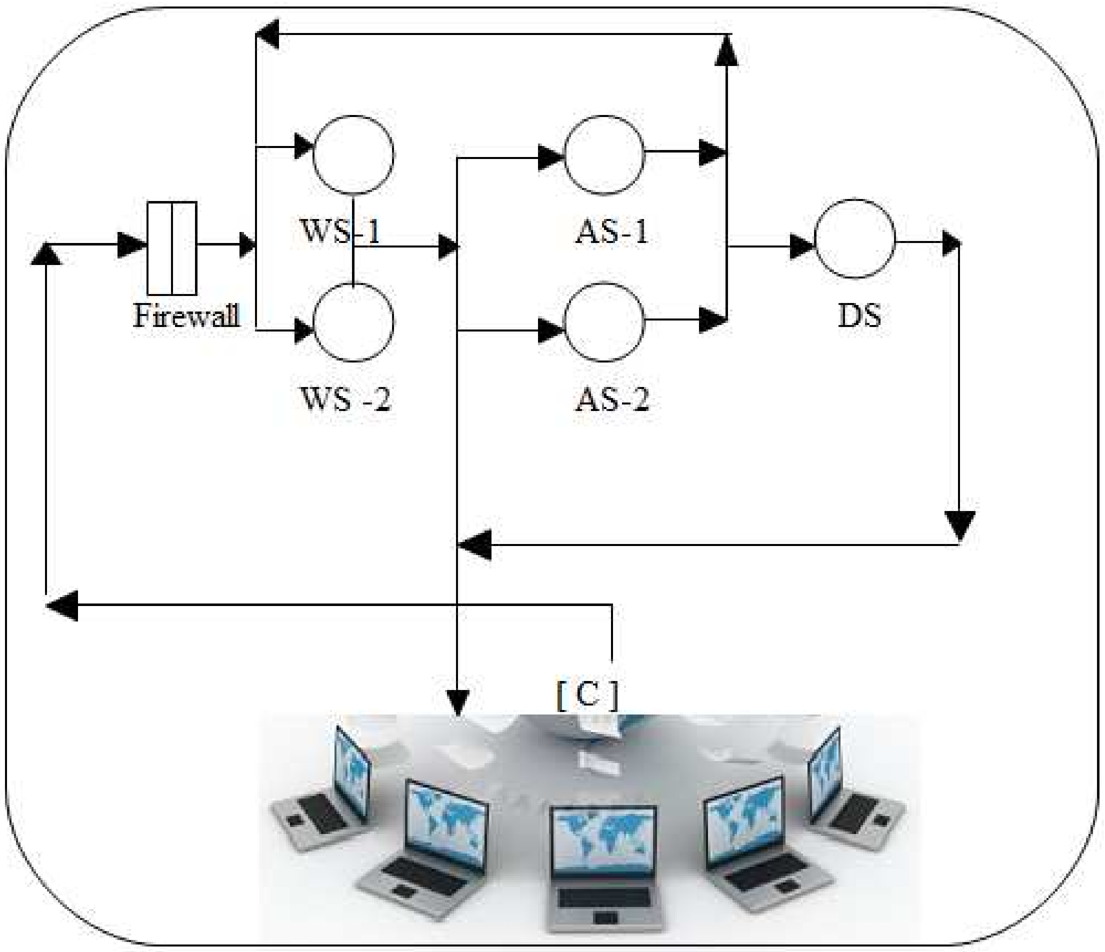


Figure 3.1: Typical web application deployment

3.1 Data center architecture

Modern data centers are large, complex and modular infrastructures with thousands of servers that host multiple Internet applications. Incoming requests are distributed among the servers of the data center through a complex dispatching mechanism guided by the URL of the request. A first level dispatcher, operating at the level of the entire data center, decides which building block has to receive the request, then a second level dispatcher selects a server within the building block. As shown in the figure 3.2.

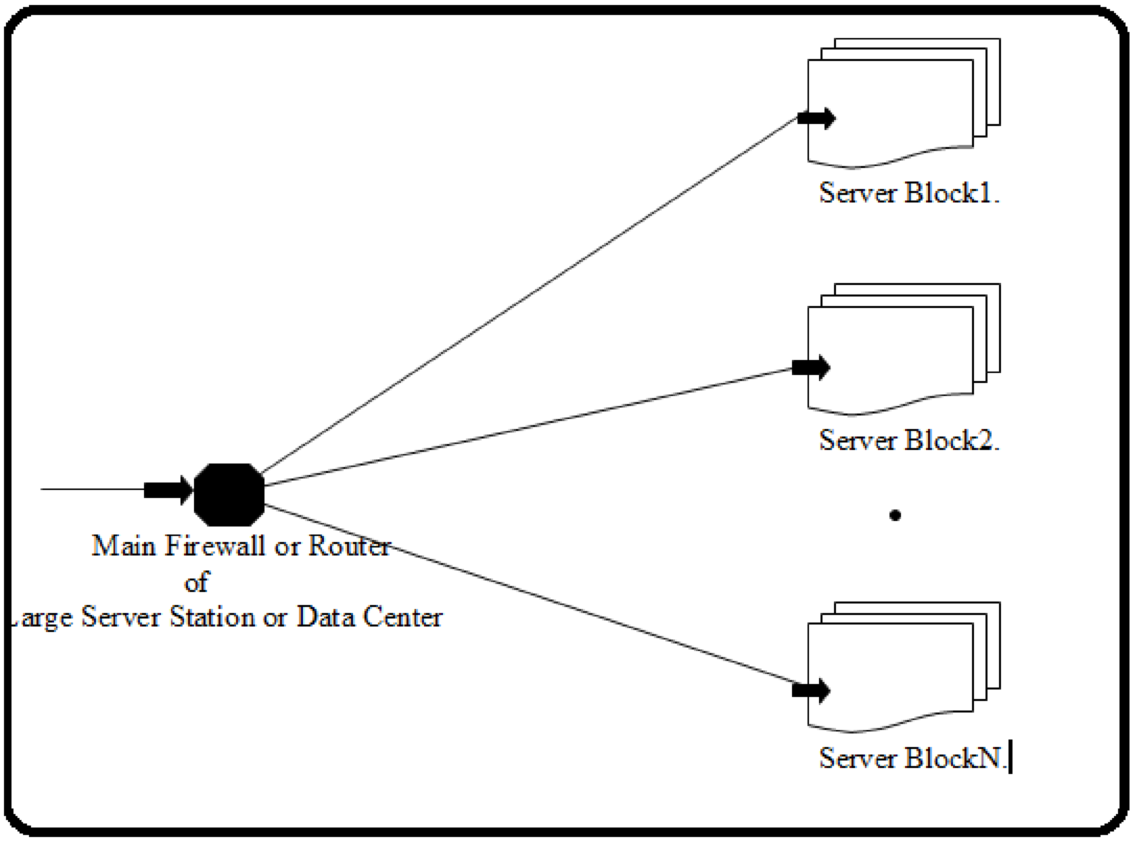


Figure 3.2: Large Internet Data Center

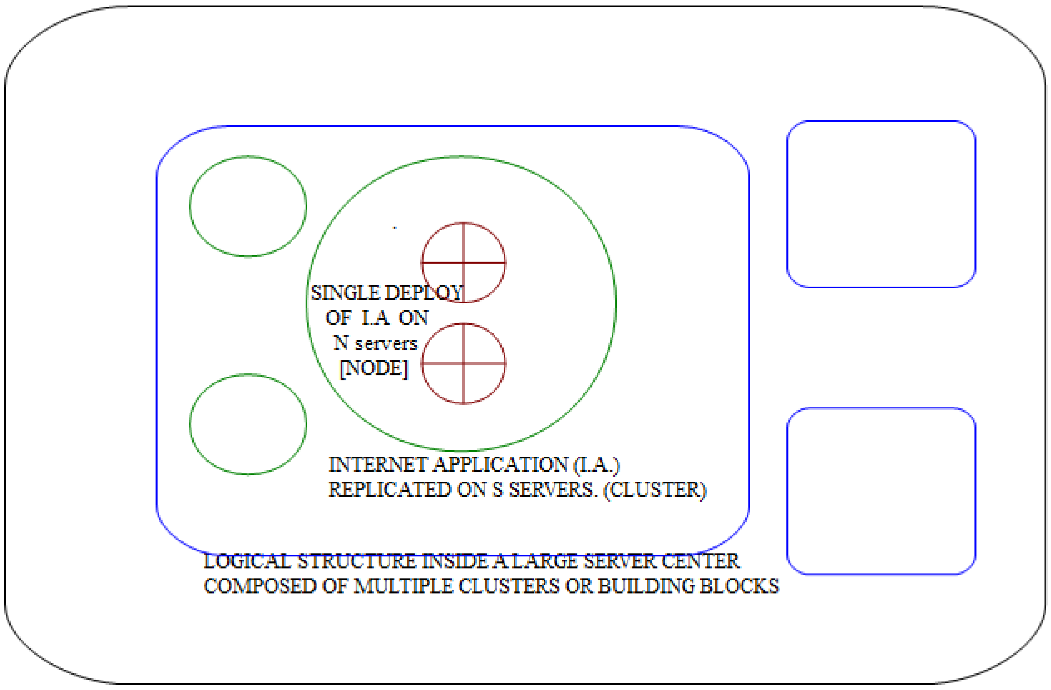

SINGLE DEPLOY
OF I.A ON
N servers
[NODE]
INTERNET APPLICATION (I.A.)
REPLICATED ON S SERVERS. (CLUSTER)
LOGICAL STRUCTURE INSIDE A LARGE SERVER CENTER
COMPOSED OF MULTIPLE CLUSTERS OR BUILDING BLOCKS

Figure 3.3: Logical partitioning of the Model

The building blocks of an Internet data center as a set of servers, where each server is basically a time shared processor. Hence forward the server will be known as **Resource**. refer figure 3.3 where a single server or resource is represented by red circles.
A single deploy of given internet application can be on a set of N servers. Hence forward such set of N servers will be called **Node**. refer figure 3.3 where a single node is represented by green circles.
The building block hosts also a Storage Area Network where multiple Network Attached Storage (NAS) are shared by the servers. It is assumed that each Internet application is replicated on a subset S of servers within the building block that share the same NAS. The redirection of a request for the Internet application will occur among the servers of the set S. We do not consider request redirection outside the set S, such as geographical or inter-block migrations, because each application must access information stored in a NAS. Migration and replication of these data across building blocks of across data centers would introduce an overhead that is not compatible with real-time redirection. The subset S will be hence forth known as **Cluster**. refer figure 3.3 where a a cluster is represented by blue.
We further elaborate the four different levels of abstraction as introduced above:

- **Resources** are units which process requests, like CPU cores, Disks, or even a more complex elements such as web-servers or Databases.

- **Nodes** are collections of interconnected resources. Jobs entering a node requires several services from each of the available resources. Jobs leave the system as soon as they have completed their services.

- **Clusters** are collections of nodes. All the nodes in a cluster can perform the same tasks. They are all equally reachable and there is no particular advantage in choosing a node in place of another. Every job entering the system, requires the services provided from only one node (i.e. a cluster could be a pool of servers). We also imagine that there is a router in front of the cluster, that can send the jobs to the best available nodes.

- **Clouds or very large Computing center** are collections of several clusters. At this level, clusters present in a cloud are not equivalent, not all the clusters can be equally reached, and other issues (such as cost, or consumption) vary from cluster to cluster. We also imagine that it applications are not aware of the structure of the cloud in which they are running.

3.2 Resource

A resource, or system resource, is any physical or virtual component of limited availability within a computer system. Every device connected to a computer system is a resource. Every internal system component is a resource. In our model, by resource we mean items like processor unit or I/O disk unit or a Web server or Database server. But such items, individually are of not of much use, but collectively they form a computing unit. In our model, resources are represented by Queueing stations where a service time required to service a request is already known.

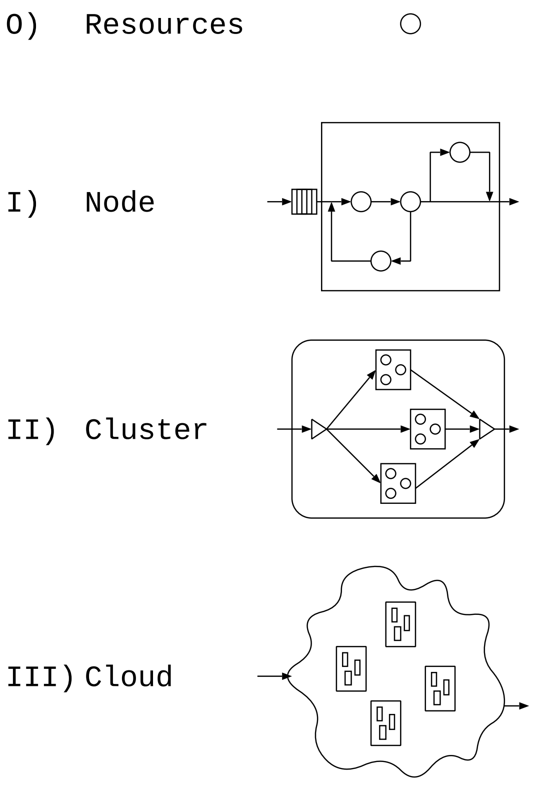


Figure 3.4: Levels of abstraction

3.3 Node

A Node is an atomic unit of computing, which is composed by group of resources that can fulfill the service needs of a job or of a request. Node can be a PC, which is a collection of resources like CPU core, Memory and Disk or it can be an Enterprise subnetwork where resources can be Application servers, Database servers and Web servers. Each job or request queues at the node for processing.

The performance of the Node can be modeled by a simple queueing network, where multiple classes of jobs are associated to different service requirements. To achieve performance boost at this atomic level, the idea is to develop a admission control mechanism for the incoming jobs based on the concept of *common saturation point* (or *region*) in a group of resources. In a multi class queueing network, it has been shown that, at a certain mix of request rates coming from the different classes, the system has a common saturation point or region. This region is also a point where the system has the best performance [3] [6], [11]. To exploit this property, we propose a new scheduling policy that pushes the node towards common saturation point, by controlling the mix of requests coming from different classes inside a node.

