Managing Load Balancing, Energy Efficiency and Performance of Cloud Data Centers with Service Level Agreement Guarantees

Von der Fakultät für Informatik, Elektrotechnik und Informationstechnik der Universität Stuttgart zur Erlangung der Würde eines Doktor-Ingenieurs (Dr.-Ing.) genehmigte Abhandlung

vorgelegt von

Maggie Mashaly

geb. in Kairo

Hauptberichter:	Prof. Dr.-Ing. Dr.-Ing. E. h. Dr. h. c. Paul J. Kühn
1. Mitberichter:	Prof. Dr.-Ing. Hermann de Meer
2. Mitberichter:	Prof. Dr.-Ing. Andreas Kirstädter

Tag der Einreichung:	10. Mai 2017
Tag der mündlichen Prüfung:	26. Oktober 2017

Institut für Kommunikationsnetze und Rechnersysteme der
Universität Stuttgart
2017

Abstract

This thesis proposes novel approaches for enhancing the operation and performance of cloud data centers by introducing an algorithm for automatic server consolidation for an energy efficient operation as well as two algorithms for load balancing between several data centers with priority to maintaining users' Service Level Agreements 'SLAs'. For better understanding of the proposed algorithms and fore-sighting their effect on data center's performance a modeling approach for data centers as queuing systems is first introduced where a data center is modeled by a Markov Chain and its performance can be analyzed exactly by solving the two-dimensional Markov Chain representing all data center's states under Markovian assumptions.

This thesis' first contribution is the automatic server consolidation algorithm which is able to adapt the number of active servers in the data center to its current load dynamically by adopting the hysteresis behavior. This is performed by setting defined queuing thresholds which trigger server activations only when they are reached and if the SLA is guaranteed, which reduces the frequency of servers' activations and deactivations and consolidates the work load on the lowest possible number of servers such that idle servers can be turned-off or put into sleep for reducing power consumption. This algorithm is novel in its ability to model all realistic aspects of a data center such as the activation time required by servers to re-boot from an off state or wake-up from a sleeping state as well as modeling different sleep states (C-states) and reduced frequency states specified by servers' various P-states, respectively. Besides providing mathematical analysis for the algorithm and data center model, a simulation model is also provided to support the findings of the analysis and to test several cases that cannot be solved by analytical solution due to its restrictive -yet valid- assumptions.

Second main contribution of this thesis is the two load balancing algorithms introduced for optimizing data centers' performance and preventing over-load situations while maintaining service level agreements of users. The two algorithms can be adopted by data centers at different scenarios depending on the type of services offered by the data center, requirements of its users and the geographical distribution of data centers between which load balancing is implemented. The first proposed algorithm (Local Server System First - LSSF) tends to suppress migrations only for arriving requests that could not be served at their local data centers and migrates them to another data centers where their service level agreements can be met. The second proposed algorithm (Shortest Response Time First – SRTF) aims at providing arriving requests with the least possible response time by routing them to the DC providing immediate service or least waiting time. Both algorithms are modeled by Markov Chains and solved analytically for predicting the effect of each algorithm on data centers' performance. For further analysis of both algorithms simulation models are implemented as well as test-bed experimentation on a small data center in order to test the algorithms under various test-cases and conditions.

Results obtained from different platforms analyzing the proposed algorithms in this thesis have proven their efficiency at reducing the power consumption of data centers at low to intermediate load conditions as well as balancing the load among several data centers to meet defined criteria by data center operators or users up to high load conditions. Analysis of the proposed hystereses algorithm for energy efficiency has shown a significant reduction of servers' activation rates resulting in a reduction in power consumption with bounded average delays for delayed arrivals with a negligible increase in the delay at load regions between 5-95%. Efficiency of the load balancing algorithms is also shown by a reduction in loss probabilities of arriving requests by migrating them to foreign DCs as well as a reduction in average delays of delayed requests in case of SRTF algorithm. Thus, data centers' administrators can choose to implement one or more of the proposed algorithms depending on their defined goals for the data center's operation where its behavior can be exactly predicted using analytical and simulation tools introduced in this thesis.

Kurzfassung

In dieser Arbeit werden neue Ansätze zu einer effektiveren Betriebsorganisation von Cloud-Rechenzentren vorgeschlagen basierend auf Algorithmen zur automatischen Server-Aktivierung ("Konsolidierung") für einen energieeffizienten Betrieb sowie zwei neue Algorithmen zum Lastausgleich zwischen Server-Gruppen unter gleichzeitiger Berücksichtigung vorgegebener und einzuhaltender Dienstleistungsparameter (sog. "Service Level Agreements", SLA). Zum besseren Verständnis der vorgeschlagenen Algorithmen und ihren zu erwartenden Auswirkungen auf die Leistungsfähigkeit von Rechenzentren werden diese mit Hilfe der Warteschlangentheorie modelliert und mittels der Methodik von zweidimensionalen Markoff-Ketten exakt untersucht.

Im ersten Beitrag dieser Arbeit wird ein Algorithmus zur automatischen Server-Konsolidierung untersucht, bei dem die Anzahl aktivierter Server an die momentane Last dynamisch mittels eines zustandsabhängigen Hysteresemechanismus angepaßt wird. Dabei werden die Schwellwerte zur Server-Aktivierung so eingestellt, daß eintreffende Server-Anforderungen gepuffert werden unter Einhaltung der vorgegebenen SLA-Einschränkungen, wobei erst nach Erreichen des Schwellwertzustands eine neue Server-Aktivierung erfolgt. Dieses Verfahren reduziert die Häufigkeit von Server-Aktivierungen ("Bootings") bzw. Deaktivierungen zum Übergang in einen Ruhezustand ("Sleeping") und trägt auf diese Weise zu einem optimierten Energiebedarf bei. Dieser neuartige Algorithmus erlaubt ferner die Einbeziehung realistischer Aspekte eines Rechenzentrums-Managements hinsichtlich des Energiebedarfs mittels gedrosselter Werte von Versorgungsspannungen und Taktfrequenzen der elektronischen Bauelemente ("Dynamic Voltage and Frequency Scaling", DVFS; P-Zustände) bzw. Schlafzustände (C-Zustände). Außer der mathematischen Analyse dieses Algorithmus für das Rechenzentrum wird ein Modell zur Computer-Simulation entwickelt, mit Hilfe dessen auch Anwendungsfälle untersucht werden können, die der exakten mathematischen Analyse nicht zugänglich sind.

Im zweiten Beitrag dieser Arbeit werden zwei Algorithmen zum Lastausgleich zwischen Server-Gruppen eingeführt zur Untersuchung des Echtzeitverhaltens sowie des Schutzes gegen Überlastsituationen, welche ebenfalls unter Einhaltung der SLA-Einschränkungen der Nutzer operieren. Derartige Algorithmen können für Rechenzentren (bzw. Server-Gruppen) in unterschiedlichen Anwendungs-Szenarien hinsichtlich von Benutzeranforderungen oder hinsichtlich des Lastausgleichs unterschiedlicher geographischer Lagen dieser Rechenzentren eingesetzt werden. Beim ersten Algorithmus LSSF ("Local Server System First") werden Prozeß-Verlagerungen ("Migrations") zu einem zweiten Rechenzentrum bzw. einer zweiten Server-Gruppe nur in dem Falle durchgeführt, wenn die Anforderungen nicht im lokalen Rechenzentrum ausgeführt werden können solange jedoch die SLA-Einschränkungen weiterhin eingehalten werden. Der zweite Algorithmus SRTF ("Shortest Response Time First") zielt darauf ab, in jedem Falle die kürzeste Fertigstellungszeit (Antwortzeit) zu garantieren, indem eintreffende Anforderungen demjenigen Rechenzentrum bzw. derjenigen Server-Gruppe zugeordnet werden, welche entweder eine sofortige Bearbeitung erlauben oder die kürzeste Antwortzeit benötigen. Die Modelle der beiden Verfahren werden mathematisch exakt mittels zweidimensionaler Markoff-Ketten beschrieben und hinsichtlich der Vorhersage ihrer Leistungsfähigkeit analysiert. Zur allgemeineren Analyse beider Algorithmen wurden jeweils Simulationsmodelle implementiert sowie in einem experimentellen Server-Testbed einer Multiprozessor-Konfiguration konfiguriert, um die vorgeschlagenen Algorithmen testen zu können.

Contents

List of Figures

List of Tables

Abbreviations and Symbols

Abbreviations

APP	Application Specific Software
BMC	Baseband Management Controllers
CAPEX	Capital Expenditure
CPU	Central Processing Unit
CRAN	Cloud Radio Access Networks
CSB	Cold Stand-by Mode
DAS	Directly Attached Storage
DC	Data Center
DoS	Denial of Service
DPM	Dynamic Power Management
DRS	Dynamic Resource Scheduler
DT-PALB	Double Threshold Energy Aware Load Balancing
DVFS	Dynamic Voltage and Frequency Scaling
FC	Fiber Channel
FIFO	First In First Out
FSM	Finite State Machine
HSB	Hot Stand-by Mode
IaaS	Infrastructure as a Service
ICT	Information and Communication Technology
INI	Initialization File
I/O	Input / Output
IT	Information Technology
iSCSI	Internet Small Computer System Interface
JSQ	Join Shortest Queue
KVM	Kernel Virtual Machine
LBMM	Load Balancing Min-Min Algorithm
LBIMM	Load Balancing Improved Min-Min Algorithm
LSSF	Local Server System First

LUNs	Logical Unit Numbers
NAS	Network Attached Storage
NED	Network Description File
OLB	Opportunistic Load Balancing Algorithm
OPEX	Operational Expenditure
OS	Operating System
PA-LBIMM	User-Priority Aware Load Balancing Improved Min-Min Algorithm
PALB	Power Aware Load Balancing Algorithm
PaaS	Platform as a Service
PUE	Power Usage Efficiency
QoS	Quality of Service
QoE	Quality of Experience
RAM	Random Access Memory
SAN	Storage Area Network
SaaS	Software as a Service
SCSI	Small Computer System Interface
SLA	Service Level Agreement
SRTF	Shortest Response Time First
VLAN	Virtual Local Area Network
VM	Virtual Machine
VMM	Virtual Machine Monitor
VRF	Virtual Routing and Forwarding
WCAP	Workload and Client Aware Policy

Symbols

A	Carried Traffic
B	Loss Probability
C	Sum of Capacitances in Server's circuit
$\boldsymbol{C_{ij}}$	Condition of migration of requests from $\boldsymbol{DC_i}$ to $\boldsymbol{DC_j}$
d	Defect value
$\mathrm{E}[T_W]$	Mean waiting time of arriving frames
$\mathrm{E}[T_W \mid T_W > 0]$	Mean waiting time of delayed frames
f	Server's operational frequency
I	Probability of immediate service of requests at a DC
L	Mean queue length
M	Migration probability
n	Total number of servers
P	Server's Power Consumption
$\mathrm{P}(i)$	Probability of having i active servers
$\boldsymbol{p(x,z)}$	Probability of being in state (x,z)
P_{DVFS}	Power consumed by a server under DVFS strategy
$P_{running}$	Power consumed by a running server
P_s	Power consumption of servers inside the DC
$P_{s,CSB}$	Power consumption of servers in CSB mode
$P_{s,HSB}$	Power consumption of servers in HSB mode
P_{static}	Power consumed due to leakage mechanisms
P_x	Server's P-state
$\mathrm{Q}(z)$	Probability of having z occupied buffers
R_A	Activation rate of servers
R_D	Deactivation rate of servers
REL	Relaxation Factor for Gauss-Seidel method
s	Total DC's buffer capacity
$\boldsymbol{t_M}$	Migration time of a request to a foreign DC
$\boldsymbol{t_{w,T}}$	Request's delay threshold defined by SLA
$\boldsymbol{t_{w0}}$	Worst-case mean delay of an arriving request
u	Server's operational voltage

W	Delay Probability
$\boldsymbol{w}^{(x)}$	Width of the hysteresis for activating server x
w_i	Width of the ith hysteresis in the Multiple Serial/Parallel Model
x	Number of occupied servers in a DC
$\boldsymbol{x_i^*}$	Migration threshold of $\boldsymbol{DC_i}$ under LSSF algorithm
$\boldsymbol{Y_A}$	Average number of servers in activation phase
z	Number of occupied buffers in a DC
$\boldsymbol{z}^*$	Buffering threshold for application of DVFS strategy
α	Server activation rate
λ	Arrival rate of requests
$\boldsymbol{\lambda(Q_1)}$	Arrival Conditions for $\boldsymbol{DC_1}$ under SRTF strategy
$\boldsymbol{\lambda(Q_2)}$	Arrival Conditions for $\boldsymbol{DC_2}$ under SRTF strategy
μ	Service rate of requests
μ^*	Reduced service rate of servers under DVFS strategy
ρ	System load
η	Power-saving efficiency
$\boldsymbol{\eta_{CSB}}$	Power-saving efficiency of CSB mode
$\boldsymbol{\eta_{HSB}}$	Power-saving efficiency of HSB mode
ε	Threshold for Gauss-Seidel Method

Chapter 1 Introduction

The ICT industry has always been a field with huge potential for technological developments aiming at providing individuals as well as industry with services for raising their welfare and surging their business development. Cloud Computing is one of those developments that have been widely adopted by millions of users and companies world-wide as it provided a leap at how different types services are offered for users. Based upon the concept of virtualization, Cloud Computing offered different types of services for its users ranging from Infrastructure-as-a-Service allowing them to rent whole data center infrastructures, to Platform-as-a-Service where users are allowed access to any desired platform they require and finally to Software-as-a-Service which provides numerous services provided by applications hosted at cloud data centers and accessed remotely by users anywhere in the globe. Many reasons have caused all cloud users to embrace this new technology; among these reasons is its reduced cost resulting from eliminating the need for upfront investments and its attractive pay-on-the-go model where users are only charged for their usage of the services rather than being charged with costs of idle operation. Another reason is cloud's elasticity and flexibility at leasing or abandoning services and resources on-the-go according to current needs. Cloud Computing did not appeal only to users, but also to owners of data centers as they could rent their infrastructure or host services upon them at times when their equipment is not utilized.