3.4 Cluster

Cluster is the level of categorization that considers a collection of nodes. All the nodes in a cluster can perform same tasks or can process same requests. A cluster can be for example a pool of servers. A cluster is modeled by considering a different queueing network for each of its nodes. Not all the subnetworks may be working at their common saturation region at a given time instant. This fact can be exploited to route new requests to the nodes that will get closer to their common saturation region.

3.5 Cloud or Large Data Center

Cloud computing is a paradigm that enables to run an application on a large scale computer system without having to buy it but just paying for the resources actually used. The programmer just creates an application using some high level API and then deploys it on the cloud infrastructure of the provider, having little or no knowledge on what virtual machine his application is going to be run.
Due to the high number of nodes involved and the complex and usually unknown interaction between them, it is infeasible to use queueing networks to optimize performance indices such as response time or energy consumption. In order to overcome this difficulty we need an appropriate approximated modeling technique. This thesis does not cover the approximate modeling techniques. Hence the macro performance issues on a large cloud or very large server system is out of scope of this thesis.

3.6 Request dispatching mechanism

In this section we illustrate how a user request is managed or directed to the correct server. When a request r is received by the data center main firewall or dispatcher, the first level of routing selects a building block in the data center that hosts the Internet application to which r refers.

The building block dispatcher selects a cluster, that we call ca to which the request is forwarded. As mentioned above cluster ca is a set of nodes sharing the same NAS or storage. If the request r belongs to an existing user session, the dispatching operation selects a node on the basis of a binding table that maps user sessions to nodes. If the request belongs to a new session, the building block dispatcher must select a new node in a given cluster that will host the request user session. Mostly the the selection of a new server is carried out through a load blind algorithm, such as Round Robin Random algorithms to load-aware solutions, such as Weighted Round Robin [10].

Chapter 4

Proposed Algorithm

This thesis proposes policies or algorithms exploiting the concepts of bottleneck migration and common saturation sector as introduced earlier [3], to derive performance benefits in a large server system. The policies proposed, are deployed on Node and Cluster level, relying on continuous monitoring of the servers.

4.1 Window based request management

Policies proposed in this section aims to improve the performace at a node level.

4.1.1 Window based admission algorithm

The basic idea is to add an extra waiting queue for the jobs, when they queue at a node where the admission control mechanism verifies the classes of the jobs in queue and admits only those jobs whose classes pushes the node towards the common saturation sector. To avoid stagnation of unfavorable jobs in the waiting queue, a timeout is associated with all the incoming jobs. When the time out expires, the undesired job is admitted in the node, even if this action pushes the system out of common saturation zone.

Window based admission algorithm is described below

```
1. STEP : INPUT Window w, Class_Desired cd, Time Timeout.
2. STEP : OUTPUT Job j.
3. STEP : FOR EACH i IN SIZE_OF(w) DO.
4. STEP :         IF CLASS_OF_JOB_AT(i) EQUALS cd.
5. STEP :         RETURN JOB_AT(i).
6. STEP : END FOR.
7. STEP : FOR EACH i IN SIZE_OF(w) DO.
8. STEP :         IF AGE_OF_JOB_AT(i) $\geq$ Timeout THEN.
9. STEP :         RETURN JOB_AT(i).
10. STEP :        END IF
11. STEP : END FOR.
12. STEP : RETURN FIRST_JOB_AT(w).
```

4.1.2 Window based request redirection

In the above mechanism of `Window based admission algorithm`, it is proposed that unfavorable jobs are admitted in the node, even if pushes system out of common saturation zone, To overcome this limitation, another improvement one the idea is to redirect the unfavorable jobs to another node, where it can be served immediately. In this mechanism, it is proposed to redirect the unfavorable jobs from the waiting queue of a given node to another node, where the job will push the node towards common saturation.

Window based request redirection is described below

```
1. STEP : INPUT Window w, Time time_out.
2. STEP : OUTPUT Node n.
3. STEP : DECLARE Node out, NUMBER min.
4. STEP : FOR EACH i IN SIZE_OF_(w) DO.
5. STEP :         IF AGE_OF_JOB_AT(i) $\geq$ Timeout THEN.
6. STEP :         j[i] = JOB_AT(i).
7. STEP :         END IF.
8. STEP : END FOR.
9. STEP : FOR EACH n IN List_of_(N) DO.
10. STEP :        c[n] = getCurrentClassMix of n.
11. STEP :        o[n] = getOptimunClassMix of n.
```

```
12. STEP :     h[n] = hypotheticalClassMix(cn, ji)if n receives j.
13. STEP :     dh[n] = **MODULUS**(o[n] - h[n]).
14. STEP :     dc[n] = **MODULUS**(o[n] - c[n]).
15. STEP : **END FOR**.
16. STEP : **FOR EACH** k **IN** List_of_(N) **DO**.
17. STEP :     **IF** dh[k] $\leq$ dc[k] **THEN**.
18. STEP :         **IF** min $\leq$ dh[k] **THEN**.
19. STEP :         **ASSIGN** out = k.
20. STEP :         **ASSIGN** min = dh[k].
21. STEP :         **END IF**.
22. STEP :     **END IF**.
23. STEP : **END FOR**.
24. STEP : **RETURN** out.
```

4.2 Common Saturation directed Routing

The policy described in this section is aimed at cluster level. The routing policy is designed so that it detects the class of a new incoming job or request and proceeds to calculate the change in the workflow mix for each of the forwarding nodes. The policy then routes the incoming job to the node for which such arrival will cause the best increase (or the least decrease) in performance, by bringing it closer to its common saturation sector.

Equiutilization oriented Routing Algorithm is described below

```
1. STEP : INPUT List_Nodes N, Job j.
2. STEP : OUTPUT Node n.
3. STEP : DECLARE Node out, NUMBER min.
4. STEP : FOR EACH i IN Size_of_(N) DO.
5. STEP :         ASSIGN c[i] = getCurrentClassMix of i.
6. STEP :         ASSIGN o[i] = getOptimunClassMix of i.
7. STEP :         ASSIGN h[i] = hypotheticalClassMix(ci, j)if i receives j.
8. STEP :         ASSIGN dh[i] = MODULUS(oi - hi).
9. STEP :         ASSIGN dc[i] = MODULUS(oi - ci).
10. STEP : END FOR
11. STEP : ASSIGN min = MAX_INTEGER.
12. STEP : FOR EACH i IN List_of_(N) DO
13. STEP :         IF dh[i] $\leq$ dc[i] THEN.
14. STEP :                 IF min $\leq$ dh[i] THEN.
15. STEP :                 min = dh[i].
```

```
16. STEP :             **END IF**.
17. STEP :       **END IF**.
18. STEP : **END FOR**.
19. STEP : out = Node_at(N, min).
20. STEP : **RETURN** out.
```

Indeed, a cluster hosts hundreds of nodes, all mirroring the same Internet application, and the cluster router needs to collect and aggregate in a timely way the information about the load of each node.

4.3 Hypothetical implementation

User request `r` is routed from the main firewall or gateway to the cluster which hosts or has the installation of the application. This is done by reading the header information in the request packet. Inside the cluster the web application is mirrored at various nodes.

The cluster router then directs the request to the most appropiate node $\mathtt{n}_i$, depending on the best increase (or least decrease) in performance, on recieving the new request. Each node contains a window management module that is responsible for the admission of requests at a fine-grained time scale. Request `r` of class $\mathtt{c}_i$ is received by the node $\mathtt{n}_i$ at time t. Depending on the system conditions, the request may be processed locally or redirected to a node $\mathtt{n}_j$ of the same cluster. It should be noted that in modern web based servers, the service of a user request involves access to data, such as the user session information, that must be migrated at the moment of request redirection. Hence, by redirection we mean both the migration of user session information and the actual forwarding of request r to the node $\mathtt{n}_j$. We can summarize the functions of the admission controller as follows:

1. Should request r be processed locally on node $\mathtt{n}_i$?
2. Should it be migrated to another node $\mathtt{n}_j$?
3. In the latter instance, which is the best node $\mathtt{n}_j$ to which the request should be sent?

Refer figure 4.1 for the graphical illustration of the proposed algorithms at various levels in an enterprise network.

.

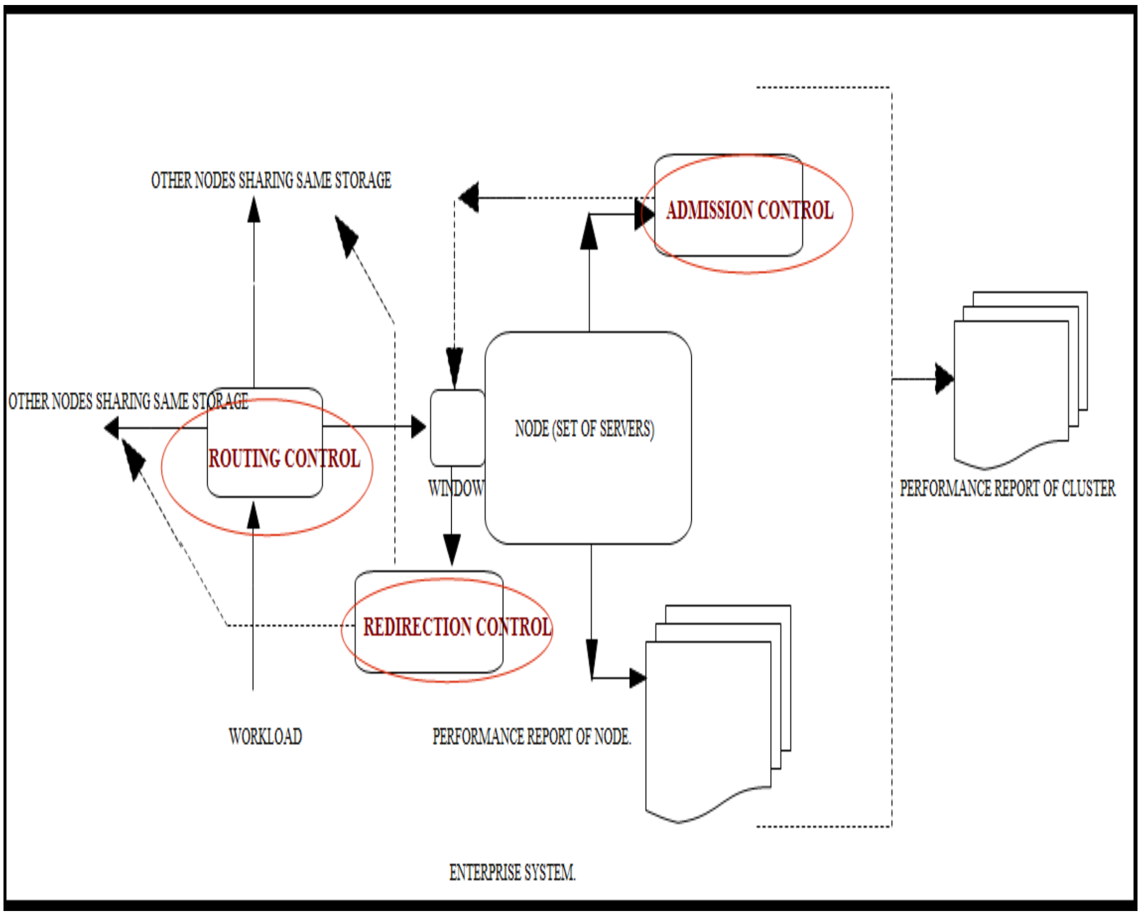

OTHER NODES SHARING SAME STORAGE
ADMISSION CONTROL
OTHER NODES SHARING SAME STORAGE
ROUTING CONTROL
NODE (SET OF SERVERS)
WINDOW
PERFORMANCE REPORT OF CLUSTER
REDIRECTION CONTROL
WORKLOAD
PERFORMANCE REPORT OF NODE.
ENTERPRISE SYSTEM.

Figure 4.1: Hypothetical implementation at various levels

Chapter 5

Testbed for Performance Evaluation

In this chapter the testbed and the workload considered for the simulation evaluation is explained. The simulation and modeling of the techniques developed in this thesis were tested in a tool called Java Modelling Tools (hence forth will be called as JMT). JMT is a suite of applications developed by Politecnico di Milano and released under GPL license. The project of JMT offers a complete framework for performance evaluation, system tuning, capacity planning and workload characterization studies. JSIMgraph, an simulation application within the suite of JMT allows the design of a queuing network model in graphical way. The model can be solved either with simulation techniques or, if the model is BCMP compliant, automatically exported to JMVA to be solved with exact or approximate MVA algorithm.

5.1 JSIM*graph* - Intro

JSIM supports state-independent routing strategies, e.g., Markovian or round robin, as well as state-dependent strategies, e.g., routing to the server with minimum utilization, or with the shortest response time, or with minimum queuelength. Performance indices like throughputs, utilizations, response times, residence times, queue-lengths are evaluated. The simulation engine supports several probability distributions for characterizing service and inter-arrival times, e.g., exponential, hyperexponential, uniform, Erlang and Pareto.

The project aims at offering a complete framework for performance evaluation, system tuning, capacity planning and workload characterization studies. JSIMgraph allows the design of a queuing network model in graphical way and the model can be solved with simulation techniques.

It provides following functionality: 1. Creation of stations, links and grouping into blocking regions with drag and drop functionality 2. Show queue animation during simulation process: that will give a smart information on queue state and utilization.

5.1.1 Architecture of the tool

The JSIM*graph* has been designed to be very flexible. The important feature is separation between the GUI and computation engine by introducing an XML layer as shown in 5.1.

The tool can be divided in 3 layers.