However the rise of Cloud Computing technology has led to several challenges faced by its users and providers. For users the dilemma of data security whether it is hosted at a public, private or hybrid cloud has been and still is an open question. As for data center owners the challenges faced by how data centers are operated are critical. Since data centers are typically composed of huge numbers of server blades hosting hundreds of servers upon which cloud services are hosted, the amount of power required for operating and cooling these devices is tremendous. This greedy consumption of power by data centers is alarming as it increases the carbon footprint by consuming non-renewable energy resources and most importantly, increases the cost of operating data centers which will result in an increase in the cost of cloud service thus depriving cloud computing of one of its most important advantages, and most importantly as it is forecasted to increase rapidly over the coming years.

Numerous approaches have been adopted for attempting to reduce the power bill of cloud computing data centers, such as placing data centers at Polar Regions where extremely low temperatures allow reducing or eliminating the energy required for infrastructure cooling. However reducing the cooling costs is not the optimum solution as it only contributes to a small fraction of the total power consumption which is dominated by operational power of servers. To reduce the power consumed by data centers' servers several solutions have been proposed such as server consolidation, sleep modes and Dynamic Voltage and Frequency Scaling. The first approach of server consolidation tends to consolidate the work load of the data center into a fewer number of active servers in order to reduce the number of operational servers and accordingly power consumption. The second approach of sleep modes suggests putting idle servers into sleep modes where they consume much lower power compared to idle consumption and can wake up from sleep states faster than re-booting. Various sleep states depend on which components of servers are put to sleep, where deeper sleep states consume lower power by putting more server components into sleep. The third approach of Dynamic Voltage and Frequency Scaling specifies several P-states for servers where a server can reduce its operation frequency by a certain factor specified by each P-state which is directly proportional to a reduction in power consumption. All these approaches are briefly discussed and tested within later chapters in this thesis.

Another challenge faced by cloud data centers' operators is to find efficient load balancing approaches that accommodates the nature of cloud computing services, fulfill the requirements of cloud users and satisfy their specified service level agreements while enhancing data centers' performance via load balancing. Despite the existence of several load balancing approaches in literature that are applied to various systems successfully, most of them could not offer efficient load balancing solutions for cloud computing for being static, having single points of failure, requiring long transmission delays or long processing delays due to complexity. Cloud Computing however require algorithms that are dynamic with automatic resource provisioning for fast adaptation to any updates in the data center's state, proactive to over-load situations by preventing them from occurring, distributed without centralized points of failure to avoid relying the data center's operation on a single node, and finally and most importantly perform load balancing decisions without any compromise to users' specified Service Level Agreements (SLA).

1.1 Problem Overview

Since it was first introduced in 2007 [148], Cloud Computing has taken over the IT industry worldwide with a continuous evolution as the demand for its services is highly increasing. Numerous startup companies as well as well-established ones nowadays are starting to migrate their infrastructures and services into the cloud. This is a result of the numerous benefits that cloud computing provides; few of them are reduced costs, ease of management, and a cost-effective on-demand policy that allows for dynamic provisioning of resources with minimal upfront investments for customers. However such attractive advantages of the rising technology come with a price tag. Cloud service providers establish and maintain huge data centers for providing cloud services, and these data centers are considered major consumptions of energy in the ICT field. According to studies and measurements reported in [34] and [81] the amount of energy consumed for operating and cooling data centers is estimated to be between 1.1 and 1.5% of the total energy consumption world-wide. An increase in this percentage will cause the energy costs to increase and accordingly the cost of cloud services, thus depriving cloud services of one of their critical advantages. Another challenge faced by cloud operators is a side effect for the main technology behind cloud computing; i.e. Server Virtualization. Although virtualizing servers allows for sharing hardware resources among several customers and thus an economic operation of data centers, a dynamic allocation strategy is of high importance to avoid overloading machines with workload that can affect their performance or result in server hotspots that cannot be handled effectively by cooling systems and could result in server downtime [160]. Accordingly the need for energy efficient operations in cloud data centers is crucial for maintaining and evolving Cloud Computing technology.

Besides the energy efficiency problem in cloud data centers, balancing the load among several data centers is one of the major factors affecting their performance. As cloud providers offer their services to clients typically via several DCs located at various geographical locations for satisfying clients' prescribed Service Level Agreements (SLAs), decisions on where to route each incoming request has a significant impact on the DC's performance as it could lead to overload at some DCs or under-utilization and energy wastage at others. Selecting the best load balancing criterion among cloud DCs is of high importance due to the fact that it not only affects DCs' performance, but also affects prescribed users` SLAs. Both of these factors need to be taken into consideration while making load balancing decisions so that none of them is compromised.

1.2 Thesis Contributions

In order to enhance the performance of cloud data centers for achieving energy efficient operation and taking optimized load balancing decisions, data centers' behavior has to be studied first. This thesis introduces an approach for modeling a data center as a queuing system used to understand system behavior and predict its performance under different load conditions. This model represents all states at which a DC can be modeled using Markov Chains where these system states can be solved exactly under Markovian assumptions in order to gain insight into how DCs operate and which parameters impact their performance.

The main contributions of this thesis are two approaches for solving energy efficiency and load balancing predicaments in cloud data centers. The first contribution for achieving energy efficiency introduces a model for data centers that allows for a load dependent operation of servers so that the number of active servers in the DC at any given time is only sufficient to serve its current load, where un-utilized servers can be switched-off or put into sleep mode to save the energy consumed while being switched-on and idle. The proposed algorithm uses the hysteresis behavior to reduce the frequency of servers' activations/deactivations by limiting activations of servers only in case of significant load surges and deactivations only when servers becomes idle to guarantee minimal delays for users and thus optimal SLAs. It also takes into consideration activation times of servers as well as reduced service rates for implementing sleep states of servers adopted from Dynamic Voltage and Frequency Scaling 'DVFS' approach. The algorithm is modeled using a two-dimensional Markov Chain and solved exactly by a novel iterative recursive algorithm under Markovian assumptions. Proposed algorithm is tested also using OMNeT++ simulations and implemented on a VMware operated test bed for verification and to test the algorithm under general assumptions other than Markovian. Through results the algorithm shows its efficiency at maintaining a load-adaptive data center with minimal effects on user's delay times without affecting their SLAs while achieving energy saving up to 50% of total DC consumed power depending on hystereses parameters and how much servers' frequencies are scaled during sleep states.

The second main contribution in this thesis is presented through two dynamic load balancing approaches between cloud DCs. These two novel approaches avoid all drawbacks of load balancing algorithms in literature by being decentralized with no single points of failure, performing proactive decisions for enhanced automatic resource provisioning, adapting to rapid load variations by operating in a dynamic manner, and maintaining optimized SLAs for cloud customers. The first algorithm namely Local Server System First 'LSSF' maintains un-balanced load situations between servers as long as a request can have its SLAs satisfied at its local DC, otherwise it is migrated to a foreign data center as long as it's migration will not affect performance of the foreign DC or its ability to maintain SLAs of its requests. The second algorithm Shortest Response Time First 'SRTF' is an upgrade for the known Join Shortest Queue 'JSQ' load balancing algorithm [76]. Instead of routing a request to the DC which has the shortest queue size, SRTF accounts for more realistic cases of heterogeneous servers with various number of servers and diverse service rates by calculating the expected mean waiting time of the request at its own DC and all foreign DC then routes it to the one offering the least time. Both algorithms are modeled using a basic example of two homogenous DCs to construct a two-dimensional Markov Chain representing all combined states for the two DCs. Models are solved exactly under Markovian assumptions upon which mathematical analyses are based to provide performance metrics essential for evaluating and optimizing system's performance. Analytical results are verified with OMNeT++ simulations and test bed experimentations which are used for testing more cases of the algorithm that cannot be handled with more than two DCs under non-Markovian assumptions. The two algorithms show their efficiency at balancing the load among several heterogeneous cloud DC while maintaining an efficient operation and most importantly users' SLAs.

1.3 Thesis Outline

This thesis provides an overview on cloud computing technology with a state-of-the art literature study on its architecture and challenges. Essential cloud computing technologies such as virtualization and migration are explained followed by a highlight on the major challenges addressed in this work: Energy efficiency and Load balancing. The work is organized as follows: Chapter 2 gives an overview on cloud computing technology, its various types and models and why it evolved to be one of the most widely spread technologies nowadays. The structure of cloud data centers are explained with attention to most important challenges faced by cloud providers for managing their data centers efficiently. Separate sections are dedicated for both server virtualization as well as virtual machine migration as they are two of the most important technologies upon which cloud computing is deployed. The concept of server virtualization which enables servers to host and serve more than different customers with different service needs at the same time is explained followed by examples for several hypervisors used currently for performing this task. As fulfilling users' requests typically requires creation of virtual machines of different types and service models, these virtual machines often need to be migrated between servers for maintenance, energy efficiency or load balancing purposes. Different techniques for VM migration are also addressed in this chapters with different use cases for each technique. Finally the chapter concludes with an overview on various factors that contribute to the high cost of running a cloud data centers that are crucial to consider for maintaining the desired cost-efficiency of cloud computing technology.

Chapter 3 in this thesis focuses on the two main cloud data centers' challenges addressed in this work. Section 1 of this chapters addresses the energy efficiency of cloud data centers and explains how its current operation leads to an enormous consumption of energy leading to an increase in its energy bill as well as performance deficiencies. Several approaches used for solving this problem and adopted in the algorithms proposed in this thesis are addressed and explained with detailed reviews on existing implementations for these technologies in literature. The first of these approaches is server consolidation, where the load of a data center is consolidated on a smaller number of servers in order to reduce the number of active servers in the data center. For in-active servers, sleep modes are one of the explained approaches that suggests putting servers into sleep modes while being un-utilized so that they can be brought into service when load surges with lower activation time than if they were switched-off, thus saving energy consumed by idle servers with compromise to performance. Another approach for an energy efficient operation is to reduce the frequency at which servers operate through low load durations, thus increasing the service duration of requests by reducing servers' service rates instead of complete shut-down or sleep. This Dynamic Voltage and Frequency Scaling approach 'DVFS' has been studied extensively in literature as being one of the most flexible solutions for dynamic provisioning of data centers' energy consumption in accordance to their current loads. Section 2 of this chapter addresses the load balancing dilemma across several data centers, with the main challenges facing load balancing algorithms currently. Several examples of load balancing algorithms in literature are introduced and compared, each with its main advantages and drawbacks that motivated for the two algorithms presented in this thesis; where most of the existing algorithms have problems of being centralized, re-active to over-load conditions instead of reactively predicting and solving them, being highly complex thus causing relatively long delays for taking load balancing decisions or being static algorithms that do not adapt dynamically to the current data centers' loads. All these aspects have been avoided in the algorithm introduced later in Chapter 6.