1. GUI LAYER(Graphical User Interface to design and configuring models)

2. XML LAYER(saving and reading models to file system)

3. SIMULATION LAYER(actual implemented strategies and logic)

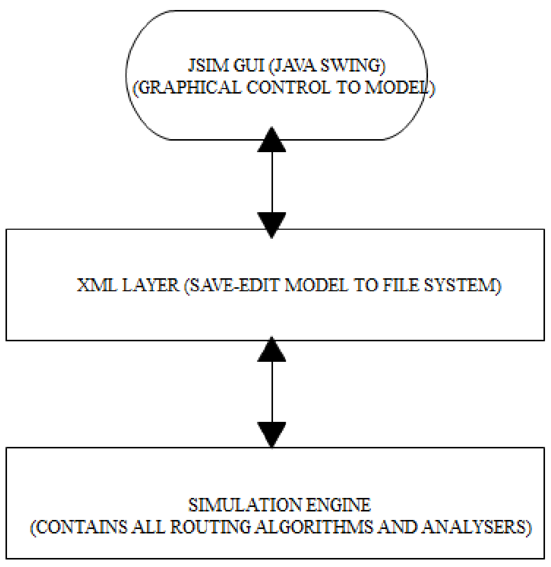
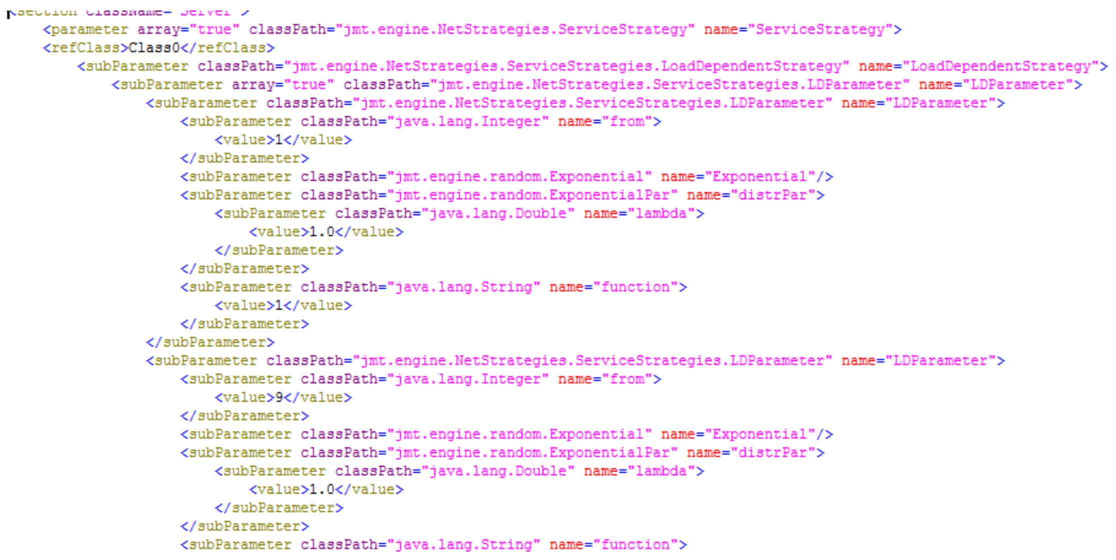

JSIM GUI (JAVA SWING)
(GRAPHICAL CONTROL TO MODEL)
XML LAYER (SAVE-EDIT MODEL TO FILE SYSTEM)
SIMULATION ENGINE
(CONTAINS ALL ROUTING ALGORITHMS AND ANALYSERS)

Figure 5.1: JSIM Framework

In JSIMGraph, the model of Queueing Network created using the Graphical User Interface (i.e. GUI) is saved in the form of a XML. The graphical user interface developed using Java Swing Architecture. The gui modifies the model which are saved in the the java class ClassModel, which is then converted into an XML file using the JSIM.xsd file. The XMLFile 5.2 which is a well formed XML file contains all the details regarding to the model.

The architecture allows reuse of the simulation or computation engine by other tools or projects by providing a suitable XML input file. At the end of the computation and simulation the performance measures and indices are in to the solution element of input file.



```xml
<section className="Server">
    <parameter array="true" classPath="jmt.engine.NetStrategies.ServiceStrategy" name="ServiceStrategy">
    <refClass>Class0</refClass>
        <subParameter classPath="jmt.engine.NetStrategies.ServiceStrategies.LoadDependentStrategy" name="LoadDependentStrategy">
            <subParameter array="true" classPath="jmt.engine.NetStrategies.ServiceStrategies.LDParameter" name="LDParameter">
                <subParameter classPath="jmt.engine.NetStrategies.ServiceStrategies.LDParameter" name="LDParameter">
                    <subParameter classPath="java.lang.Integer" name="from">
                        <value>1</value>
                    </subParameter>
                    <subParameter classPath="jmt.engine.random.Exponential" name="Exponential"/>
                    <subParameter classPath="jmt.engine.random.ExponentialPar" name="distrPar">
                        <subParameter classPath="java.lang.Double" name="lambda">
                            <value>1.0</value>
                        </subParameter>
                    </subParameter>
                    <subParameter classPath="java.lang.String" name="function">
                        <value>1</value>
                    </subParameter>
                </subParameter>
                <subParameter classPath="jmt.engine.NetStrategies.ServiceStrategies.LDParameter" name="LDParameter">
                    <subParameter classPath="java.lang.Integer" name="from">
                        <value>9</value>
                    </subParameter>
                    <subParameter classPath="jmt.engine.random.Exponential" name="Exponential"/>
                    <subParameter classPath="jmt.engine.random.ExponentialPar" name="distrPar">
                        <subParameter classPath="java.lang.Double" name="lambda">
                            <value>1.0</value>
                        </subParameter>
                    </subParameter>
                    <subParameter classPath="java.lang.String" name="function">
```

Figure 5.2: Sample XML

5.1.2 Simulation Engine

The core module of the simulation engine is a discrete event calendar that acts as messaging service provider, sending messages to simulation engines. The arrival of a job to the queueing station and corresponding departure after service completion, is represented by message. When all the event for a given time are processed, the simulation current time is moved forward to the next instant with a given time in the calendar.

In the simlation network, each service station is composed of three parts, called Sections. The three sections are named as Input section, Service section and Output section. The service demands are specified through the GUI interface in the service section of the queueing station. In a queueing station the input section receives incoming jobs to be processed by the service section. The service section simulates the service process with the user specified service time distribution. After service completion, jobs are forwarded to the output section called Router which sends the request to the input section of another queueing or service center according to user specified routing policy.

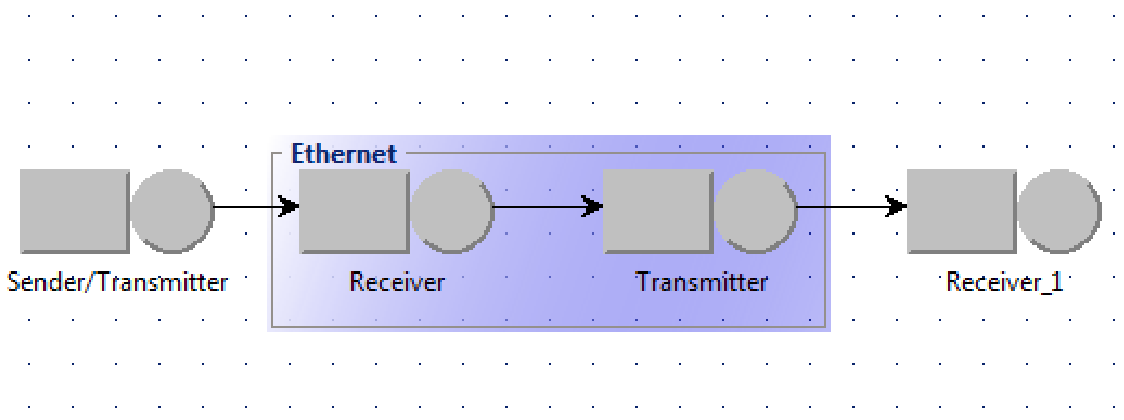


Figure 5.3: FCR in JMT

5.1.3 Finite Capacity Regions

Models of simultaneous resource possession due to memory or software constraints often require Finite capacity regions (see, fig 5.3). These are subnetworks where the number of circulating jobs is constrained. Shared constraints impose an upper bound on the allowed number of jobs in the region regardless of their service class. Dedicated constraints, instead, limit the number of cycling jobs for a specific class. Jobs arriving to a full region enqueue in a waiting buffer outside the region. The presence of the waiting buffer makes it dicult to obtain an analytical solution to models with Finite capacity regions. Therefore, only approximation techniques have been de- veloped. However, simulation remains the most important analysis technique in presence of realistic workloads with multiple classes.

Finite capacity regions are implemented as follows. The waiting buffer is a service center with infinite capacity queue and tunnel service section. The output section implements the access control policy according to the user-specified shared and dedicated constraints. Waiting jobs are selected to enter the region according to a FCFS discipline. We point out that region access control is centralized, i.e., all arrivals are routed to the same waiting buffer. Currently, the simulator does not support multiple waiting buffers and nested or intersecting regions. We also remark that, when a region is full, the user can force the simulator to drop arriving jobs. This may be used to represent systems with losses, e.g., the M=M=1=k queue.

For the implementation of the new routing algorithms and to validate the theory of existence of population mix which affects the utilization of the stations and the presence of optimal operational point based on population mix of classes, the JSIMgraph framework is used. JSIMGraph code was modified to incorporate the simulation of an queueing network model with the proposed modifications of adding a new way of forwarding a job to the Queueing station based on the population mix of the system.

5.1.4 Performance Indices

JMT simulator allows the computation of several performance indices. To collect samples a special data structure known as JobInfoList is used. It logs the arrival time of jobs and feeds a given statistical analyser with the collected data.

5.1.5 Statistical Analysis

The default measure analysis is using transient detection and confidence interval estimation algorithm. The analysis are executed at run time of the simulation. The confidence intervals are computed using spectral methods. Unless long run simulation are considered, the transient effects can significantly affect simulation results. Transient detection uses R5 heuristics and MSER-5 stationary rule. In addition to these pre existing analysis, normal moving average algorithm was also coded to the simulation layer, to see the behaviour of the performance indices without advanced statistical methods.

5.2 Testbed

The theoritical foundation was based an closed network model with two fixed rate single server stations and two classes of customer. But modern computer infrastructure can not be accuretly represented by such closed network queueing models. To extrapolate the theory for large computing networks, the closed class model has to converted into an open class network model. To achieve the conversation, the closed class model was modeled as open class model with a finite capacity region. Finite capacity region limits the number of customer for a given queueing stations and hence recreating the same effect as number of customer in closed class model.
Please see figure 5.4. This is the model used to simulate the behaviour of a Node. The window is located just before the FCR region. We assume a large pool of jobs of different classes (here two), to be present outside the window. Whenever a job exists from the FCR, a new job waiting in the window queue enters the system. The

dynamic admission policy selects the job depending on the parameters set as shown in figure 5.5. The Admission control panel allows the user to control the following:

1. The input workload population.
2. Window size
3. Age which calibrates the timeout parameter.
4. Pop Mix. in Execution controls a the mix of classes at run time.
5. Redirect Jobs. allows the window to redirect unfavorable jobs to other nodes.
6. Analyser. allows user to select a given statistical analyser for the simulation.

As seen in the figure 5.6, where the testbed used to find the performance benefits of using the Equiutilization based routing policy in a Cluster to select the best possible node.

Experiments have been conducted for variuos workloads. The results generated are compared with cases where the proposed algorithms are switched off. The case studies are further discussed in the next chapter, where we simulate the testbed discussed in this chapter.