In Chapter 4 the methodology adopted for carrying out the required analysis, simulations and implementation of the proposed algorithms is explained. In this thesis data centers are modeled and analyzed as queuing systems, where the basic queuing models and their analysis methods are explained the first section of Chapter 4. Queuing systems under Markovian assumption are also explained as these assumptions are the key for exactly solving the proposed models. Section 2 follows with an introduction to OMNeT++ which is one of the simulation software mostly used for network simulations for having built-in modules for all network types and for its powerful simulation analysis. Simulations are of high importance for verification of analytical solutions as well as for testing the algorithms under various conditions that the analytical solution cannot accommodate. Another important verification of the proposed algorithms is by deploying them on existing data centers, thus a test-bed for a small data center is explained in Section 3 upon which all algorithms have been tested for gaining insights on how they perform.

The main contributions of this thesis are introduced within Chapters 5 and 6. Chapter 5 starts with explaining the Multiple Hystereses Model with activation overheads that performs dynamic server consolidation within the data center for adapting the number of active servers within the data center to its current load. After the model is explained with all its notations and assumptions in Section 1, Section 2 explains different analysis methods for studying how a DC performs under this algorithm. The first method is using detailed mathematical analysis for solving steady-state probabilities of all system states, followed by the simulation model as well as the test-bed configuration and an outline for used scripts. The model has two add-ons for more realistic modeling which include accounting for activation overhead delays spent by switched-off or sleeping servers to start servicing a request as well as allowing for reduced service rates by servers that is the main idea behind DVFS approach. Finally, results for the algorithm's performance using analysis are introduced in Section 3 and verified with results from simulation and test-bed. The same analysis and verification approaches are adopted in two sections within Chapter 6 for the two proposed load balancing algorithms LSSF and SRTF. In each section an algorithm is explained, modeled and analyzed mathematically followed by a comparison of results from analytical solution, simulation and experimentation. Algorithms are tested under several load situations and using various sets of parameters such as different values of migration overhead and different migration conditions. In the third section of Chapter 6 the two algorithms are compared against each other for better understanding of which situations are best for deploying each of them. Finally the work concludes in Chapter 7 with an outlook for future research.

Chapter 2 Cloud Data Centers

As a huge part of the IT infrastructure is starting to rely more on cloud computing, cloud data centers are becoming a major focus in research for being the source of cloud services. Data centers continue to grow in size, complexity and importance; which triggers the need to study and understand operational aspects of cloud data centers in order to be able to enhance their performance. In this chapter the fundamentals behind cloud data centers are addressed, starting with a definition of cloud computing, its offered services and how its data centers are structured. Then a brief discussion on virtualization and virtual machine migration strategies is introduced as they are the main technologies behind the cloud. Finally, the chapter concludes with a report on the costs encountered by operating a cloud data center.

2.1 Cloud Services

Cloud Computing has been forecasted by Leonard Kleinrock, one of the founders and chief scientists of ARPANET (Advanced Research Projects Agency Network) in 1969, which later was developed into the internet as we know it nowadays. In Kleinrock's words: "*As of now, computer networks are still in their infancy, but as they grow up and become sophisticated, I will probably see the spread of "computer utilities" which, like present electric and telephone utilities, will service individual homes and offices across the country*" [105]. Years later after the huge development in cloud computing, NIST issued a formal definition of cloud computing: "*Cloud Computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g.: networks, servers, storage, applications and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction*" [136].

Cloud computing has emerged as a new attractive solution for offering large scale distributed computing for all types of services. Based upon several existing technologies such as virtualization and utility computing, cloud computing is not a new approach; but rather a unique operations model making use of existing technologies to run business in a different way [144]. It aims at providing end-users with a service that is reliable, customizable and guaranteed in terms of QoS [109]. It has moved computing from local DCs and limited customer equipment to large and heavily equipped DCs hosted by cloud providers that can offer numerous services for their users by making use of already existing networks as the Internet. Services offered by cloud computing can be described as an off-site access to a pool of shared resources in an on-demand fashion, and are characterized by the following key factors [136]:

i. On-demand self-service
Automatic provisioning offered by cloud computing for its users allows them to add/remove required resources as needed without service providers being involved. Cloud users can increase required storage/processing/memory/bandwidth resources or release them only by filing a request which is automatically fulfilled by the provider. This eliminates the need for planning for provisioning far ahead to accommodate any load surges [111].

ii. Broad network access
Users can easily connect to service providers making use of all their offered services as long as they have access to the network through commonly used standard mechanisms.

iii. Resource pooling
Cloud providers' resources are pooled, i.e. shared by all cloud users to provide multiple on-demand access, which allows for a multi-tenant cloud model. Being a distributed system; the cloud provides location transparency in terms of hiding the location details of resources while being accessed, so that a user making use of a service mostly cannot tell the physical location of the resource hosting this service.

iv. Rapid elasticity
Ease of automatic add/release of resources according to the amount of user workload is an essential characteristic of the cloud. Elasticity in provisioning of resources allows for an increase of service efficiency, and leaves users with the impression that resources are unlimited and can be easily adapted as needed.

v. Measured service
Cloud providers use a pay-per-use model for commissioning their users according to their actual usage of resources. Resource usage is easily monitored by both users and providers, which provides mutual transparency and eases metering of resource usage.

Cloud computing is based on two basic concepts: Abstraction and Virtualization [63]. Abstraction refers to hiding all system details from customers and only providing them with the required services. Details of the physical systems hosting applications are unknown to the user, as well as locations where data is stored. Requests arrive at the cloud provider and get processed according to its SLA terms without any information about protocols used, computation or storage complexity. These details are taken care of by the network and back-end servers which are responsible for providing the best performance required by requests' SLAs [150]. Virtualization refers to the idea of creating multiple virtual servers hosted on a single physical server, where each user is only aware of the virtual server it is currently being served at. Virtualization is the basis concept behind cloud computing, as it provides simultaneous execution of various tasks belonging to different users on the same physical platform. A more detailed discussion on virtualization will follow in Section 2.3.

Cloud providers nowadays provide numerous solutions for individuals and business platforms; these services can be characterized into three service models, briefly explained and compared in the following Table 2-1 [33]:

Table 2-1: Types of Cloud Services

	Software as a Service (SaaS)	Platform as a Service (PaaS)	Infrastructure as a Service (IaaS)
Type of offered service	Provides access to applications hosted at the cloud provider's infrastructure, where applications can be accessed via thin or thick clients. Users can only make use of the application without any knowledge or control over the infrastructure, network or operating systems underlying it.	Provides access to computing platform hosting numerous applications, with full access to all platform capabilities and denied access to the underlying infrastructure of servers, storage or operating systems.	Provides access to processing, storage, networking and all fundamental infrastructure of a data center. Users acquiring IaaS type of service can install and use any desired operating systems and applications using the existing infrastructure.
Characteristics	- Central management of the software at provider's infrastructure, where providers handle all software upgrades and maintenance. - 'One-to-many' service delivery model.	- Provides a platform for developing and testing applications - Integration with existing databases and web applications	- Multi-tenant model, where multiple users can be hosted on the same physical equipment - Possible dynamic scaling of acquired resources
Appropriate usage scenarios	- Essential business solutions that have fundamental needs, e.g.: E-mail - Software with short-term need or frequent usage spikes.	- Application development with multiple developers, where 'Multi-tenant' architecture allows for multiple access for developing and testing applications	- Businesses with limited capital to invest in establishing own data centers -Growing businesses where adding more equipment is problematic, or when spikes of work load exist.
Examples	Mail servers Database Storage Application-Specific Software "APPs"	Google App Engine [70] Microsoft Azure Services [69]	Outsourced Data Centers

In their definition of Cloud Computing [136], NIST defined 4 models by which cloud computing can be deployed, differences between them depend on whether resources are shared among different users or provisioned exclusively for some of them. The four deployment models are:

i. Private Cloud
Refers to clouds that are provisioned for use by a single organization and users only with the organization where access by the general public is forbidden. It can be hosted by the owning organization or by a third party provider, and the physical infrastructure can also exist inside or outside organization’s premises.

ii. Community Cloud
Providing less privacy than private clouds, community clouds have the same characteristics as private clouds but are provisioned for use by selected community of organizations with common and shared interests.

iii. Public Cloud
Public clouds are available for use by the general public. Hosted at cloud providers; any user or any organization can acquire access to services offered by the public cloud.

iv. Hybrid Cloud
Combining two or more of the above mentioned cloud models, hybrid clouds offer more flexibility to achieve defined goals of data security, resource sharing and service expansion.

The huge success behind Cloud Computing is not a result of the high technology it was built upon, but rather due to economic reasons for clients where it is more convenient for them in terms of cost, feasibility and maintenance to outsource their systems. This computing paradigm has been attractive for many users and companies that start migrating their services to the cloud for the following main reasons:

i. Reduced Costs
Since the cloud is characterized by an on-demand service, users can easily rent, provision and pay-per-use any required resources from the provider without requiring any upfront investments, which saves users the huge cost of establishing own data centers with all expensive hardware and software requirements, maintenance and upgrading costs. From an opposite perspective, cloud computing is also an attractive service to offer by organizations who tend to have over-equipped infrastructure, where equipment is only needed to satisfy short peaks of workload but mostly not fully utilized. By adopting the cloud computing service model, these organizations make better use of their investment by allowing access of their resources to outside users across public platforms as the internet.

ii. Scalability of resources
Cloud Computing provides its users with the illusion of availability of access to infinite resources, where users can easily scale the amount of provisioned resources according to their needs instead of adding more equipment or purchasing software systems. This important property of cloud computing eliminates the risks of over-

provisioning or under-provisioning of resources by users, where users may provision the amount of required resources based on durations of high load or low load, so resources end-up being under-utilized or over-utilized, respectively.

2.2 Data Center Architecture

Cloud data centers are the backbone of all services offered by Cloud Computing, they host all the physical equipment that are virtualized to provide tailored services for cloud clients. Structures of data centers consist of rows of racks where each rack carries the functional servers, switches and storage units that are connected together via extremely fast communication links. Other additional components in the racks include -but are not limited to- rack power distribution, built-in keyboard-video-mouse (KVM) and rack-level air or liquid cooling [95]. Typically, data centers contain thousands of servers connected together and to storage elements via switches, routers or other network fabrics [144]. To connect all these data center components together efficient networking architectures are essential, as they will strongly affect the system throughput and applications' performance. Also as cloud services and data centers continue to grow in size, scalability and extensibility of data centers in terms of physical equipment and their functional protocols need to be carefully considered. Layered network architecture is one of the most deployed in today's data centers, an example is the Three-Tier architecture (3TA) where the data center is divided into three layers as shown in Figure 2-1.

i. Access Layer
   It is the layer at which servers and storage elements organized in racks connect to the network, where each rack connects to an access layer switch via a 1 Gbps link. Switches at the access layer relay traffic from/to servers in racks to/from aggregation layer switches. For redundancy purposes, each access layer switch must connect to more than one switch at the aggregation layer, typically using 10 Gbps links.

ii. Aggregation Layer
   Contains all aggregation layer switches which connect to access and core layer switches through redundant links. It hosts important functionalities such as location and domain services as well as firewalls and content switching [91].

iii. Core Layer
   Acting as the gateway of data center to the outside network, core layer routers carry traffic in/out of the data center. It hosts more than one router with redundant links to avoid traffic bottlenecks and single points of failure or for the purpose of load balancing.

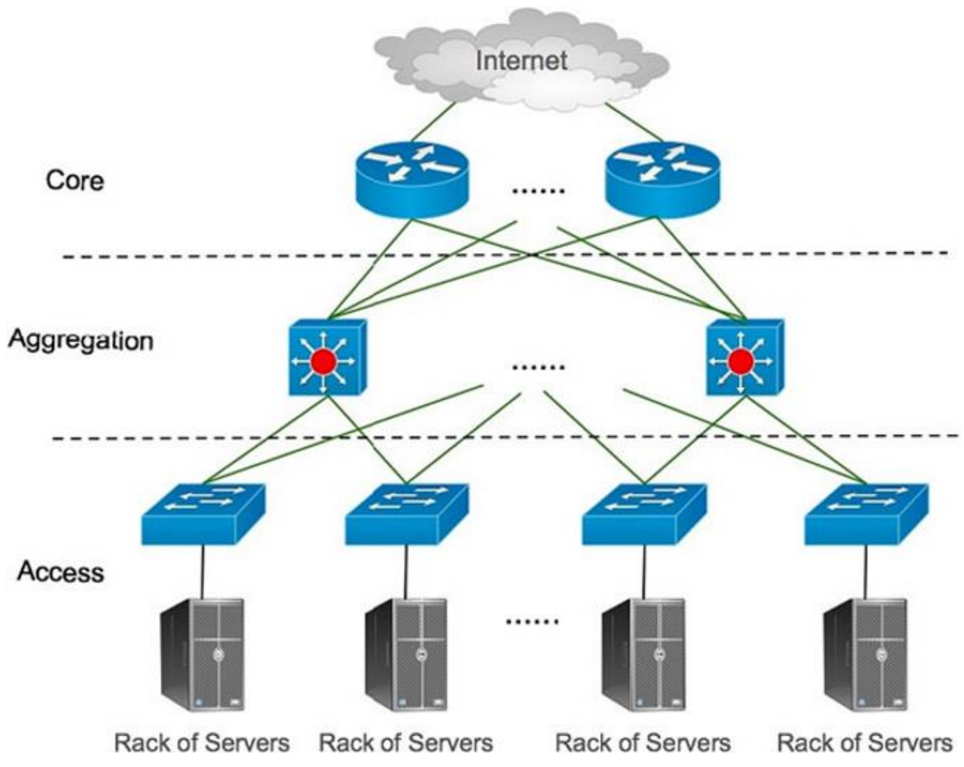


Figure 2-1: Layered Architecture of a Cloud Data Center [144]

Cloud data centers require different types of physical networks to accommodate different types of access to/within elements of the data center [95], architectures of these networks can follow one of the following types:

i. Client-Server Network: connects clients of the service provider to their data hosted on the servers. This network must use a technology widely used by end-users, such as Ethernet or wireless LAN.
ii. Server-to-Server Network: provides the required high-speed connections between servers of the same data center. Network type could be Ethernet, Infiniband, or any suitable high-speed technology.
iii. Storage Access: to provide access for stored data, Fiber Channel could be used for this type of network to provide fast and efficient data delivery.
iv. Management Network: for managing all data centers' devices, the management network may use Ethernet, separate cabling, or exist as a sideband on the mainstream network.

Many methods are available to provide storage solutions for data centers. If storage components are not directly integrated with the servers in racks, separate storage bricks could be added. These storage bricks could be simple ones mounted on the rack slots or high performance storage units hosted separately in special storage towers and connected to racks via network links. For providing efficient access to any of these storage types it is important that the data center implements high performance file systems. In general; storage within data centers follows one or more of the following implementations:

i. Direct Attached Storage (DAS)
Where servers have their own built-in storage, for example in the form of hard disks attached through Small Computer System Interface (SCSI). DAS has limitations in terms of size and sharing flexibility, but also has its benefits of low delay and high performance due to being locally connected to the server.

ii. Network Attached Storage (NAS)
Storage is provided via storage devices connected to the Local Area Network of servers via Ethernet. As data storage in NAS is file-based, it is easy to deploy with different types of file-sharing protocols. It provides higher storage capacity than DAS, as well as simple configuration and administration.

iii. Storage Area Network (SAN)
All storage devices are grouped together in a high-speed network that connects to servers in the DC providing block based storage. Communication between servers within the SAN is usually carried over Fiber Channel (FC) or Internet Small Computer System Interface (iSCSI) to provide data communication at high speeds. SAN Technology improve storage utilization by consolidating all storage in a network that can be accessed by any device in the DC. It also allows for cost reduction by eliminating any extra costs of unnecessary storage.

Finally, for management purposes of data center equipment, a management infrastructure is implemented by connecting all Baseband Management Controllers (BMC) found at each server in the data center. This infrastructure is responsible for controlling switching-on/off of servers, managing hardware and software alerts, maintaining configuration data of devices in case of breakdown and providing remote management capabilities [95].

2.3 Server Virtualization

Virtualization of data centers can be envisioned as an evolution of dedicated physical owned resources to outsourced, shared and geographically distributed infrastructures that are able to provide the same level of control and isolation as owned physical resources [95]. Despite its numerous challenges, virtualization is considered the core idea behind cloud data centers for achieving optimized flexible resource utilization by execution of various tasks simultaneously over a shared hardware platform [103]. It provides a proficient approach for managing cloud data centers to achieve efficient resource utilization through dynamic provisioning and monitoring of resources [88]. Virtualization as a concept has been around for long, Virtual Memory in multi-programmed paged computer systems with dynamic address translation at execution time, Virtual Local Area Networks (VLANs), Virtual Routing and Forwarding (VRF) and Logical Unit Numbers (LUNs) are examples of long-existing virtualization technologies. However, server virtualization began to be widely adopted among commodity servers in 1998 when VMware [190] implemented the Virtual

Machine (VM) concept on x86 hardware servers by effectively partitioning clock cycles among different processor tasks [56]. As hardware technologies continue to develop more powerful servers with increased processing power and storage capacities, VM consolidation ratios tend to increase and thus arising the need for sever virtualization.

When properly implemented, virtualization brings numerous advantages to cloud data centers, including and not limited to:

i. Reduced Costs
The fact that multiple virtual servers can be hosted on single physical server results in requiring less physical servers at the data center, thus reducing capital expenditure (Capex) by reducing the cost of infrastructure, cooling, and reducing the size of data centers.
ii. Higher Utilization
As servers in data centers typically experience very low utilization levels (as low as 10% [89]), virtualization allows servers to host more virtual machines thus increasing their work load and utilization, making more use of existing equipment that is already running and having their temperature controlled.
iii. Availability
Unlike physical servers' failures which are harder to resolve, virtual servers are much easier to handle at failure situations by migrating them from failed servers to another running server. As will be explained in the next section, VM migration can be done in a fast way without any interruption to the work flow on the migrated server, thus providing fast and reliable failover scenarios that can be resolved automatically by the hypervisor.

In addition to its advantages, virtualization subjects cloud data center to a major vulnerability in terms of security. As data centers host more virtual machines and thus grow in size and importance, they become more vulnerable to attacks. Strict security measures must be taken in order to prevent an attack targeting one of the hosted VMs at a server from affecting other co-hosted VMs.

The concept of virtualization is to add a software layer between operating systems and hardware -instead of direct installation- to be responsible for installing several VMs with various operating systems on a single physical server, where this layer is called the Hypervisor layer or Virtual Machine Monitor (VMM). Virtualization is simply implemented at cloud data centers by installing a Hypervisor on "bare metal", i.e. on any server without requiring a supporting operating system to be installed. A hypervisor can easily create, provision, manage and shutdown virtual machines on the physical server as per user's requests, where each VM is assigned a set of virtual hardware resources upon which it can install its operating system and host its services. Although physical resources are shared among all users, this information is not visible to any of them. Figure 2-2 illustrates the process of server virtualization, where each VM is assigned portion of the virtualized servers' CPU, I/O, RAM and Disk capacity resources. Usually, service providers tend to perform overprovisioning of resources assigned to each VM hosted at virtualized servers, where the total resources assigned are greater than existing ones [10]. This approach is efficient because rarely all VMs will have peak loads at the same time, so different peak loads at different times will eventually lead to steady utilization at virtualized servers.

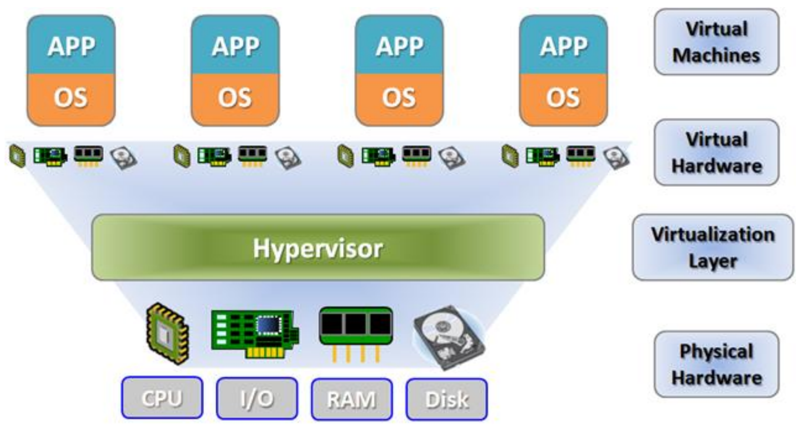


Figure 2-2: Layers and Components of a Virtualized Server [89]

Being responsible for all virtualization tasks in data centers, hypervisor implementations face several challenges for achieving concise server virtualization, such as [132]:

i. VM Isolation
Virtual machines hosted on the same physical server must be isolated in terms of execution, so that execution of one VM cannot affect execution of others. Co-located virtual machines can only communicate together through the hypervisor, and each has access only to its data and allocated resources. This is most important for preventing denial of service (DoS) attacks, where one VM's improper execution can cause failure to the physical server and thus denial of service at all other concurrently hosted VMs. This multi-tenancy problem imposes a huge overhead especially at public clouds, where authors of [20] have reported experiencing unpredictable and unstable performance while testing computationally intensive tasks hosted at commercial cloud facilities.

ii. OS Support
Hypervisors need to support multiple types of operating systems to be installed by VMs such as Windows, Linux, etc… in order to accommodate different technology requirements.

iii. Minimal Performance Overhead
Overhead resulting from creating and managing of multiple VMs on the same physical server is required to be minimal for sparing physical resources to VMs. Also operational expenditure (Opex) needs to be minimized for virtualization to have an edge over non-virtualized environments where each server is individually managed.

Current implementations of hypervisors provide effective server virtualization as well as numerous advantages to cloud computing industry. For example, Xen hypervisor [132] which is implemented by multiplexing of physical resources at a granularity level of an entire operating system aims at simultaneous hosting of up to 100 VM on a single modern server while sustaining performance isolation among them. It enables attractive applications such as dynamic instantiation of VMs and their operating systems, server consolidation and VM mobility. Kernel Virtual Machine (KVM) is another hypervisor solution for Linux environments which maintains standard Linux services such as scheduler and memory management to allow developers to focus on virtualization instead of replacing the core kernel [72]. Another example is VMware ESXi hypervisor [190] which performs VM provisioning with intelligent load balancing for enhanced system performance while maintaining both system-level constraints and service level agreements for users [19]. VMware's memory mechanism used by the ESXi server is shown in [36], namely the ballooning technique, which allows de-allocation of memory pages that was previously assigned to a VM (balloon deflation) to be assigned to another virtual machine (balloon inflation) without shutting down any of them. This is implemented by loading a balloon module onto the guest operating system which communicates with the server to perform balloon inflation when memory needs to be reclaimed and deflation when memory is to be released [182]. VMware implements other efficient mechanisms such as VMware's Dynamic Resource Scheduler (DRS) which acts as the hypervisor and is responsible for the allocation of virtual machines on physical servers. Its operation is enhanced by Dynamic Power management (DPM), which performs server consolidation to group virtual machines on less number of hosts so that lightly utilized servers can be evacuated and either powered-off or put to a low-power mode. DPM's approach of operating servers is partly adopted and modeled among energy efficiency models in this thesis, as will be shown in Chapter 5.

2.4 VM Migration

Migration of virtual machines between different servers at the same data center or across data centers is one of the most significant advantages provided by virtualization. Being considered an evolution of process migration techniques [161], migration of virtual machines provides an efficient solution as elaborated by Clark et al in [39], where they clarified that migrating a virtual machine including its operating system and applications is more efficient and avoids many of the difficulties faced by process migration techniques. Basically, a virtual machine can be migrated to another host within the same Local Area Network (LAN) or to another LAN across Wide Area Network (WAN) links [9]. Migration within the same LAN [177] is easier as only the VM is migrated with no need to migrate its storage as well since NAS storage solutions can still be accessed among the same LAN. While for inter-LAN migration across WAN links a VM need to be migrated along with its storage, where the size of CPU and devices' states are in order of several KBs which is relatively small compared to the size of memory pages which typically ranges in GBs [24]. This leads to longer migration times for migrating both entities added to it the overhead for IP addressing and possible delays to network congestion or limited available bandwidth [137].