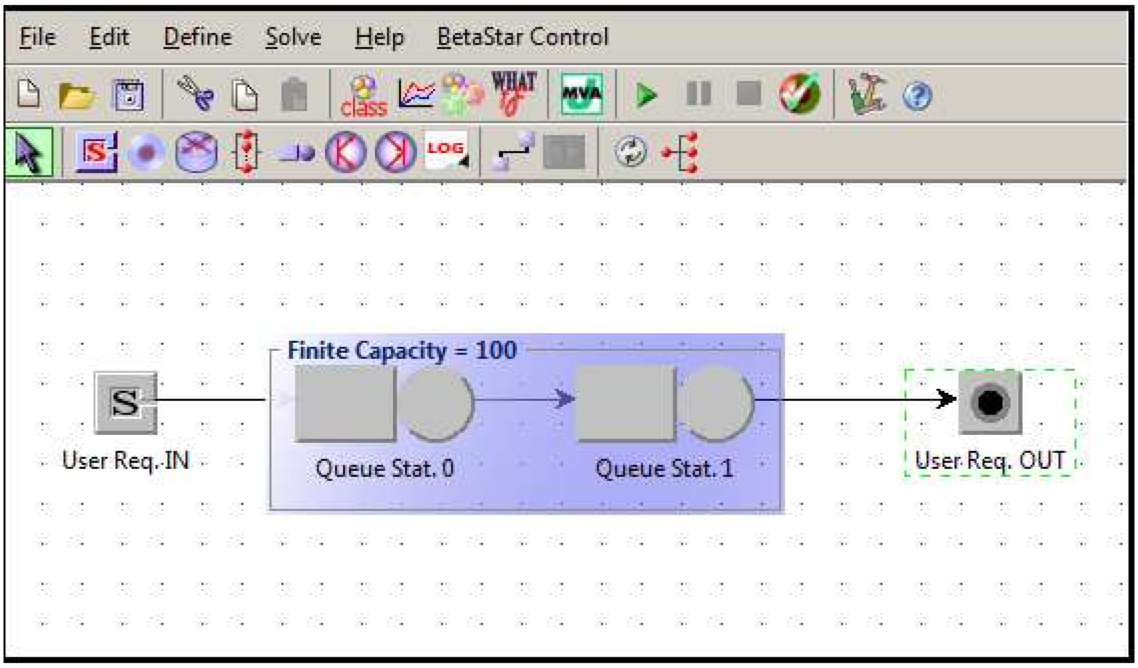
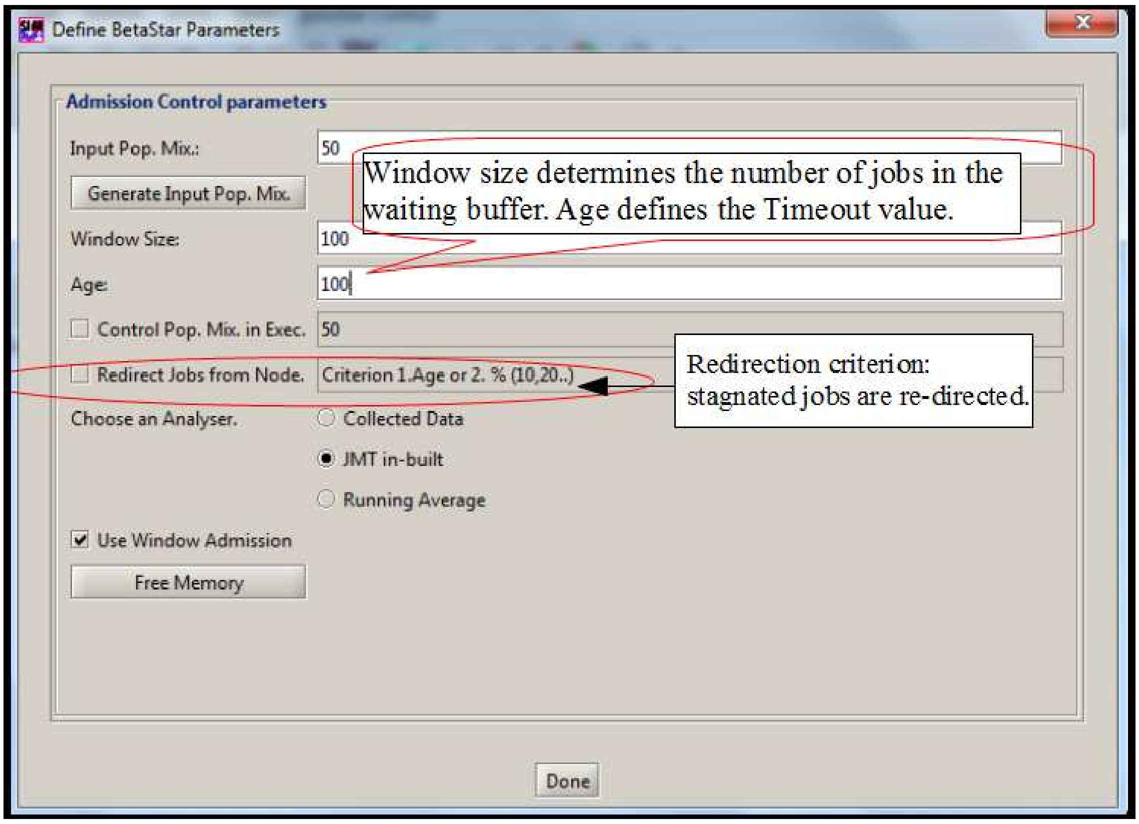


Figure 5.4: Admission Control testbed of node



Define BetaStar Parameters

Admission Control parameters

Input Pop. Mix.: 50

Generate Input Pop. Mix.

Window Size: 100

Age: 100

Control Pop. Mix. in Exec. 50

Redirect Jobs from Node. Criterion 1.Age or 2. % (10,20..)

Choose an Analyser. ○ Collected Data ● JMT in-built ○ Running Average

☑ Use Window Admission

Free Memory

Done

Window size determines the number of jobs in the waiting buffer. Age defines the Timeout value.

Redirection criterion: stagnated jobs are re-directed.

Figure 5.5: Admission Control testbed parameters.

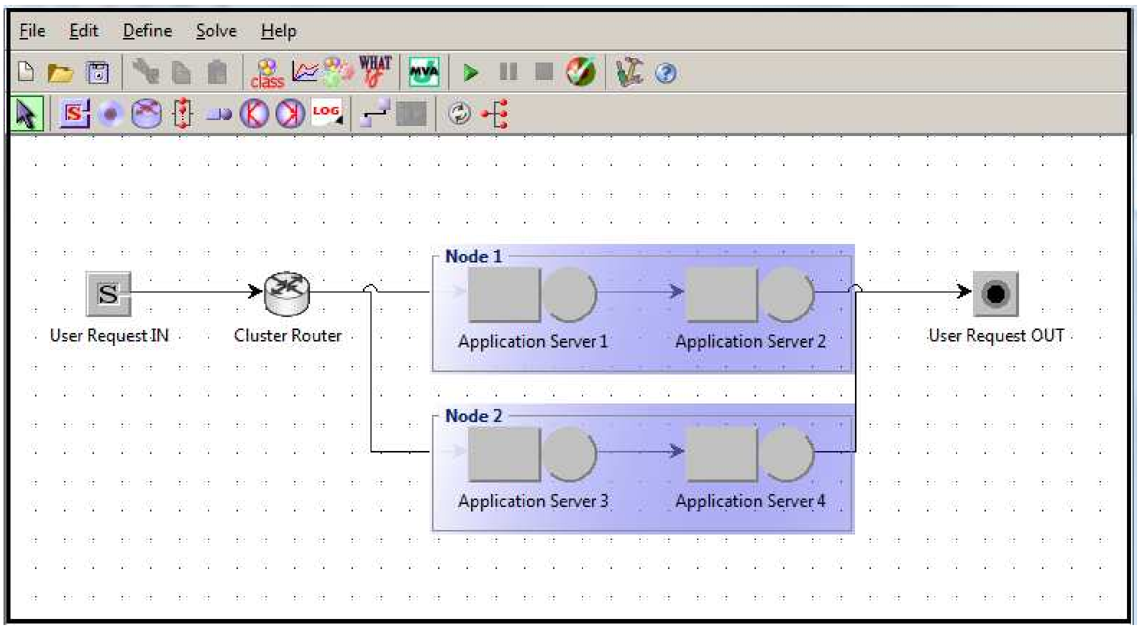


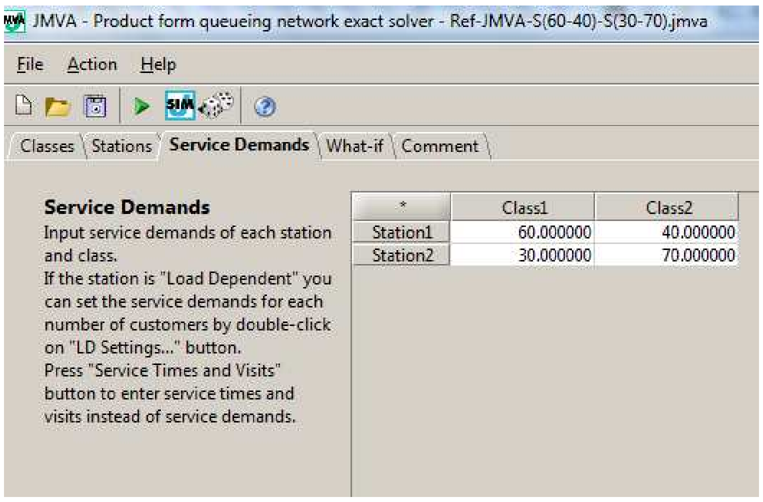
Figure 5.6: Routing Algorithm testbed of cluster

Chapter 6

Results

6.1 Validation

As discussed in the previous chapters, the theoretical results are based on two station and two class closed queueing network model. The closed class queueing network model was converted in to an open class model, using the concept of finite capacity region. First need to validate the above assumption, for which we need to compare simulation results of throughput and response time of the open class model, with the analytical values obtained from closed models. The loading matrix of the closed class model is shown in figure 6.1. The total number of customer is 100 for the analytical model. The testbed with the admission control parameters are shown in the figure 6.2 On simulation the open class model or testbed by controlling the population of class 1 in execution by using the window admission mechanism, following results are obtained. On analysing the resutls, we conclude that, indeed the open class model with FCR accurately gives the same result as closed class model. Hence we take this testbed as reference proceed to experiment further to understand the benefits of the new logic.


JMVA - Product form queueing network exact solver - Ref-JMVA-S(60-40)-S(30-70).jmva
File Action Help
Classes Stations Service Demands What-if Comment

Service Demands
Input service demands of each station and class.
If the station is "Load Dependent" you can set the service demands for each number of customers by double-click on "LD Settings..." button.
Press "Service Times and Visits" button to enter service times and visits instead of service demands.

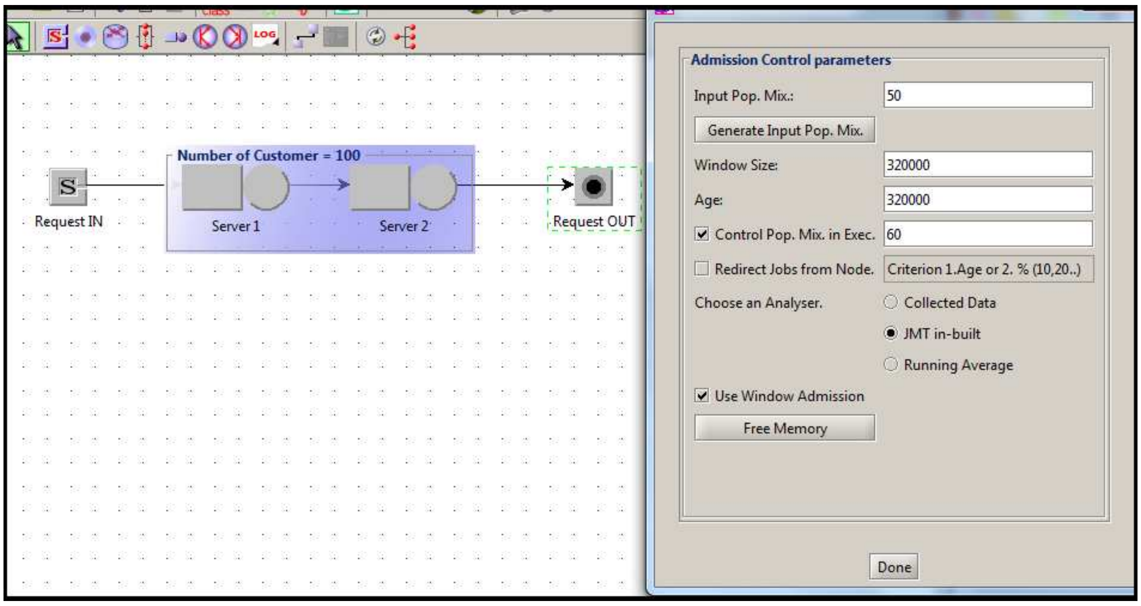
*	Class1	Class2
Station1	60.000000	40.000000
Station2	30.000000	70.000000

Figure 6.1: Service demand matrix

6.2 Case Study

As seen in the previous section, we were successfully able to validate that the simulation model and the analytical results from JMVA as equivalent. Now we simulate the given model for various workloads, to understand the limitations and benefits of the proposed algorithms in the thesis.

It should be pointed out that the performance of a system is determined by its characteristics as well as by the composition of the load being processed [5]. Different



Admission Control parameters

Input Pop. Mix.:	50
Window Size:	320000
Age:	320000
[✓] Control Pop. Mix. in Exec.	60
[ ] Redirect Jobs from Node.	Criterion 1.Age or 2. % (10,20..)
Choose an Analyser.	○ Collected Data
	● JMT in-built
	○ Running Average
[✓] Use Window Admission	

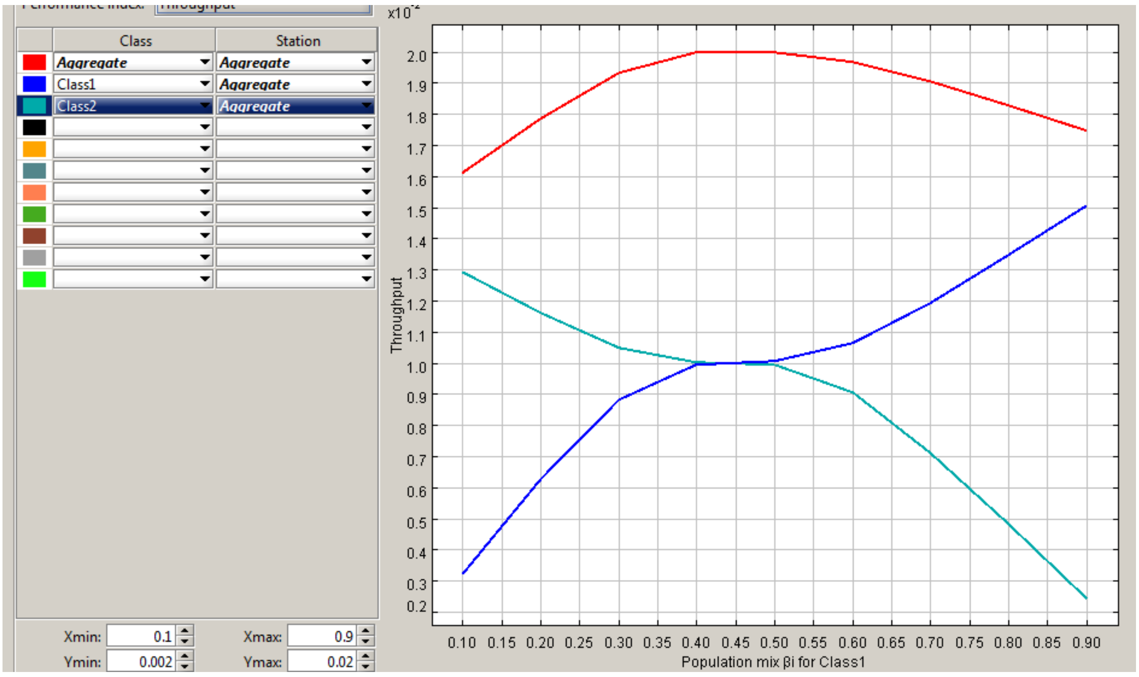
Figure 6.2: Test bed of node with admission control