VM Migration brings several benefits to cloud data centers by achieving efficient resource management goals without any effect to hosted VMs. The capability of redistributing the load across servers via VM migrations enables highly responsive resource provisioning to compensate for any sudden peak loads resulting in hotspots [144]. Among goals achieved by VM migrations are [145]:

i. Power Management
   Performed by server consolidation, where virtual machines from under-utilized servers can be migrated to other functional servers that are not overloaded in order to either switch-off under-utilized servers or put them in a low power mode by reducing their CPU clock rates using Dynamic Voltage and Frequency Scaling (DVFS) method. Although power efficiency at the data center is achieved at a cost of performance degradation, careful consideration by power management algorithms is important to guarantee that customers' SLAs are not violated.
ii. Load Balancing
   VM migration allows for eliminating hotspots in the data center by migrating workload from overloaded servers to under loaded ones to prevent performance degradation [155], as well as achieving any defined load balancing strategy defined by DC administrator among DC resources.
iii. Resource Sharing
   VM migration allows for resource-hungry applications to be migrated from an overloaded server and assigned to another resource-rich server, thus providing better performance for applications and servers [183].
iv. System Maintenance
   As dynamic and periodic system maintenance is of high importance to the system's performance and extends its life span [152], VM migration allows for easier maintenance by shifting VMs from a server during its maintenance duration without affecting VM's workload, and shifting them back after maintenance is complete.
v. Fault Tolerance
   A data center that is fault tolerant detects faults and migrates VMs from faulty servers until they are available again [6], or preferably migration occurs even before a fault occurs, thus improving system's reliability.
vi. Mobile Computing
   As cloud users are becoming more mobile and increasingly on the move, a user can require his running application on a desktop to be available on a mobile device or vice versa [122]. VM migration enhances users' mobility by migrating their currently running applications across any desired platform. Another application is Migrating users among Baseband units 'BBUs' within the uprising technology of Cloud Radio Access Networks CRAN [12].

There exist two types of VM migration: live and non-live. Live migration migrates the VM without any interruption to the workload, thus providing seamless connectivity and maintaining service level agreements of users. Non-live migration stops the workload, migrates the VM and continues service again at the destination server. Downtime during non-live migration time is usually predictable so that the migration decision can be taken based on whether downtime will comply with or exceed defined SLAs. Each of the before mentioned types can be classified into different approaches such as pre-copy -which is the default live migration method for many hypervisors as Xen, KVM and VMware [110]-, post-copy and hybrid methods for live migration and internet suspend resume and process domain for non-live migration. All different live and non-live approaches are explained and compared in Table 2-2.

Table 2-2: VM Migration Approaches

Approach	Live VM Migration			Non-live VM Migration	
Definition	Virtual machine and its hosted applications are kept running during migration process, where iterative copies of the VM state are continuously being copied from source to destination server finalized by a stop-and-copy approach [98]. The number of iterations determine the live migration time.			Virtual machine along with all its running application are suspended and completely transferred to destination server where it resumes its workload.	
Advantages	- Provides applications with uninterrupted workflow during migration, thus enhancing their performance - Minimal downtime during migration			- Downtime is predictable - Single transmission of memory pages as they don't encounter modifications during migration process, unlike live migration.	
Disadvantages	Extensive consumption of network and DC resources during copy iterations especially for memory-intensive applications, thus degrading DC's performance if resource consumption is not carefully optimized [145][110].			Applications' QoS suffers degradation due to downtime during migration process.	
Schemes	Pre-Copy	Post-Copy	Hybrid	Internet Suspend Resume [117]	Process Domain [162]

Definition	- VM's memory is transferred first to the destination server in rounds, then VM is resumed at destination. - Each round transfers dirty memory pages which have been modified at source after migration [24].	- Minimum VM state is transferred and resumed at destination server first, followed by memory pages pushed from source server and requested memory pages by the VM [121].	- Combination of pre and post-copy for enhanced performance. - Starts with bounded pre-copy phase, followed by post-copy migration of minimal VM state to the destination server.	After the VM is suspended, it is transported to the destination server via a distributed file system.	Zap system is introduced for transparent process migration of unmodified applications by transferring a process domain to destination server
Advantages	Uninterrupted workload for VMs and applications	- Reduced VM migration time and downtime - Enhanced performance for write-intensive applications [176]	Additional pre-copy phase reduced page faults as a large percentage of memory pages are already transferred during pre-copy phase	Trivial process as migration becomes a transparent process [145].	Low overhead general process migration method
Disadvantages	-Longer total migration time - High occupation of system resources by migration daemon and hosted applications, which increases SLA violation probability of applications [145]. - In-efficient for write-intensive	Possible service degradation due to page faults as a virtual machine might request memory pages that are not yet transferred to the destination server	Although hybrid method perform much better than post-copy in terms of page faults, applications' performance is still affected if memory dirtying rate is higher than memory transfer rate.	Lack of locality heuristics and secure channels for efficient data transfer [145].	- Prone to security threats - Lack of definite decisions on which process domains are to be migrated and when.

	applications as number of rounds for dirty data transfer increase rapidly [176]				

Throughout load balancing algorithms proposed in this thesis at later chapters, the type of VM migrations across servers or data centers is not strictly specified. In our work virtual machines are migrated before starting their service at the source server, so the problem is more of a scheduling problem for virtual machine requests where challenges of life migration are not applicable and accordingly not considered. The only consideration in the proposed algorithms is the migration time for a virtual machine request, which is the main element to consider while making migration decisions.

2.5 Costs of operating Cloud Data Centers

The cost of operating a system 24/7 with acceptable latency is crucial, and being aware of these costs is the main step towards reducing them. According to [84], 53% of the total expenditure of operating a data center is consumed by powering and cooling of data center equipment. Virtualization has been of much benefit to running system costs, as using single physical server for running multiple systems deployed as VMs have reduced the cost of equipment, cooling and data center space, thus reducing overall Capex. Opex also contribute to the overall running system costs, where compared to non-virtualized data centers, managing VMs deployed at a physical server by a hypervisor requires more attention due to sharing and contention, unlike individually deployed systems. According to authors in [19] a considerably large amount of Opex in a virtualized data center is consumed for determining optimum VM-to-host- mappings, and updating these mappings for consolidation or load balancing purposes. As for the cooling costs, cooling infrastructure of a Data Center could be quite expensive due to its numerous components which include large chiller plants, fans and air-circulation systems. Despite the heavy equipment, the cooling infrastructure might not always be able to handle the offered thermal load, which leads to difficulty at loading the racks to their capacity [95].

Another contributor to the cost of operating a data center is its architecture. Nowadays, the most dominant architecture for cloud data centers is in the form of large data centers that are managed in a centralized fashion. Despite its efficiency due to the effect of economy of scale [144], this architecture has high expenses for equipping data centers and cooling infrastructure. A counter initiative has been advocated by authors of [93] and [184], where they suggested switching to a distributed architecture by having smaller sized data centers. This approach replaces large data centers hosting in order of tens of thousands of servers consuming tens of Mega Watt power by smaller distributed data centers hosting only thousands of servers that draw an amount of power in order of hundreds of Kilo Watts [18]. Arguments supporting this approach are that smaller data centers are cheaper to build, require less cooling infrastructure, and they can be geographically distributed. These arguments result in relatively lower energy consumption than centralized approach, as well as lower response times for users especially for time-sensitive applications, since distributed data centers are physically closer to users than a central entity and are appropriate for ‘embarrassingly distributed applications’ as e-mail by acting as nodes for content distribution networks [93]. As reported in [149], reducing the latency of users’ requests increases performance and directly increases business revenue. Google and Amazon had reported a drop by 20% and 1 % in their revenues and sales when their requests' latency increased by as low as 500 and 100 milliseconds, respectively [1]. This clarifies the need for careful consideration of a data center’s performance in order to increase its revenue and maintain users’ service level agreements. Important considerations for selecting data center locations are places that are cheap to rent, provided by cheap electricity resources as well as water supplies for cooling, and cheap manpower wages.

Authors of [93] reported that data centers are usually managed in-efficiently with a minor utilization rate as low as 10%, thus resulting in wasted operating costs. Reasons behind this include:

i. Equipment provisioning for long durations
As equipment purchases is not a task done within short durations, data center administrators usually purchase components in bulk in order to last longer times and increase time between purchases. This usually leaves a remarkable amount of the equipment unutilized.

ii. Inexact fit of applications to servers
As applications make use of servers’ memory, CPU, network and storage elements, it is not necessarily the case that hosted applications fully utilize all these components, thus leaving behind unutilized resources.

iii. Resource Overprovisioning
Administrators usually tend to perform overprovisioning by assigning more resources to an application than it regularly uses by planning for peak loads, which results in unutilized resources during normal load operation times.

iv. System level resilience
Administrators usually design data centers while taking the important factor of redundancy into consideration by planning for redundant equipment so that a data center never fails. As redundancy level increases infrastructure cost also rises along with the administrative effort to manage all redundant infrastructure to handle all failure scenarios. Authors of [18] proposed an approach to reduce such costs by allowing data centers to fail while arranging for other data centers to handle their workloads, thus removing several redundancy layers from the data center.

All these factors leading to in-efficient use of data center resources result in increased running cost, and can be resolved by dynamic assignment of resources to applications on-demand according to current system load, as will be introduced by proposed energy efficiency model in Chapter 5.

Chapter 3 Challenges of Virtualized Cloud Data Centers

This chapter introduces two of the main challenges that face cloud data centers' implementation nowadays. First the problem of energy efficiency is considered as the huge amount of energy consumption is one of the most critical downsides of cloud computing data centers. Numerous solutions have been proposed for reducing the consumed energy and achieving energy efficiency, such as server consolidation, sleep modes and Dynamic Voltage and Frequency Scaling. Within the following subsections each approach will be explained and discussed briefly, along with implementation examples from literature. Second is the load balancing paradigm; although it has always existed in different types of networking systems, special considerations must be taken while introducing existing algorithms to the cloud due to its unique operational model. In the second subsection a review on most of the existing load balancing approaches is introduced where different approaches are explained and compared to give a complete view on the state-of-the-art in this area.

3.1 Energy Efficiency of Cloud DCs

As cloud computing becomes more popular and more companies start migrating their business into the cloud, cloud providers increase the sizes of their data centers to accommodate the increasing demand. This growth in data center components and capabilities is accompanied by a corresponding increase at equipment size as well as power consumption. Since 2008 the ICT industry has been considered among the top energy consumers by the EU advisory group [76] for manufacturing equipment, using and disposing them. As reported in [34] and [81], between 1.1% - 1.5% of the total world-wide generated energy was consumed by operating and cooling of data centers, and this percentage was foreseen to grow annually by 18%. In order to evaluate data centers' energy efficiency authors of [178] defined the metric Power Usage Efficiency (PUE), which is a ratio between the total amount of energy consumed by the data center and the amount of energy consumed by servers. Typically, an inefficiently managed data center will have a PUE value in the range of 2.0 - 3.0, a top tier data center will have a PUE of 1.2, and a well-managed facility could have a PUE of 1.7, but this number lies much below the average for the data centers among the world [18][124]. As explained in other words by authors of [18], a PUE of 1.7 reflects that for every energy Watt delivered to the data center only 59% of it is used by processing server, while 33% is used by cooling infrastructure and 8% is estimated to be lost due to power distribution.

For example, a typical data center of size 500 m^2 consumes daily an estimate of 27,048 kWh [4], which is the amount of power required for supplying around 2500 houses in the European Union [31]. Being one of the market leaders in the field of cloud computing, Google estimated the amount of energy consumed by its data centers in 2013 [87] to be 260 million energy Watts, equivalent to 0.01% of the total energy of the globe that is sufficient to power almost 200,000 homes. Huge as this amount of energy consumption is, it is foreseen to increase. In a survey by [3] among several data center managers, 38 % of the sample had solid plans for building new data centers and 47% planned for expansions or renovations for their existing ones, which raises concerns towards the expected rapid growth in data centers' sizes and energy consumption rates. Recently, huge DCs have been installed in Polar Regions where the gain by energy reduction over-compensates additional transmission costs.