Table 6.1: Analytical values: system throughput

Class-1(%)	Global	Class-1	Class-2
10	0.01611	0.00320	0.01292
20	0.01786	0.00626	0.01160
30	0.01934	0.00884	0.01050
40	0.01997	0.00994	0.01003
50	0.01998	0.01004	0.00994
60	0.01968	0.01064	0.00905
70	0.01904	0.01193	0.00711
80	0.01827	0.01345	0.00482
90	0.01748	0.01505	0.00243

Table 6.2: Simulation values: system throughput

Class-1(%)	Global	Class-1	Class-2
10	0.016174	0.003157	0.013106
20	0.017831	0.006376	0.011442
30	0.019330	0.008879	0.010441
40	0.019992	0.010011	0.010034
50	0.020026	0.010160	0.009981
60	0.019657	0.010676	0.009050
70	0.019062	0.011993	0.007157
80	0.018374	0.013515	0.004860
90	0.017289	0.014873	0.002410



	Class	Station
🟥	Aggregate	Aggregate
🟦	Class1	Aggregate
🟦	Class2	Aggregate

Xmin: 0.1 Xmax: 0.9
Ymin: 0.002 Ymax: 0.02

Figure 6.3: Analytical(JMVA) graph: throughput

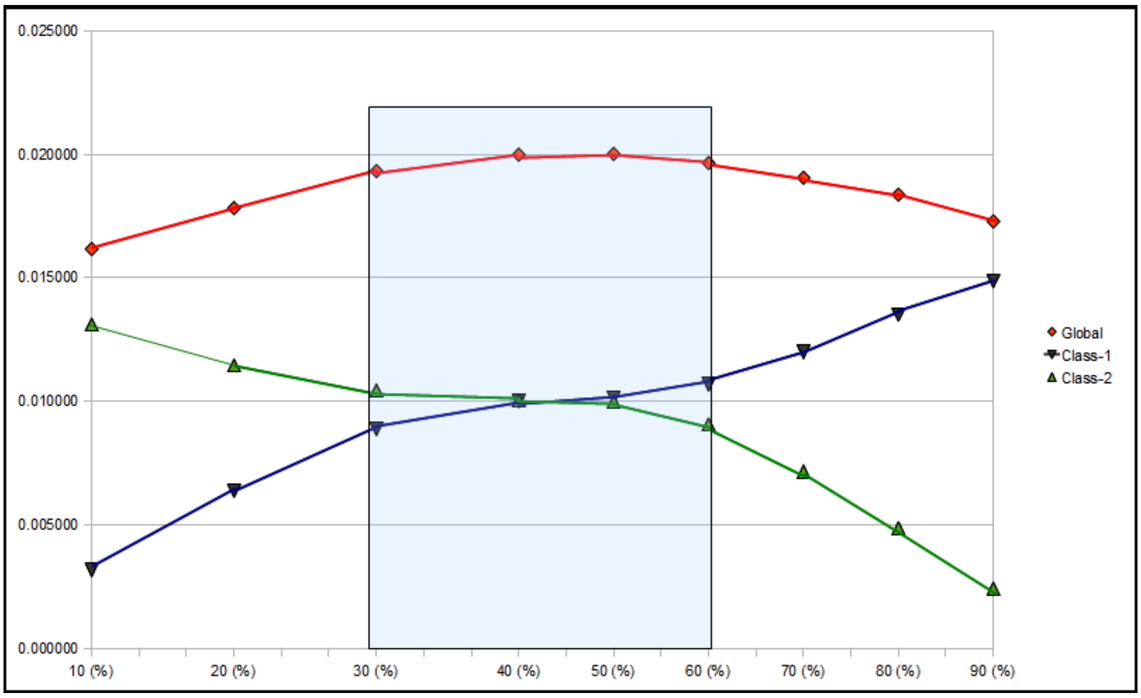
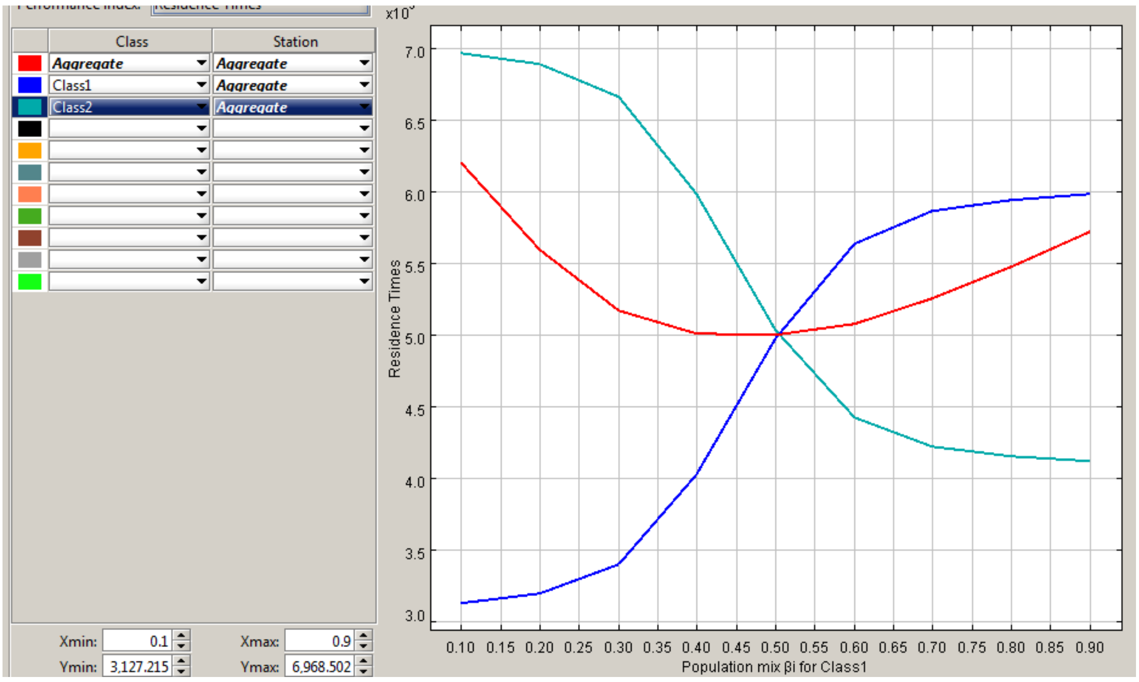


Figure 6.4: Simulation graph: throughput



	Class	Station
🟥	Aggregate	Aggregate
🟦	Class1	Aggregate
🟦	Class2	Aggregate

Xmin: 0.1 Xmax: 0.9
Ymin: 3,127.215 Ymax: 6,968.502

Figure 6.5: Analytical graph: response time

Table 6.3: Analytical values: system response time

Class-1(%)	Global	Class-1	Class-2
10	6206.17261	3127.21513	6968.50236
20	5597.68257	3193.38970	6895.60166
30	5171.60062	3394.18224	6668.11459
40	5008.64743	4024.24265	5984.61266
50	5005.29805	4979.01842	5031.85657
60	5080.78534	5641.21494	4421.84884
70	5252.98218	5869.33730	4219.16215
80	5472.36608	5946.74507	4148.60824
90	5721.99212	5981.20380	4116.42442

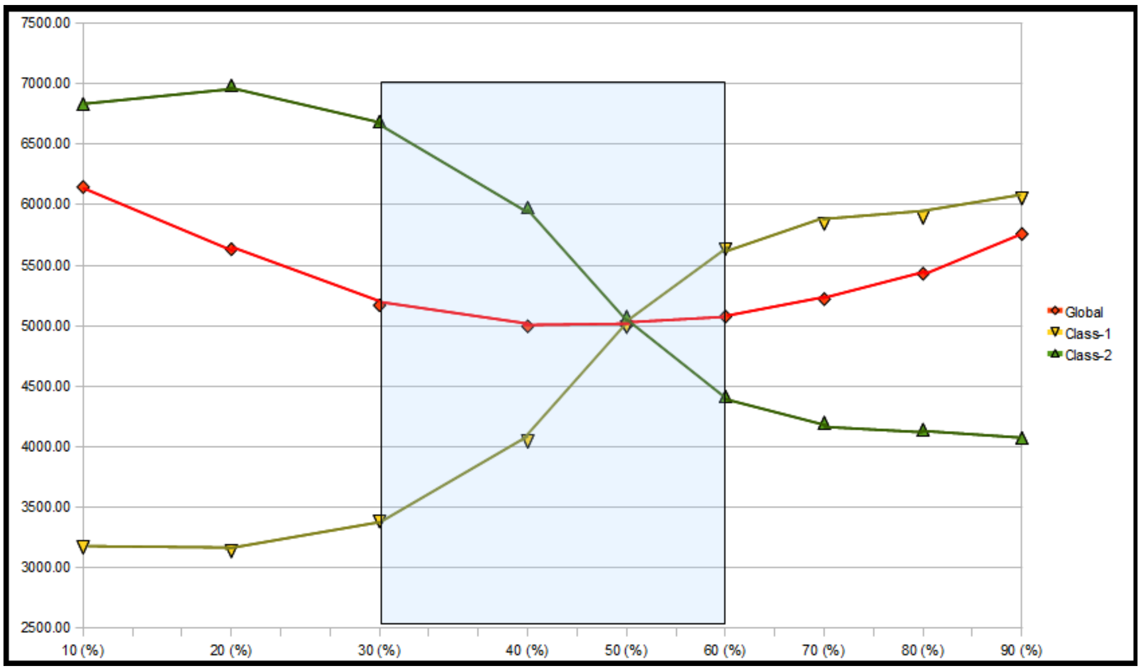


Figure 6.6: Simulation graph: response time

Table 6.4: Simulation system response time values

Class-1(%)	Global	Class-1	Class-2
10	6147.567768	3165.123871	6833.866859
20	5633.165951	3134.985289	6985.694241
30	5169.577336	3381.590296	6688.414854
40	4995.610733	4040.014477	5975.268473
50	5001.859845	4988.997857	5074.940096
60	5076.337129	5634.489243	4415.606436
70	5220.813767	5843.174367	4199.042470
80	5430.747119	5893.036246	4140.974379
90	5759.473705	6053.084228	4071.676114

methodologies were involved for the construction of the workload models, which have important effect on the algorthms under study. Behavior of real workload is very complex and difficult to reproduce, so we develop simple techniques to capture the static and dynamic behavior of the real load.

Since the study of this thesis is limited to two classes, we produce the load behavior through simple random number generator. When the goal is to generate workload having `50% class-A` and `50% class-B`, we create a `class-A` when the random number ≥ 5 and `class-B` when ≤ 5 in the scale of `(0-10)`. But this is not enough and sufficient to conclude about the benefits of the proposed algorithms, as one of the goals of this new techniques is to also handle unbalanced workload conditions. So we generate another workload where we define two process. Process A generates a certain percentage of `class-A , class-B` and Process B generates a different percentage of the same classes. Both the process have a certain switching probability as shown in Figure 6.7.

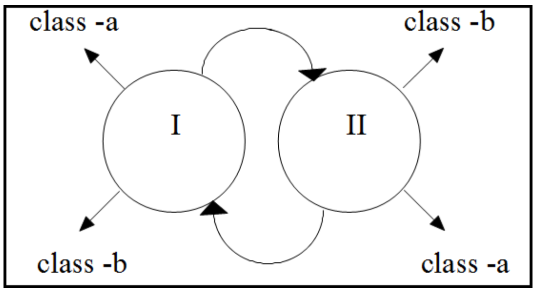
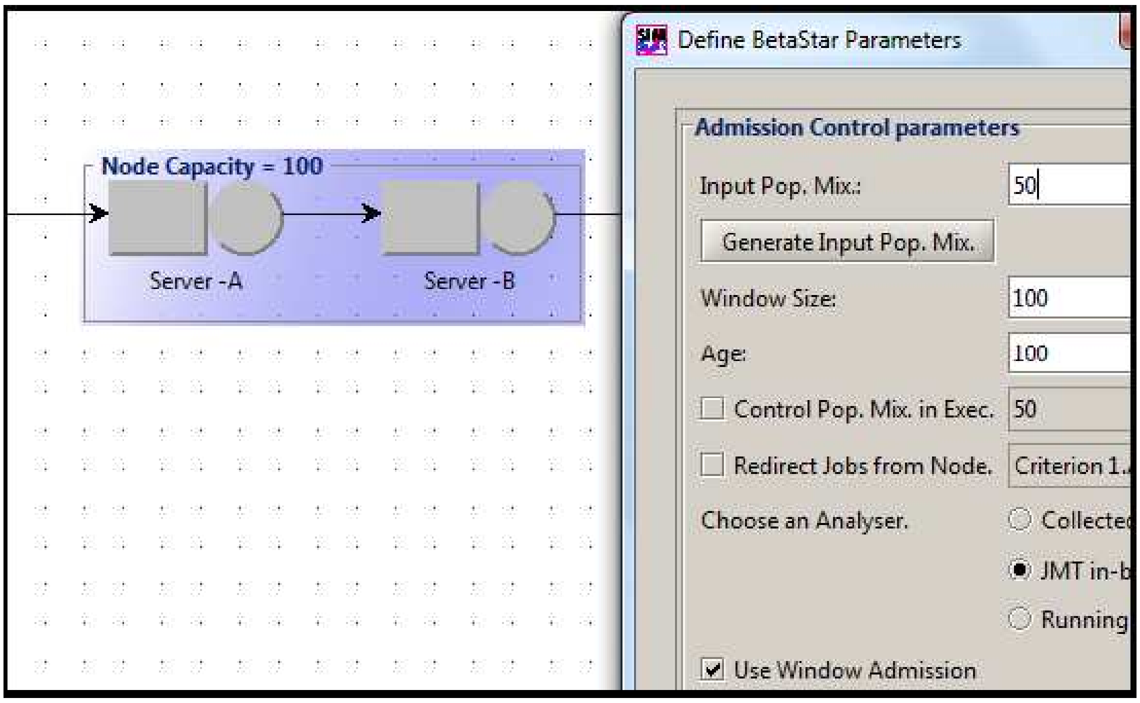


Figure 6.7: process I and II generate jobs of class a and b

6.2.1 Admission Policy

6.2.2 Case : balanced workload

In this case, we use balanced mixture of class `A` and class `B`. i.e. we generate static mixture of both the classes using the fixed random number generator as discussed above. We generate different models, for e.g. we first simulate the system with fixed `20%` of class`A` and then `40%` of class`A` and so on. The results are shown in the tabular 6.9 and 6.10.



Define BetaStar Parameters

Admission Control parameters

Input Pop. Mix.:	50
Generate Input Pop. Mix.	
Window Size:	100
Age:	100
☐ Control Pop. Mix. in Exec.	50
☐ Redirect Jobs from Node.	Criterion 1.
Choose an Analyser.	○ Collecte
	● JMT in-b
	○ Running

☑ Use Window Admission

Figure 6.8: Case 1 with Window =100, Timeout(Age) = 100

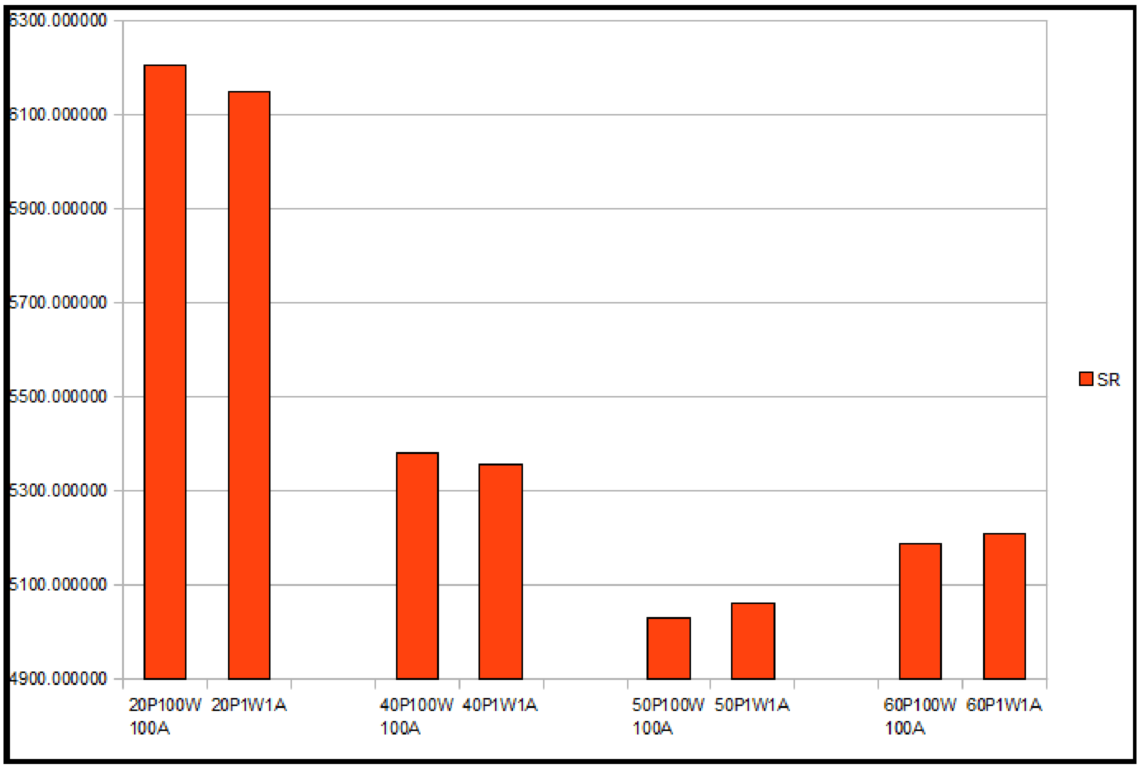


Figure 6.9: Response time comparision with window admission

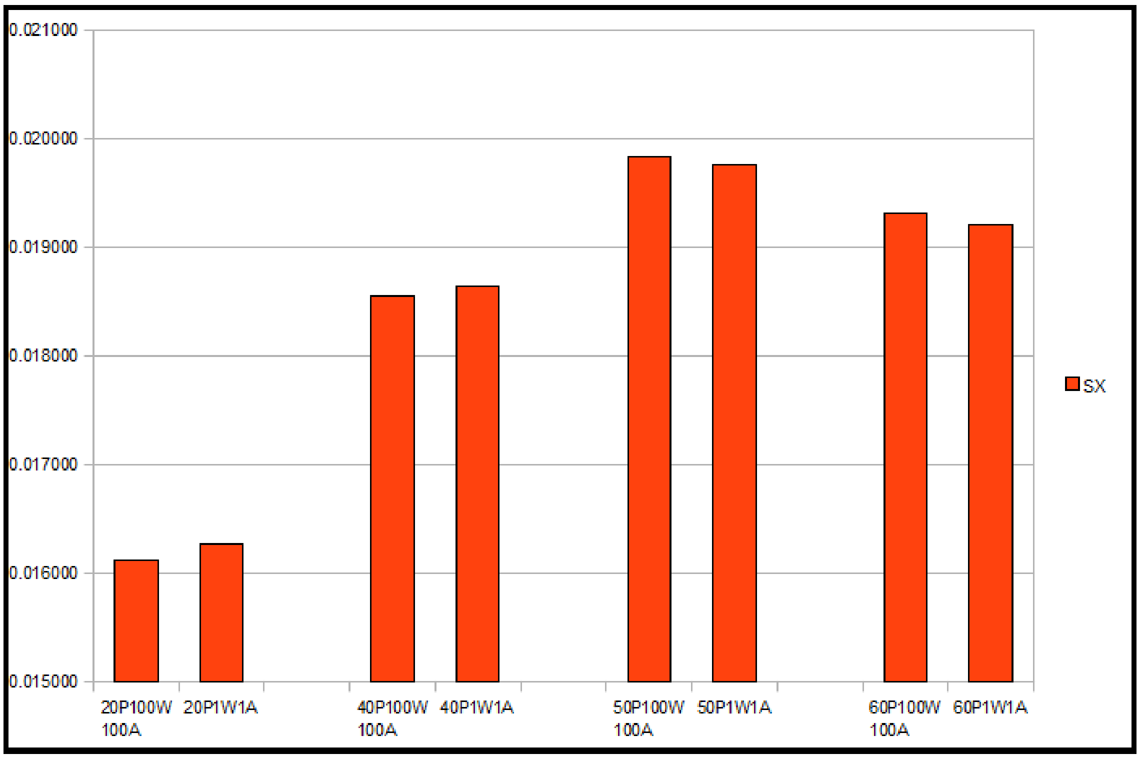
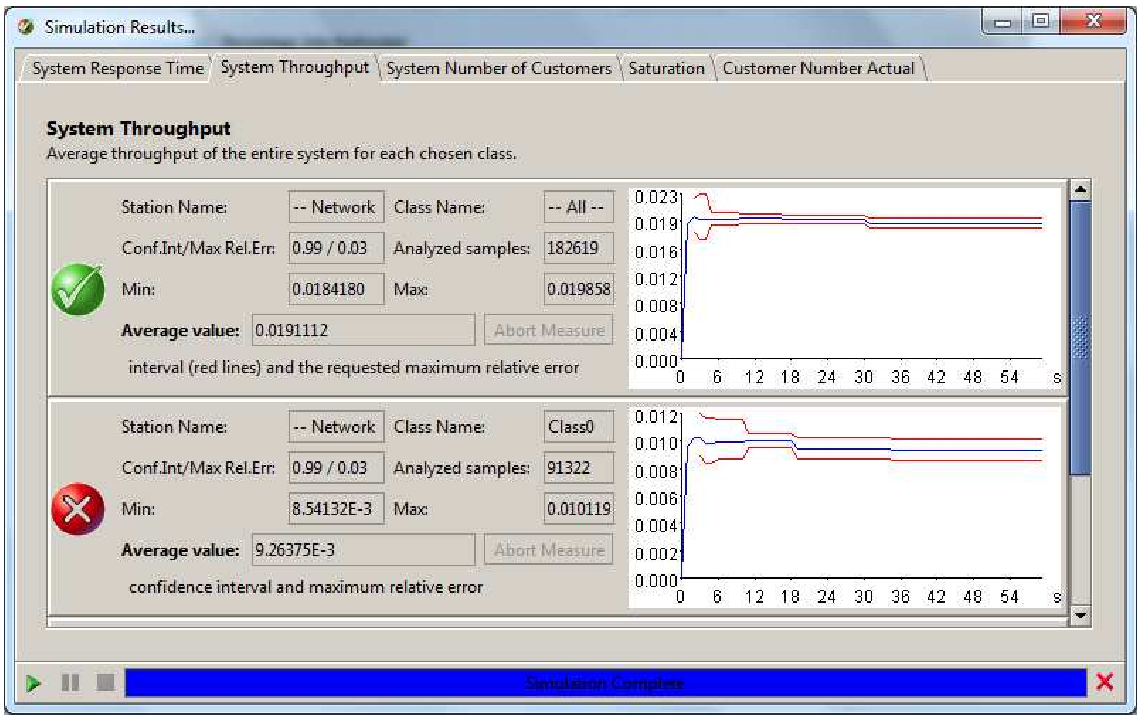


Figure 6.10: Throughput comparision with window admission


Simulation Results...

System Response Time | System Throughput | System Number of Customers | Saturation | Customer Number Actual

System Throughput

Average throughput of the entire system for each chosen class.

Station Name: -- Network
Class Name: -- All --
Conf.Int/Max Rel.Err: 0.99 / 0.03
Analyzed samples: 182619
Min: 0.0184180
Max: 0.019858
Average value: 0.0191112
Abort Measure
interval (red lines) and the requested maximum relative error

Station Name: -- Network
Class Name: Class0
Conf.Int/Max Rel.Err: 0.99 / 0.03
Analyzed samples: 91322
Min: 8.54132E-3
Max: 0.010119
Average value: 9.26375E-3
Abort Measure
confidence interval and maximum relative error

Simulation Complete

Figure 6.11: Simulation example

6.2.3 Case : unbalanced workload

Here we use a loading matrix of

$$L_{ir} = \begin{bmatrix} 100 & 1 \\ 1 & 100 \end{bmatrix} \tag{6.1}$$

Also we use two different process to generate mixture of classes. As seen in figure 6.7, the process A generates one of the classes at `90-99%` and the other process genarates the same class at `1-10%` probability. This results in a very unbalanced workload, where initially we have only one of the classes and later the other class. As seen in the figures 6.12 and 6.13, we can see that window mechanism coupled with the timeout concept, provides excellent improvement in the system response time and throughput. Hence here in this case we were able to demonstrate that using the presented concepts

Table 6.5: [SR] and [SX] with unbalanced input

Mode	[SR]	[SX]
Without Window	6658.7320	0.0150
W=100 Timeout = 100	6007.8014	0.0166
W=1000 Timeout = 1000	5208.0126	0.0191

Table 6.6: System response time with redirection

C-1(%)W=100, TO=100, RD=95	Global	Class-1	Class-2
20	5917.839009	3176.030423	6890.287610
40	5013.610755	4234.443032	5832.647553
50	5009.199387	4504.911331	5503.909721
60	5031.367791	4721.086050	5294.199805

and approach in the thesis, we were able to obtain good performance and behavior of unabalanced input load.

6.2.4 Admission with redirection

As seen from the previous section that Window mechanism is not sufficient to achieve enough improvement unless the network is very skewed and the input load has very high degree of burst in one of the class. Hence we had proposed the redirection mechanism where we redirect the jobs from the given node based on their timeout value. The test and evaluation of such results are shown in the figures 6.14 and 6.15.