Accordingly, approaches to reduce the amount of power consumed by data centers are topics that have been and still are of huge interest in current research and literature. 'Green Computing' is the name given to approaches for reducing energy consumption in computing centers. Green computing defines a set of procedures according to which computing is done efficiently, while guaranteeing minimal energy consumption by resources. Before briefly explaining three among the most common procedures in the following sub-sections, reasons for in-efficient consumption of energy are to be considered first. Authors in [172] identified two reasons of energy in-efficiency where consumed energy is either lost or wasted. Lost energy refers to energy that was not consumed by the data center for its main task, i.e. computing. This includes energy lost during transportation, conversion, or consumed by secondary supporting systems as cooling which consumes around 0.5-1 Watt of energy for each Watt consumed by computing servers [40]. Wasted energy refers to energy that was used by servers of the data centers without producing an output, i.e. energy consumed by idle/sleeping servers, where an idle server can consume more than 50% of the amount it consumes while running at full power [74]. Energy waste is typically encountered by data centers where servers run idle or at a low utilization level between 10-50 % [100]. According to [4] a data center that operates at 20% of its operational capacity approximately consumes 80% of consumed power when it runs at full capacity. IBM reported similar results in its report [77] where it estimated an amount of 85% of data center computing equipment to be idle.

While reducing lost energy is not related to the operation of servers in a DC, it can be done by introducing new technologies for minimizing the energy required for DC supporting systems, and will not focused upon in the context of this thesis. The main focus will be on approaches for reducing the energy wasted by servers running unnecessarily, as there is a huge space for development in this area due to fluctuating load levels in data centers during users' idle times at which saving computing and cooling power could be achieved [51]. This was also reported by Intel [2] where they estimated that the amount of power consumed by a data center can be reduced by as much as 20% without any impact on its performance by reducing wasted energy. Approaches include switching-off or putting un-/under-utilized resources to sleep as implemented in VMware's Dynamic Power Management (DPM) [19] or keeping them switched-on at low voltage and frequency levels using DVFS (Dynamic voltage and Frequency Scaling) [167]. These methods will be implemented, modeled and discussed thoroughly within Chapter 5 with the aid of Queuing theory and simulation tool.

However few research work have considered methods for energy reduction that are aware of the system performance and maintain service level agreement of users, which is the main consideration for this work. Reducing energy consumption usually means reducing system utilization and thus affecting system's performance in terms of quality of service and experience for users. Designing algorithms that takes both requirements into consideration has only been addressed few times in literature. Authors of [154] illustrated this conflict by an experiment using a 13-node test bed where a power manager and a virtualization manager were deployed separately to monitor the unique effect of each of them. The experiment concluded that mostly violations in service level agreements and power consumption threshold occur at the same time and that they are strongly correlated as they cause one another. In other words, when the level of power consumption increases the power manager reacts by reducing the frequency of CPU cycles to reduce it, which causes violations of SLA. SLA violations on the other hand cause virtualization manager to increase frequency of CPU cycles to maintain SLA levels resulting in power threshold violation, and so forth. Thus, algorithms that consider aspects of power efficiency as well as users' SLAs are of much importance to maintain a balanced and efficient system performance. In the following subsections we discuss different approaches for implementing energy efficiency, along with references to state-of-the-art implementations from literature.

3.1.1 Server Consolidation

Server consolidation refers to the method of consolidating work load of the data center on fewer number of servers so that lightly loaded servers can be shut down or put on low-power mode. It is an effective approach for minimizing the number of active servers and thus the amount of energy consumed by the data center for achieving better utilization of system resources. In cloud data centers virtualization has made it possible via migration of VMs from low-loaded servers to under-utilized servers, allowing the earlier to be shut down and the latter to be efficiently utilized. The process of server consolidation includes several decisions that need to be performed by a consolidation framework, where different framework implementations differ in their handling of these decisions which are explained below as stated in [151]:

i. Resource Assignment Policy

   It describes how system resources are assigned to VMs, where resource mapping could be done in a static or dynamic fashion. Frameworks assigning resources statically grant a VM the maximum amount of resources it could need upon deploying it to a server. On the other hand dynamic assignment allocates resources on-demand based on VM's current work load. Most consolidation frameworks use the more energy efficient dynamic allocations such as in [197] and [52], as algorithms following static allocation criteria e.g.: [169] suffer from inefficient VM placements compared to dynamic assignment since not much VMs can be packed on one server if each VM occupies its maximum capacity [68] .

ii. Architecture

   Represents the architecture of the consolidation framework where it can be centralized or decentralized. Decentralized approaches as implemented in [7] show better efficiency and reliability as they scale well with increasing system size and eliminate single point of failure risk experienced with centralized approaches, such as centralized approach implemented by authors of [171].

iii. Co-location Criteria
It is the criteria upon which decision of which VMs to be placed together is made. Co-hosting particular VMs together can be done for communication purposes where the communication cost needs to be reduced e.g.: [9] [52] , for increasing availability of shared resources e.g.: [123] [174] or energy efficiency purposes as will be addressed shortly in details.

iv. Migration Trigger
Decides when a particular VM is to be migrated. This trigger could be computed using several approaches such as scheduled migration where a schedule is defined so that an evaluation of the system is done to decide whether trigger needs to be activated, e.g. [123]. Triggered migration could also be based on historical data to predict future load behavior, e.g.: [52] [67] or it could be based on heuristic-based trigger as done in [34] [169] [8].

v. Migration Model
Decides how the VM will be migrated depending on the nature of its hosted applications. As previously discussed in Section 2.4, migration techniques include pre-copy, post-copy or hybrid approach.

Another important property of server consolidation mechanisms is presented by authors of [144], where they highlighted the necessity of automated service provisioning in cloud data centers in order to have close-to-instant response to rapid load fluctuations. They have specified steps for reaching automated provisioning through constructing application performance models for predicting demands and adjusting allocated resources and performing consolidation automatically. Although using proactive algorithms for predicting application demands ahead and allocating required resources accordingly seems more favorable than the reactive method, they require high system processing capabilities for constructing traffic models and continuously calculating predicted load and required resources to accommodate this load. An efficient solution is to use reactive algorithms with fast responses to variations in load, such as the method proposed in this thesis where the number of servers required for serving arrivals to a data center is automatically provisioned every time a request arrives to the system, thus providing a fast responsive consolidation framework with minimal overhead.

Server consolidation for energy efficiency has been explored heavily in research where several approaches have been proposed for this problem. In [34] EnaCloud approach is proposed which implements dynamic live placement of virtual machines in a cloud data center upon the minimum number of servers for maintaining energy efficiency using heuristic algorithm. It models the VM placement problem as a bin packing problem, where servers are loaded with VM requests until reaching their maximum capacity while accounting for variable amounts of resource a VM can request. Another approach is adopted by authors of DT-PALB algorithm (Double Threshold Energy Aware Load Balancing) [78] by considering servers' utilization percentages where VMs hosted at a server with utilization level below a threshold level of 25% are migrated to another server to shut off the current one. Higher threshold level is determined at 75% CPU utilization, where a new VM arriving when all servers are at or exceeding this level will be hosted at an idle server brought into operation. A multi objective optimization problem for VM placement is introduced in [193] where authors propose an ant colony multi-objective algorithm for minimal energy consumption with simultaneous efficient usage of resources. Similarly, authors of [47] divide the allocation problem into three single objective problems for optimizing energy consumption under constraints of VM performance and vice versa, followed by a third optimization for a combination of both.

A few significant approaches in literature have considered both energy efficiency of data centers as well as users' SLAs while proposing implementations for server consolidation. As the goal of server consolidation is to co-host several VMs upon shared resources of smaller number of servers, it directly impacts the amount of resources assigned to each VM and thus could affect performance of its hosted applications as well as service level agreements of its users. Although infrastructure as DeSVi introduced in [179] was solely developed for the detection of SLA violation, it is important that consolidation infrastructures take it into consideration in order to prevent violations instead of reacting to them. Authors of [7] propose a method based on determining upper and lower adaptive thresholds for server utilization to perform dynamic consolidation while attaining users' service level agreements, where these thresholds could be adapted automatically depending on system's requirements of increased energy savings or enhanced SLAs. In another paper by the same authors [9] they show simulation results for VM dynamic allocation heuristics based on performance of servers' CPUs. In their approach a VMM keeps track of all nodes and their utilization levels to decide if VMs need to be migrated in cases of overload, where a local manager decides upon which VMs will be migrated and where. Authors show that compared to a data center that is not power aware, data centers implementing energy efficiency heuristics show an 83% reduction of energy consumption and an improvement of 66% when compared to others implementing only DVFS solutions.

In [197] authors propose an algorithm for SLA violation decisions and migrates these VMs to another server where minimum power could be achieved while utilization is maximized. Algorithm runs in two steps where the first step is to detect overloaded servers and VMs exposed to SLA violations, followed by the second step of migrating VMs in a decreasing order of their resource consumption intensity to be migrated to underutilized servers until they are utilized maximally. Another aggressive consolidation technique with consideration to SLAs of users is described in [104] where authors describe current consolidation frameworks to be 'rigid' as they consider requested resources by VMs to be fixed. However in their approach this assumption is replaced by a more flexible one as they suggest adjusting/reducing the amount of assigned resources to the VM as long as its SLA is not affected. This treatment of VMs as 'moldable' allows for reducing assigned resources per VM thus allowing more VMs to be co-hosted by the same server, where the modified resources per VM and VM to server mappings are decided by a Genetic algorithm. In their paper authors also investigate how to minimize the transition time between initial system state and the modified state using Genetic algorithms.

Table 3-1 provides a summarized comparison among all previously discussed server consolidation approaches. As all these algorithms tend to perform consolidation for energy efficiency, a few pitfalls of this process that are only accounted for by few algorithms and must be considered in future research while attempting to reduce consumed energy include [151]:

i. Overlooking Migration overhead

As some algorithm perform migration decisions without consideration to the overhead latency caused by migration that highly affects SLAs of virtual machines.

ii. Aggressive consolidation with excessive VM Migrations

Some consolidation frameworks tend to perform server consolidation aggressively by hosting VMs onto servers till the latter reach their maximum CPU utilization, without paying attention to other shared system resources such as memory, cache and networking elements which could cause system instability [52]. Authors of [97] highlighted this problem when the usage of one resource at a server could be blocked due to insufficiency of another resource, and proposes heuristics for efficient VM allocation and efficient use of resources.

iii. Co-Location criteria

Although considered by some algorithms, many of them overlook relations between VMs that must be considered for optimal placement. For example VMs that communicate regularly should not be placed into two distant servers, but should be hosted at two servers on the same LAN to reduce communication latency.

iv. Utilization Prediction Accuracy

As predicting servers' utilization is the main triggering point for most consolidation algorithms upon which consolidation decisions are made, it must be performed dynamically using efficient heuristics such as those used by [9], [7] and [174]. Another important prediction issue is estimating the amount of resources required by an application from its hosted server, which could be performed application profiling as used by [156].

v. Security

Although energy efficiency is a major issue, security of users' data must not be compromised while consolidating servers' loads. In some scenarios where consolidation might violate security policies as placing data from competing consumers at the same platform, it should not be performed.

vi. Residual Resource Fragmentation

An important aspect referred to by authors of [97] is the waste of residual system resources after fragmentation among VMs. As VMs always experience variable loads, the amount of resources assigned to VMs hosted at a physical server varies according to the load leaving few resources that could not be enough for hosting another VM. But while examining the whole data center, the sum of all residual resources at each server can accommodate one or more VM. Authors propose an algorithm for rearranging VMs intelligently to concentrate residual resources on small number of physical machines in order to be able to host more VMs.

Table 3-1: Comparison between Server Consolidation Frameworks

Algorithm	**Type**	**Description**	**VM Placement Criteria**	**Conside-ration for SLA**
EnaCloud [34]	Dynamic	Proposes a heuristic that is energy aware for solving a bin-packing problem of optimal VM Placement with overprovisioning of resources	Fully load each server with VMs before placing VMs elsewhere	No
DT-PALB [78]	Static	Defines a lower threshold for deactivating underutilized servers and an upper threshold after which new servers are allowed to be activated	VMs are placed at servers one by one until a server reaches a 75% utilization, after which a new server accepts VMs	No
Multi-objective Ant Colony [193]	Static	Describes a multi-objective optimization problem for energy reduction and optimal resource utilization simultaneously	Fully load each server with VMs before placing VMs elsewhere	No
Beloglazov et.al. [7]	Dynamic	Adaptive thresholds for server utilization upon which decisions of which VMs to migrate and onto which servers are made	VMs are placed at servers where they will cause least power consumption	Yes
Beloglazov et.al. [9]	Dynamic	Energy efficiency heuristics for detecting overload situations and migrating VMs from to less loaded servers with attention to their QoS	Allocate VMs to servers where their SLA will not be violated	Yes
He et al. [104]	Dynamic	Adjusts amount of assigned resources per VM in order to increase number of VMs hosted per server as long as VMs' QoS is not affected	Fully load each server with VMs while SLAs are not compromised.	Yes
Cao et.al. [197]	Dynamic	Detects servers which are overloaded and runs an algorithm for detecting which VMs are at risk of SLA violation and are candidates for migration to a less loaded server	VMs are migrated and placed to servers in a descending order of the amount of resources they require.	Yes
Borgetto et al. [47]	Dynamic	Optimizes both system performance and power consumption through three single objective optimization problems.	Efficient loading of servers by fully loading each server before switching-on idle server	Yes

Rao et al. [97]	Dynamic	Reduces residual resource fragmentation by redistributing VMs among servers to concentrate residual resources on less number of servers to make use of them.	VMs are placed to servers such that residual resources are minimized	Yes

3.1.2 Sleep Modes

Switching-off idle servers or putting them to sleep is the basic idea for reducing their energy consumption commonly referred to as Dynamic Power Management (DPM). Energy consumed by servers is defined as the amount of performed work during a time duration, whereas power is the rate at which work is performed. Power consumed by servers can be classified into two major elements, a static and a dynamic one [59]. Static element of power consumption refers to the amount of consumed power to power on the server regardless of its workload, while dynamic element is load dependent and is affected by variations in current load, CPU clock frequency, system current and capacity, etc. . Saving the dynamic portion of energy consumption can be achieved through reducing approaches as Dynamic Voltage and Frequency Scaling DVFS as will be discussed in the following subsection. Whereas in order to save power consumed during idle load periods reducing the static element of power consumption is the only solution, which is targeted by server sleep modes. The amount of power consumed by idle servers is not trivial; it contributes with 66% of the maximum power consumption as reported by authors of [146] who propose a model for prediction of idle power consumption by considering numerous server types characterized by various hardware structure and energy-consumption models by the different hardware components. The solution of reducing idle power consumption can be achieved according to predictions by the operating system on the expected idle interval and expected work load, where servers can enter any state among a group of defined C-states where high C-state number indicates deeper sleep states, lower energy consumption and longer latency for the CPU to become active again. CPU C-states and their related interfaces are defined by the Advanced Configuration and Power Interface (ACPI) for x86 systems [30].

Numerous architectures for implementing sleep modes for energy efficiency in virtualized environments have been proposed in literature, where virtualization imposes an added constraint before switching-off a server or putting it into sleep mode that all its hosted VMs are idle. An approach for energy conservation ‘PowerNap’ is introduced in [53] which reduces energy by putting idle servers into sleep modes with low energy consumption and minimizes transition times in and out of these states when load spikes occur. By analysis of real-life traffic scenarios in data centers authors of this paper demonstrate that data centers experience idle periods almost 60% of the time, with idle intervals averaging around 1second. Powernap provides the ability to transit the system between two states: active state where system runs operates at maximum speed and nap mode with minimal energy draw with bounded transition delay in the range of 1-10ms. It also introduces RAILS ‘Redundant Array for Inexpensive Load Sharing’ which is an algorithm for improving efficiency of power supply through sizing modules providing power to the DC for meeting PowerNap’s demands of power supply. Although PowerNap outperforms DVFS and similar approaches at durations of low utilization [53], it is not the best approach to adopt during high utilization durations when transition delay could affect system's performance. Authors of [23] introduce NapSAC algorithm which predicts the expected workload density using several heuristics in order to find out the number of servers required to be active for serving this predicted load under given SLAs. By predicting incoming load servers can be switched-on/off accordingly before load arrives so that transition times do not affect users' SLAs. A similar approach for right-sizing the number of active servers inside the data center to be load dependent is introduced in [118]. This approach introduces an offline algorithm for calculating the required number of servers through an optimization problem, followed by an online Lazy Capacity Provisioning algorithm that is proved by the authors to be performance- and cost-competitive in comparison to offline algorithm.

Another sleep-advocating algorithm is introduced in [54] namely 'DreamWeaver' which facilitates entering deep sleep states at servers with multiple cores running different requests. DreamWeaver operates in two stages; first stage is done by Weave scheduling which coalesces idle and busy periods among all system's cores to allow all of them to execute requests at their highest efficiency then go into sleeping state at the same time. Second stage is carried out by a Dream processor which monitors incoming workload while cores sleep and determines for how long can incoming requests be stalled to allow for more sleeping time of cores and thus more energy savings without affecting users SLAs. As being workload dependent is a main advantage of DreamWeaver, added delays for stalled requests arriving while system is in sleep state is a main downside. Similar to DreamWeaver the approach of delayed activation has been adopted by many algorithms in literature. An example is proposed by authors of [38] who introduce a vacation scheme concept where traffic is reshaped into bursts so that a server can wake up to serve the burst and return into sleep state again, thus maximizing sleep durations. In [37] authors suggest procrastinating waking up of servers as long as arrivals' tail latency constraints will still be satisfied while considering variability among different arrival requests. Accordingly their decisions on when to wake servers up are dynamic and depend on the type of buffered arrivals. Authors of [115] provide an enhancement to these approaches by delaying both activation and deactivation of servers. Delayed activation allows a server to remain in sleep mode for a random time even after an arrival occurs. It offers extended sleep times and thus lower power consumption with a tradeoff of increased latency for users' requests. On the other hand Delayed deactivation keeps the server running for a random time even if there are no arrivals to be served. It allows instantaneous service for arrivals occurring while the system is idle and running with null reduction in energy consumption. This work provides a Markov Model for studying the system, solves it under stationary conditions to study the effects of delayed activations and deactivations, and come to the interesting conclusion via analysis and simulation that when both delay durations are fine-tuned they can reduce energy consumption as well as users' latency.

Stochastic modeling of data centers as queuing systems for studying the effects of DPM models has been approached by many other authors in literature. In their work introduced in [192] authors also introduce a Markov modeling approach for studying the effects of Greedy sleeping policy, Prediction, and Accumulate and Fire policies on both energy consumption and delay. Another model for servers working under adaptive DPM policies as a Markov-modulated process is introduced in [198] where an offline calculation of optimal DPM policies is performed using Markov decision processes, then chosen policies are switched upon online to optimize performance. In [55] authors propose a different approach for energy reduction via sleep modes by formulating the problem of deciding states of servers either active or sleeping as a constrained Markov decision process which is then solved to find the optimum power management solution in the data center. Following decisions upon server states a task broker takes over by distributing incoming requests among active servers so that their SLAs are maintained. Similar work is done in [180] using continuous time Markov Chains to find optimal on/off policies for single server systems, which is extended to a server of general service distribution in [113] and further more into multi-server systems in [73]. Extensive studies have also been performed on sever farms with set-up costs for energy efficiency by Gandhi et al. in their publications [14], [16], [17] and [13]. Although many of these references are very similar to the modeling work presented in this thesis, the work presented here provides many additions such as investigating both sleep states and DVFS methods under Markovian and non-Markovian assumptions. Our analytical results are also packed with results from simulation as well as experimentation on real systems.

All the above mentioned approaches for implementing sleep modes for energy efficiency are concerned with putting only servers into sleep modes. The reason behind that is that servers and CPUs consume the largest portion of energy among other components in the data center for running out computational tasks. Other approaches propose implementing sleep modes at other components in the data center such as network infrastructure as it consumes approximately 30% of the data center’s energy [48]. Suggested elements to put into sleep include whole network components such as routers and switches, line cards and network interface cards or network bundles and links as proposed in [159], [32], [135] and [189]. Rate adaptation for network links is also another approach that has been explored in [159], [46] and [125].These approaches will not be addressed nor discussed in the context of this thesis as the main focus is on saving energy consumed by servers.

In their experimentations on existing data centers’ servers with real traffic traces for testing sleep modes efficiency, authors of [14] show that sleep modes could achieve up to 50% energy savings if the power used during sleep states is less than half the power consumed during idle states. But as attractive as sleep modes can be for achieving energy efficiency, they are not suitable for all load situations. As transitioning among different system C-states requires a period of transition time that is even higher when the transition is performed between off/on states, sleep modes incur delays which could affect quality of service of applications and users’ SLAs as well as an energy penalty due to power spike at disk spin-up [99]. According to [100] sleep modes are only beneficial when idle periods are relatively long, which is not usually the case in data center environments where servers are usually lightly utilized with small tasks. Also for situations when load arriving to the data center is highly bursty sleep modes will degrade system performance due to the relatively long transition durations [181]. This raises the need for dynamic algorithms for servers to self-scale their CPU frequency according to offered load, as will be explained in the following subsection.

3.1.3 Dynamic Voltage and Frequency Scaling

Dynamic Voltage and Frequency scaling is an approach for reducing the frequency of CPU cycles of servers when idle or experiencing low workloads in order to reduce their operational voltage and accordingly their energy consumption. The amount of power consumed by a server can be estimated using equation 3.1:

$$P = Cfu^2 + P_{static} \tag{3-1}$$

where C is the sum of capacitances across the circuit, f is the frequency at which servers' operate, u is the voltage supplied and P_{static} is the static power consumed due to leakage mechanisms [58]. As the equation shows, the amount of consumed power is directly proportional to the operational frequency and square the operational voltage which is frequency dependent, thus reducing frequency will reduce voltage and cubically reduce the consumed power.

To perform this operation, DVFS reduces the amount of instructions that a processor issues with a period of time, thus reducing its performance by introducing more delays but with an advantage of reducing its energy consumption[181], making it an approach that is only suitable during average to low loads. Accordingly when implemented in a virtualized environment as cloud data centers, a server can only apply DVFS schemes if all its hosted VMs and applications can still maintain their SLAs. As energy depends on both the amount of power consumed as well as its time duration, reducing consumption must be done through reducing consumed power during non-idle periods without highly extending those durations [181]. Depending on the current system's workload, the operating system can choose to enter a state among several P-states (Performance states) which define combinations of voltage and frequency values. CPU P-states and their related interfaces are also defined by the ACPI [30], where a high P-state number indicates a state of low frequency and low power consumption. For example, state P0 indicates a state of highest frequency, P1 has lower frequency, P2's frequency is even lower, etc... Examples of P-states supported by an Intel Pentium M 1.6 GHz processor are shown in Table 3-2 [5] [65].

Table 3-2: P-states of an Intel Pentium M 1.6 GHz Processor

P-State	Frequency	Voltage	Power (Watts)
P0	1.6 GHz	1.484 V	25 W
P1	1.4 GHz	1.420 V	17 W
P2	1.2 GHz	1.276 V	13 W
P3	1 GHz	1.164 V	10 W
P4	800 MHz	1.038 V	8 W
P5	600 MHz	0.956 V	6 W

This method works best when idle periods are frequent and long, otherwise it can result in an energy consumption overhead [172]. Another trade-off for DVFS is that energy should be enough reduced to compensate for the longer time it will take the server to process requests under lower frequency [57]. Nevertheless, DVFS method is considered an efficient method for achieving energy efficiency due to the following reasons [95]:

i. Power consumption is directly proportional to the frequency and to square of the voltage, thus an attempt to reduce both can result in reducing consumed power cubically.
ii. Voltage is directly proportional to leakage current, thus reducing CPU's voltage reduces power lost due to leakage current.

DVFS has been extensively studied in literature for efficiently reducing data centers' power consumption. In [85] authors use DVFS for reducing the sudden power rise encountered at a server during the process of VM migration. As migration of virtual machines is usually performed for reducing consumed energy, this rise in energy contradicts the goal of power capping and thus DVFS prevents this rise by reducing the CPU frequency. In [15] Gandhi et al. consider the problem of allocating minimal power to servers with minimal delays using Dynamic Frequency Scaling DFS, DVFS, and combination of both for optimized energy reduction at different load scenarios. Authors model server farms as queuing systems where they can derive exactly the average delays for requests then find the optimum operating frequencies for maintaining them. Their experiments on an IBM Bladecenter concluded that dynamic DVFS can improve energy efficiency up to 5 times; and that depending on the load type, running more servers at lower frequencies could save more energy than running fewer servers at a higher frequency.