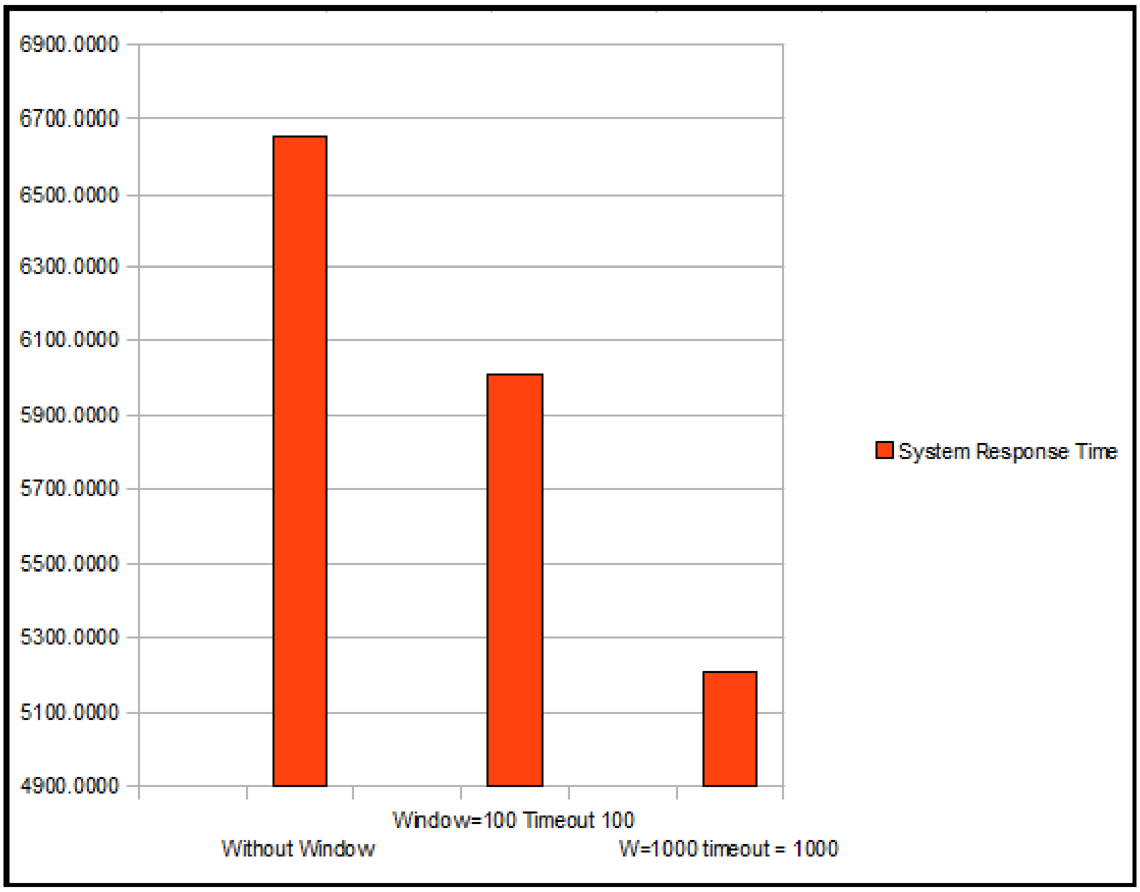


Figure 6.12: Throughput comparision with unbalanced workload

Table 6.7: System throughput with redirection

C-1(%)W=100, TO=100, RD=95	Global	Class-1	Class-2
20	0.016862	0.004317	0.012544
40	0.019938	0.010062	0.009905
50	0.020013	0.009994	0.009980
60	0.019882	0.009925	0.009962

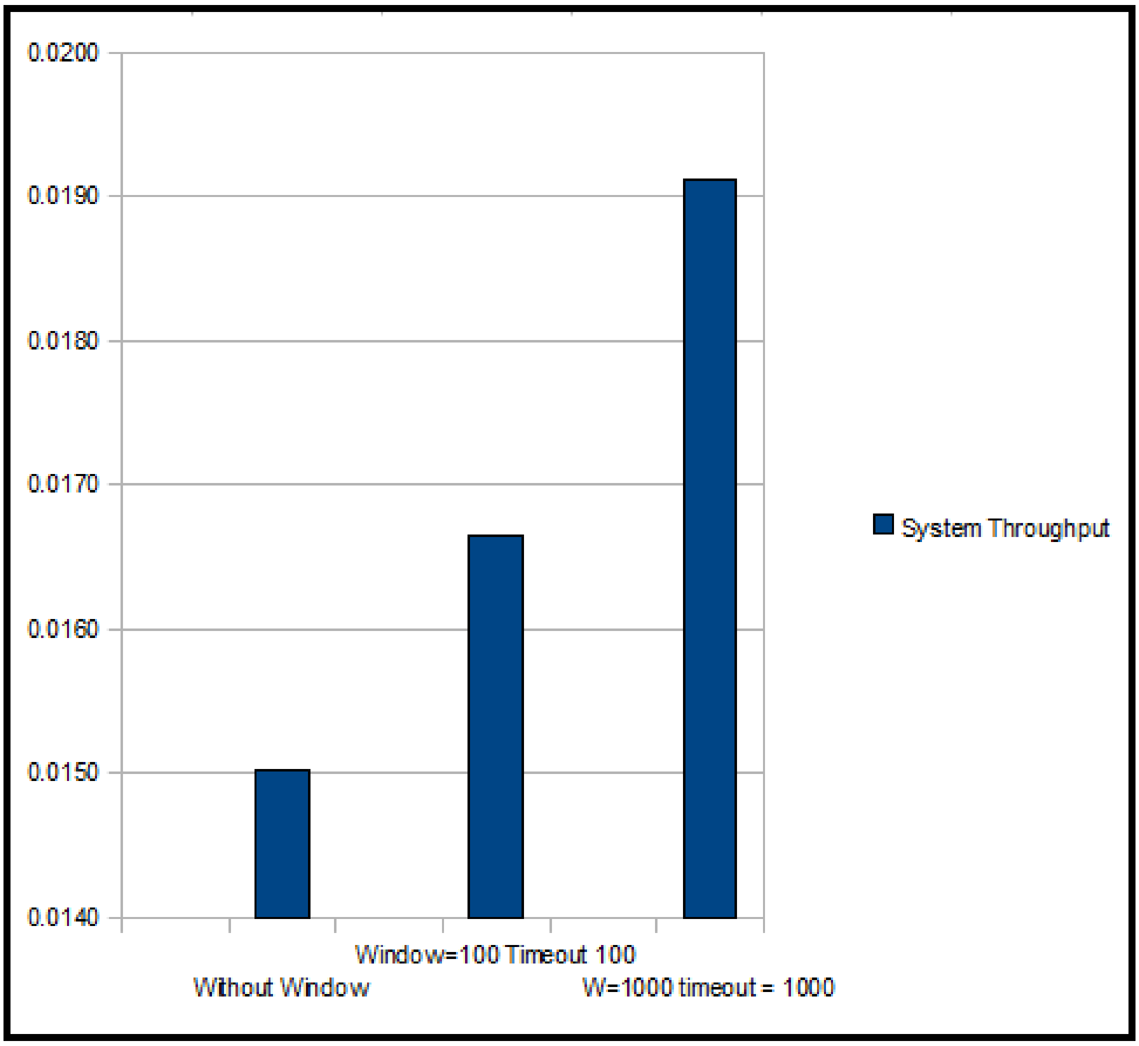


Figure 6.13: Response time comparision with unbalanced workload

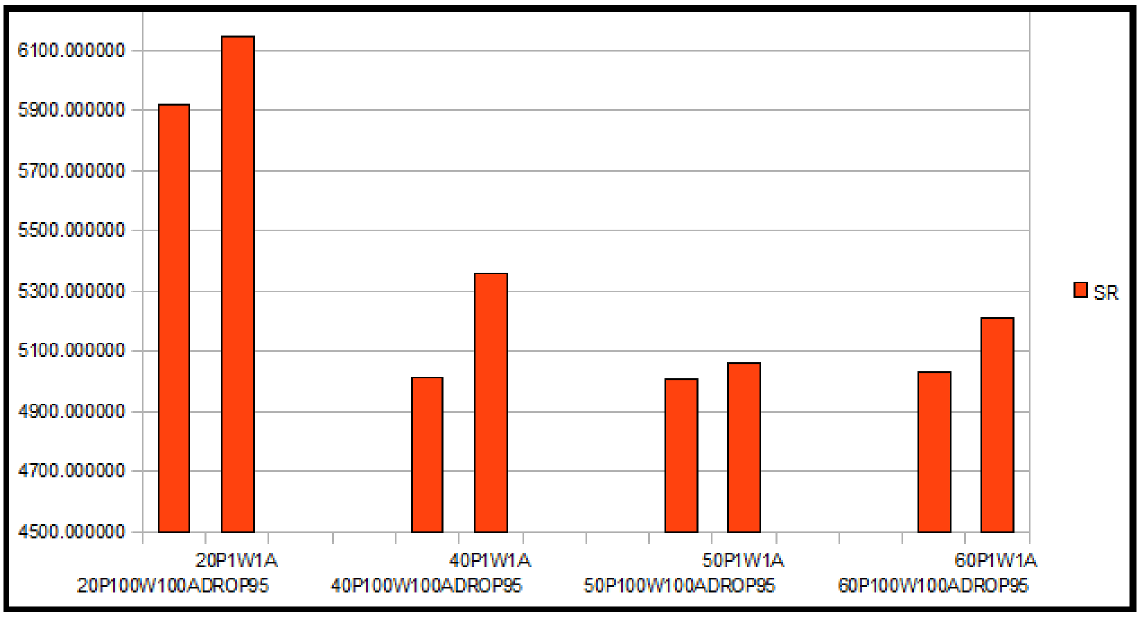


Figure 6.14: Simulation with redirection: system response time

We can summarize from the above results that, window mechanism is excellent to handle the unbalanced input load, when the mixture of the two classes is not even and highly skewed. On the other hand when the input load is well balanced with a good mixture of both the classes, then window and redirection provides us with excellent improve in the performance indices. Also it was measured from simulation results, that redirection obtained was low as `2%` when the input workload was `50%` for `class-1`. The results show us conclusive evidence of benefits of the proposed algorithms.

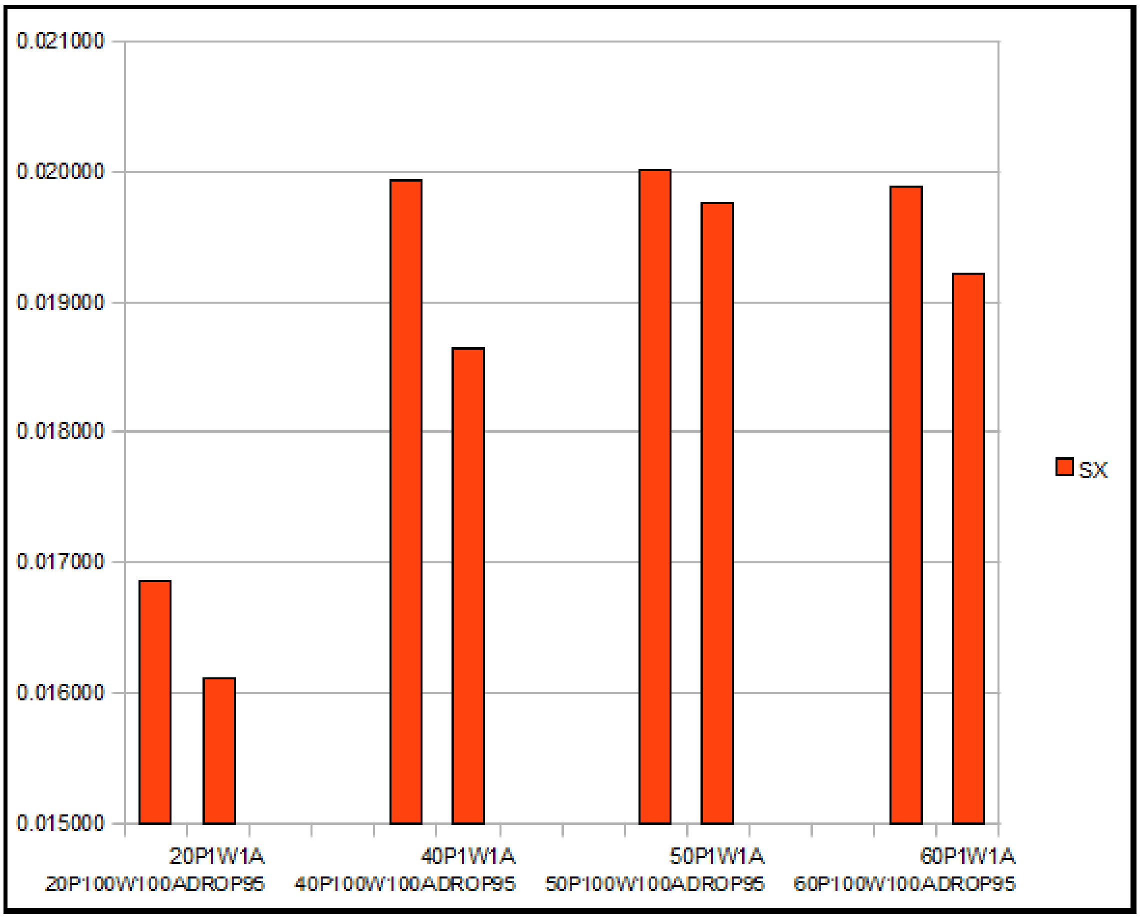


Figure 6.15: Simulation with redirection: system throughput

Chapter 7

Conclusions

In the thesis as presented contains a runtime configuration aware performance optimizing mechanism developed for a component based computing network. To summarize once more, the underlying idea was to selectively admit a given class of jobs, driven by the concepts of migration of bottleneck and common saturation sector to improve the overall throughput or response time of a given network. The capability of driving the new admission policy is achieved through the introduction of a window component that keeps the unfavorable incoming jobs waiting in a queue.
The use of a Queueing Network model was used to simulate the proposed idea and verify the theoretical increase in throughput and response time with various configuration and work load mix. Effort was given to maximize the total utility at a given Quality of Service.
As we have seen that during bottleneck migration, there is a switch between the throughput or response time of the different classes in the network, based on the mix in execution. We have seen that after a particular mixture the performance metrices

of a particular class dominates the other. This property of a network can used by window mechanism to favor a particular class and keeping the mixture of classes in the node to such a range so that a particular class of jobs always dominates the other. Also it has been seen that the global throughput and global response time is optimum when the system is in this saturation sector.

The new routing policy proposed in the thesis responds to class mix of a given switches servers in a given cluster when necessary. In addition, the admission control scheme deals with system overloading, which guarantees that the underlying system can respond to variable workload and unbalanced user requests. Performance evaluation has been done via simulation for an experimental workload. The results are compared with a system model that implements no special algorithm. Our experimental results show that the combination of the admission control scheme and the proposed redirection policy performs substantially better under various circumtances maintaining a certain Quality of Service. Also the system provides excellent improvement in performance for highly unbalanced system with unbalanced workload. Thesis has some limits:

- it does not consider the additional waiting time introduced for unfavorable job in computing the response time of a node.
- it does not consider the delay introduced by redirection and migration operations.
- it does not take the computational demand of a user request (and, consequently, the impact of its redirection on other servers) into account.

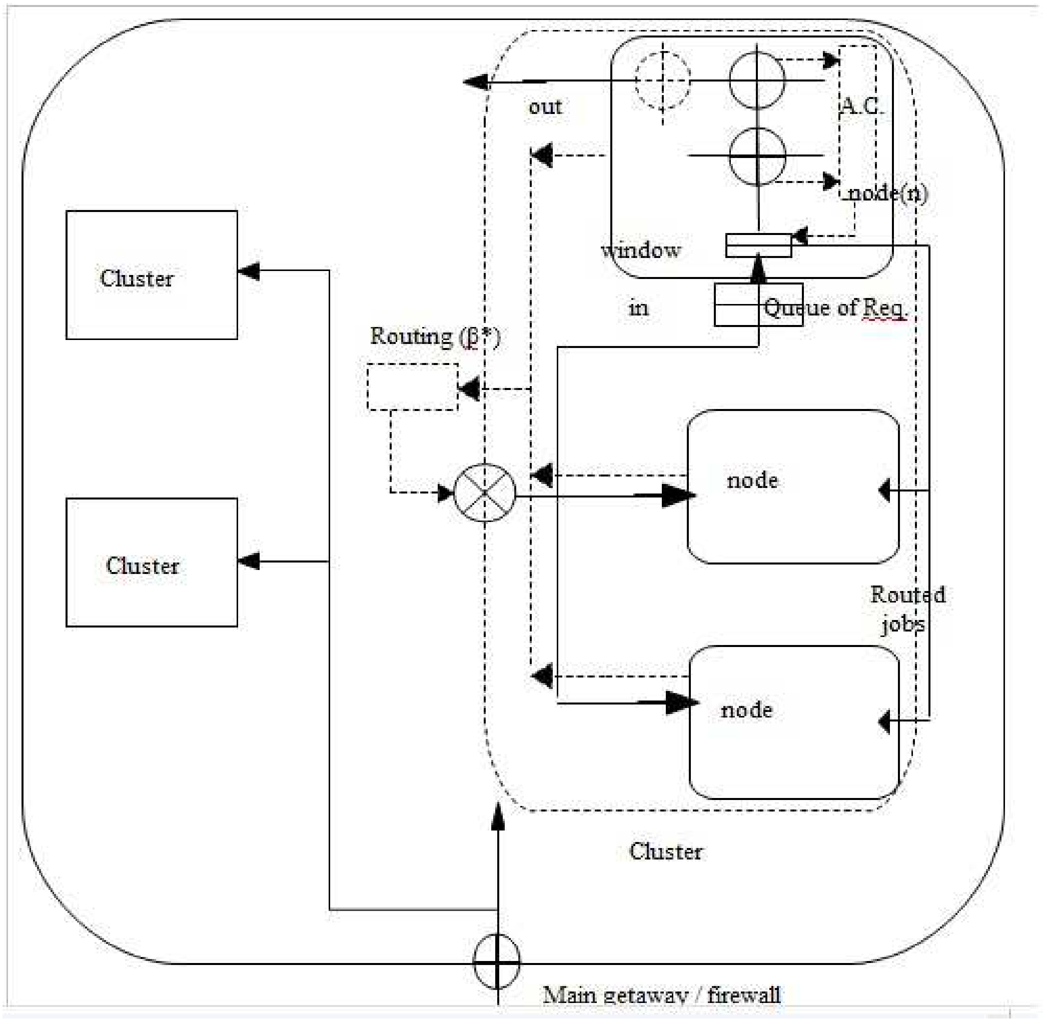


Figure 7.1: Overall Integration of all proposed methods

Chapter 8

Discussions and Future Work

After the advent of Web 2.0 and Web 3.0, Internet applications need large amount of resources and computation power to run their multi threaded **N**-tier architecture. Large data centers and data warehouses need to manage increasing user information and data. The mechanism of routing user requests to the correct application and data server and subsequent request re-direction and load balancing are based on standard routing protocols like Routing Information Protocol. Such algorithms developed from graph theory are simply based on computing least cost path or shortest path to the destination and is not sufficient to handle the temporary and unpredictable user demand.

In large Internet data centers where the applications are installed on multiple server nodes, the user [18] request cannot be routed to the various nodes in a static way and needs knowledge of run time server load. Whether request redirection can improve the user-perceived performance on real time and server conditions, remains an interesting and open issue. State of art proposals are typically based on thresholds. For example, double-thresholds on number of connection and network load are exploited in [8] to manage requests with the goal to reduce energy and network resource consumptions. Other studies, such as [1], propose request management techniques for large data centers that are based on a single threshold on the system load.

The methodology proposed in the thesis can be improved along several directions. One of the major area of research is extension of the policy to include multiple classes of requests on more than two stations. Also integration of the admission control policy with the equiutilization oriented routing, combined with the redirection of jobs gives a very powerful tool to fully realize the potential on a real time large scale server system. By simulation the effectiveness of the proposed algorithms, gives a capability to maintain the QoS targets for users while giving a strong leverage on the increase in the performance. Future works also include applying real time workload to the simulator to understand the behavior of performance metrics in order to enhance the mechanism.

SLA management, using the techniques discussed and developed in this thesis is one of the important area of future work. As already discussed in the Chapter 2 about workload characterization, if the input workload of a large network is functionaly divided on the lines of applications hosted, the window mechanism can provide higher SLA or performance metrics to a particular class of job in favor of the other. This gives the possibility of negotiating a different SLA and QoS with different client depending on the ability of paying for maintaining higher response time or throughput.

Another Routing policy developed during the thesis, but not based on common saturation sector and bottleneck migration was Load Dependent Routing. Load Dependent Routing is a routing where the packets are sent by a node to one of its forward nodes based on its own load and probability factor of the forwarding path. It requires knowledge of the network topology to make routing decisions. The network administrator configuring the routing rules should be aware of the cost of forward connections from the specific node and also the average service demand of a particular class of job on a specific node.

Please refer figure 8.1 where there is a Firewall which is connected to 3 nodes in a given network. A node can be a very fast computer or cluster of computers or a cloud on the internet, basically any processing unit connected to an I/O device. The user requests are intercepted by the firewall node and then forwarded to the specific based on a specific routing policy.

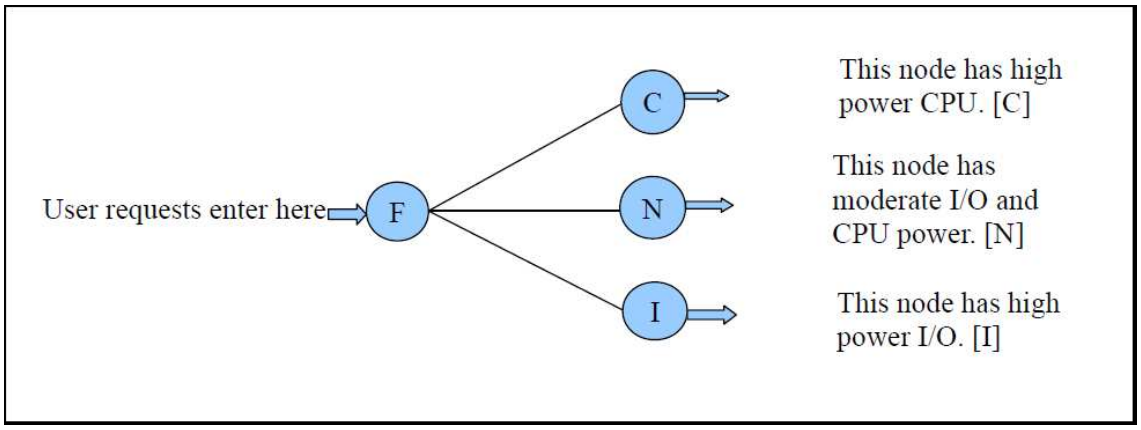

This node has high power CPU. [C]
C
User requests enter here
F
N
This node has moderate I/O and CPU power. [N]
I
This node has high power I/O. [I]

Figure 8.1: Routing based on load of Firewall F

The Node [C] has a very fast CPU and it can do complicated simulation or complex mathematical equations very quickly. But this node also consumes lot of resources, power and usage of this node costs the organization a lot of money. The Node [N] has moderate CPU and I/O. This node can be assumed to be in house data center and processor of the organization and usage of this node does not cost the company much and is normally used for day to day business. The final node has very high powered I/O bus connected to the memory and hence it is excellent in writing and reading data from the Hard Disc. This node is generally used for mass data upload and download.

As a network administrator, on normal load circumstances, we want to use the node [N] for day to day activity but during load burst and high demand we want to use the node [C] or node [I] to meet the user requirement. This arrangement is beneficial as extensive use of the high powered node [C] and [I] can cost the organization a lot. Load Dependent Routing is specifically useful in above described scenario. Here the network admin can set rules where the firewall can route request to the node [N] on day to activity, but during specific high demand or load, can route the CPU bound request to the node [C] and I/O bound request to the node [I].

Load Dependent Routing can be used to improve the Response Time or decrease the Response Time as desired by the administrator of the particular network to achieve benefits in power consumed. It can be used to specifically favor a particular class of jobs which have higher priority to system resources than other class of jobs. The algorithm can also be used to selectively drop network packets of particular class to give priority to the other class of jobs. This algorithm along with the developed algorithm in the thesis, provides an excellent future research area in controlling the request admission in a large scale server center to improve the performance of the system.

A variety of admission control mechanisms have been proposed in various journals to manage web servers, most of which have focused on overload prevention. Elnikety et al. [16] developed a DB proxy that managed load on the DB server by performing admission control on DB queries being sent from the second tier application server to the third tier DB server, but did not report the effect this had on the content of the pageview seen at the web browser. Cherkasova et al. [13] performed admission control with the goal of maximizing the number of successful sessions and limiting the amount of resources wasted on rejected sessions.

Study and analysis of bottlenecks is in itself a large topic of discussion. As discussed in [17] for mission-critical applications such as e-commerce, N-tier systems have grown in complexity with non-stationary workloads and inter-task dependencies created by requests that are passed between the various servers. These complicating factors create multi-bottlenecks such as oscillatory bottlenecks (where inter-task dependencies cause the bottleneck to migrate among several resources) and concurrent bottlenecks (where multiple bottlenecks arise among several resources). Multi-bottlenecks are non-trivial to analyze, since they may escape typical assumptions made in classic performance analysis such as stable workloads and independence among tasks. The analysis done in this paper [17] also shows experimental evidence of multi-bottleneck cases where several resources saturate alternatively.

In summary, the thesis potrays the implementation of some proposed algorithm using the concepts of bottleneck migration and saturation sector. The scope of improvement among different direction are immense, as hunderds and thousands of different configurations have to be evaluated to arrive at the best possible decision. In general performance optimization requires huge analysis, as every network serves a different need, every configuration has different operating parameters, and every system can react in a unique and unpredictable way to performance tweaks. The ideas presented in the thesis, opens new concepts and explores different techniques in the vast field of performance and optimization in computer networks.

Bibliography

[1] M. Andreolini, S. Casolari, and M. Colajanni. Autonomic request management algorithms for geographically distributed internet-based systems. In *2nd IEEE International Conference on Self-Adaptive and Self-Organizing Systems (SASO08)*, 2008.

[2] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. H. Katz, A. Konwinski, G. Lee, D. A. Patterson, A. Rabkin, and M. Zaharia. Above the clouds: A berkeley view of cloud computing. Technical report, 2009.

[3] G. Balbo and G. Serazzi. Asymptotic analysis of multiclass closed queueing networks: multiple bottlenecks. *Performance Evaluation*, 30(3):115–152, 1997.

[4] J. P. Buzen. Computational algorithms for closed queueing networks with exponential servers. *Commun. ACM*, 16(9), 1973.

[5] M. Calzarossa and G. Serazzi. Workload characterization: A survey. *IEEE Proceesings Special Issue on Computer Performance Evaluation*, pages 1136–1150, August 1993.

[6] G. Casale and G. Serazzi. Bottlenecks identification in multiclass queueing networks using convex polytopes. In *MASCOTS*, pages 223–230, 2004.

[7] K. M. Chandy and D. Neuse. Linearizer: A heuristic algorithm for queuing network models of computing systems. *Commun. ACM*, 25(2), 1982.

[8] G. Chen, W. He, J. Liu, S. Nath, L. Rigas, L. Xiao, and F. Zhao. Energy-aware server provisioning and load dispatching for connection-intensive internet services. *USENIX Symposium on Networked Systems Design and Implementation (NSDI2008)*, 5(5), 2008.

[9] P. Cremonesi, S. P. J., and G. Serazzi. A unifying framework for the approximate solution of closed multiclass queueing networks. *IEEE Trans. Comp.*, 16(9), 2002.

[10] T. Li, D. Baumberger, and S. Hahn. Efficient and scalable multiprocessor fair scheduling using distributed weighted round-robin. *SIGPLAN Not.*, 44:65–74, February 2009.

[11] M. Litoiu. A performance analysis method for autonomic computing systems. *ACM Trans. Auton. Adapt. Syst.*, 2(1):3, 2007.

[12] J. Lizy, Kurian, V. Purnima, and S. Jyotsna. Workload characterization: Motivation, goals and methodology. *Workload Characterization, Annual IEEE International Workshop*, 0:3, 1998.

[13] C. Ludmila and P. Peter. Session based admission control: a mechanism for improving performance of commercial web sites. *IWQoS*, page 226235, May 1999.

[14] K. E. E. Raatikainen. Cluster analysis and workload classification. *SIGMETRICS Perform. Eval. Rev.*, 20:24–30, May 1993.

[15] M. Reiser and S. Lavenberg. Mean-value analysis of closed multichain queueing networks. *Journal of the ACM*, 27(2), 1980.

[16] E. N. S. Elnikety, T. J., and Z. W. A method for transparent admission control and request scheduling in e-commerce web sites. *WWW2004*, page 276286, 2004.

[17] M. Simon, H. Markus, and P. Calton. Experimental evaluation of n-tier systems: Observation and analysis of multi-bottlenecks. *IEEE International Symposium on Workload Characterization*, pages 118–127, 2009.

[18] J. W. J. Xue, A. P. Chester, L. He, and S. A. Jarvis. Model-driven server allocation in distributed enterprise systems. 2009.

[19] J. Zahorjan, D. L. Eager, and H. Sweillam. Accuracy, speed, and convergence of approximate mean value analysis. perform. eval. *Performance Evaluation*, 8(4):255–270, 1988.