In the scheduling scheme proposed by [64] the scheduler performs regular checks on the processing demands of applications hosted by servers in order to adjust their energy consumption dynamically via DVFS. Another scheduling algorithm proposed in [199] also uses DVFS for reducing energy consumption by monitoring the system during intervals and setting the CPU frequency of servers during the next interval based on the energy consumed during the previous interval. Another scheduling algorithm with DVFS add-on is introduced in [45] where VMs are given weights based on their priorities and SLAs of its users then distributed among servers in the DC while DVFS algorithm regulates each server's frequency accordingly. Through simulations authors show that they can increase system's utilization and prevent wasted resources as well as wasted energy during idle periods. Authors of [94] propose another data center provisioning algorithm for scheduling VM in data centers with real time requirements while applying DVFS schemes. Authors propose and test Adaptive-DVFS and δ-Advanced-DVFS that work under soft and hard real time requirements and are able to minimize energy consumption, maintain users SLAs and reduce running costs of data centers. In [143] authors address the problem of conflict between memory system and CPU DVFS when controlled by separate entities. Their proposed algorithm 'CoScale' provides coordination between these two tasks for each of the server's cores while maintaining defined performance levels.

More often in literature DVFS algorithms are implemented along with DPM for optimum power reduction, where DVFS can be used at durations of low utilization and DPM at idle durations. Virtual Batching approach [195] is one example; it batches groups of arrivals together so that a server can execute the batch then puts the server in a deep sleep mode until another batch is followed again. The frequency of the server while serving batches of requests is adjusted according to batch size and arrival rates using DVFS. Another example is introduced by Dhiman et al. in [62] where they define a group of experts each has a set of DPM and DVFS policies and is suitable for certain load condition. Authors introduce an online algorithm that learns the current system load and brings the most suitable expert into action with consideration to energy, performance, and users' SLAs. A similar approach is introduced in [116] where instead of the online algorithm a machine-learning algorithm is used that selects the most suitable expert while providing a theoretical guarantee on the overall system performance. A different approach is introduced by authors of [101] where they model the problem of selecting which servers to be switched-off and others that will be switched-on as well as their operational frequencies as a mixed Integer Programming problem which is solved under energy efficiency and quality of users' experience constraints.

Despite the advantages it provides, DVFS technology faces a few challenge that face system administrators during real life implementations, as explained in [11]:

i. Earlier CPU models were relatively much simpler compared to today's complex CPU architectures which provide advanced features as multi-level caching, pipelining, etc., which makes predicting the level of frequency at which CPU should operate at non-trivial.
ii. The quadratic relation between power and voltage implied by Equation 3-1 is not always the case for modern processors. In [185] authors explain that some processors have different supply voltages for different processor elements, thus reducing one element of these will not have a significant effect on reducing the overall power consumption.
iii. As explained earlier that applying DVFS results in extended execution times, the relation between these two factors is not always linear and could cause execution time non-literalities' [61] as well as alterations in order of task-execution [185].
iv. Modern processor architectures tend to minimize the dynamic power range, which reduces the effectiveness of DVFS approach [57].

Thus DVFS technology needs to be implemented with caution while taking all above arguments into consideration. Following Table 3-3 provides a summary and comparison for the main algorithms introduced in this chapter that target power efficiency in data centers through DPM and DVFS approaches, or both combined.

Table 3-3: Summary of DPM and DVFS Algorithms

Algorithm	Type	Description	Pros	Cons
PowerNap Meisner et al. [53]	DPM	Processes requests at highest performance to switch-off servers as soon as possible, and wakes them up as soon as a request arrives	Race-to-halt approach quickly puts system in sleep mode	Relatively long delays due to transition times between sleep/active states

DreamWeaver Meisner at al. [54]	DPM	Schedules tasks such that all processors have idle and busy periods at the same time so that system-wide sleep state is possible	-Allows for longer sleep durations by stalling arrivals without SLA violations -Workload dependent	Additional delays caused by extended sleep durations
NapSAC Krioukov et al. [23]	DPM	Uses Heuristics for predicting upcoming load to activate servers only as much as load needs and put others in sleep state	-Load dependent -Pre-emptive algorithm thus transition delay does not affect users' SLAs	High processing for high accuracy prediction heuristics
Lin et al. [118]	DPM	Provides an online algorithm for right-sizing the number of active servers in a data center where online algorithm is motivated by optimal offline algorithm	-Energy savings due to right sizing -Algorithm has low computational complexity	Model experimentation proved to be efficient only for general workload and delay situations.
Niyato et al. [55]	DPM	Formulates a constrained Markov decision process for deciding the state of each server then distributes incoming requests among active ones	Algorithm decisions minimizes energy consumption as well as network costs	Complexity of the algorithm increases with increasing number and size of data centers
Herlich at al. [115]	DPM	Introduced random delays before waking up servers from sleep state or putting them to sleep again	Fine tuning of parameters can lead to reduced energy consumption and latency	Only average values for delay durations were studied, no upper or lower boundaries provided
Jeong et al. [85]	DVFS	Reduces CPU frequency during VM migration process in order to reduce the encountered energy spike	Maintains power capping goals by preventing power spike	Can only be applied to processors supporting DVFS

Wu et al. [45]	DVFS	Introduces a priority scheduling algorithm by giving weights to VMs according to SLAs of its users, then adjusts servers' CPU frequency using DVFS	Performs server consolidation thus prevents wastage of power at idle times	Provisions the data center for minimum VM requirements, and will not be efficient at peak load durations
Gandhi et al. [15]	DVFS	Models server farms as a queuing system whose solution allows for calculating mean delays, then finds adequate operating frequencies for maintaining these delays	- Solution is dynamic to any workload -Precise delay bounds from exact analytical solution	Doesn't allow sleep states when servers are idle, thus power is wasted when server has null load.
Kim et al. [94]	DVFS	Proposes algorithms for provisioning data centers with soft and hard real-time requirements with DVFS requirements	Achieves energy efficiency in the data center while reducing its running costs	Does not consider current system workload
Virtual Batching Wang et al. [195]	DPM & DVFS	Groups arrivals into batches and puts server into sleep after a batch is processed, where frequency of processing is varied dynamically by DVFS	Ability to save energy at idle and busy durations with dynamic adaptation to work load	Does not operate/scale well under bursty load conditions
Dhiman et al. [62]	DPM & DVFS	Defines a group of experts with different DPM and DVFS polices to match all system's conditions	Solution is dynamic and is able to optimize system's performance under all load situations	Algorithm is targeted for physical servers and was not proven effective in virtualized environments
Bertini et al. [101]	DPM & DVFS	Uses mixed integer programming for determining which servers will be on/off as well as operational frequencies while maintaining users SLAs and reducing energy	Algorithm is tested and compared against several others proving its efficiency in de/centralized approaches	Capability for providing the same energy efficiency for virtualized environments has not been investigated

3.2 Load Balancing between Cloud DCs

As demand on cloud computing services increases, cloud providers tend to increase the number of DCs they own and increase their capacity of servers and DC equipment to accommodate the increased demand and avoid bottleneck situations. Typically a cloud provider has more than one DC at which customer's requests are received and processed, and the load among these DCs is usually variable. Requests arrive to each DC depending on the type of requests, geographical proximity, or according to a selection criteria imposed by the cloud provider. However, these decisions could leave some DCs highly overloaded causing performance degradation while other DCs are lightly loaded causing poor resource utilization. Accordingly, load balancing is required to take place between DCs to avoid situations of DCs overloading and low utilization. Load balancing involves decisions on how to distribute requests among different DCs or servers, as well as migrating of requests from one DC to another in case of an overload.

3.2.1 Overview on Load Balancing

Load balancing between cloud DCs aims at optimizing resource utilization of the DCs in order to enhance overall system performance. Load balancing aims to achieve all or most of the following goals:

i. Avoiding overload: Since at overload situations the overall system performance tends to degrade, load balancing strategies aim at reducing/preventing overload situations by shifting additional loads to other lightly loaded DCs. This process helps in improving the overall system performance as well as providing better resource utilization.
ii. Enhancing offered service: As a request arriving to a heavy loaded DC could receive a better response time if it was migrated to another lightly loaded DC, decisions of keeping or migrating a request for receiving better quality of experience are taken by load balancing algorithms according to request's SLAs. Enhancing the response time for each request will lead to enhancing the makespan of the whole system, which is defined as the maximum finishing time among all received requests in the system per time [75].
iii. Overcoming fail-over situations: These are situations when one or more DCs fail during processing jobs, or a job fails to process on a certain DC. Load balancing algorithms should be able to take the decisions of how to resolve the faulty situation or where to migrate the jobs that were under processing so that their response times are minimally affected. Fault-tolerant load balancing algorithms should be able to detect when the DCs that have been under fail-over are able to receive requests again in order to assign requests for these DCs again.

   Fault tolerant algorithms can be implemented in one of two ways, as explained in [63]:

a. Proactive Algorithms
   These algorithms tend to predict failures before they happen and take corrective actions to prevent them. Methods for implementing this approach include periodic reboots, requests replication on various VM, and pre-emptive migration.
b. Reactive Algorithms
   These algorithms correct fail-over situations after they occur. This could be done by one of the following approaches: system restart from check points, resubmission of the job to the same DC or migrating it to another running DC.

3.2.1.1 Types of Load Balancing Algorithms

Authors of [21] and [28] classify load balancing algorithms into two major categories:

i. According to system load
   Depending on the how requests are distributed among DCs, algorithms in this category follow 3 different approaches:
   a. Centralized Approach
      Where a single central node receives all the requests and is responsible for distributing them among all other system nodes. This approach is fast and more efficient as distribution decisions are taken by one node only, however it carries the risk of single point of failure or bottlenecks in case this node is down or congested. This issue could be resolved at an additional cost by adding redundant nodes to the central node to replace it whenever required.
   b. Distributed Approach
      In this approach nodes communicate their loads to each other so that each node can form its own load vector. Upon receiving a job request, each node decides locally whether to keep or migrate this request based on its load vector. Although it requires more communications and data transfer between nodes, distributed approach works best with widely distributed systems. In huge systems with large and diverse number of nodes it is more efficient than centralized approach which imposes a huge overhead on the central node.
   c. Hybrid Approach
      Taking advantage of both approaches' benefits, a hybrid approach deploys both strategies of having a centralized node and a load vector at each distributed node.
ii. According to the system topology
   Algorithms that distribute load according to system topology and nodes' information status follow 3 approaches:
   a. Static Approach
      Following static defined rules, static approach distributes the load in the same way according to a set of predefined rules based on the nodes' processing capabilities, memory and storage capacity. These rules do not take the current system load into account, which makes this approach appropriate only for stable systems with low load variation, unlike distributed cloud systems.

Although it has the advantage of minimal processing for load distribution, less resource utilization and more predictability, static approach may overload a node with requests leaving other nodes lightly loaded, as its decisions are static and unaffected by the state of nodes [164]. Another disadvantage of static approaches is that they cannot resolve situations of DC failure, since no updates are taken into consideration during algorithm run-time, which makes them less reliable.

b. Dynamic Approach

Unlike static approach, decisions taken by dynamic approach are different each time a job arrives, as it depends on the variable current system state. Dynamic approach is more suitable for distributed systems as cloud computing, since decisions of job allocations must take the current system state into consideration to avoid overload. Although they have higher complexity and utilize more resources than static algorithms, dynamic load balancing algorithms are more adaptive, efficient, reliable and provide better load balancing decisions [164].

c. Adaptive Approach

According to system status, adaptive approach decides whether to use static algorithms, dynamic algorithms, or combination of both for load distribution.

Among all before-mentioned classifications the most suitable type for a cloud environment are distributed dynamic load balancing algorithms. This is due to their adaptability, reliability and suitability for a diverse and constantly changing cloud networks. According to [191], these distributed dynamic algorithms can be further classified into two categories:

i. Cooperative Algorithms

All nodes running cooperative algorithms in the distributed system work together at taking load balancing decisions while considering each other's states and resources. Decisions taken by cooperative nodes help increase over-all system performance.

ii. Non-Cooperative Algorithms

Non-cooperative nodes make their own decisions regarding workloads without any consideration to other nodes or over-all system goals; the only factor taken into consideration is the node's own resources. Although non-cooperative algorithms might enhance node's own performance, it might not make the best decisions for the overall system.

3.2.1.2 Challenges of Load Balancing between Cloud DCs

To achieve the before-mentioned targets of the load balancing process, algorithms in literature consider the following aspects listed by surveys in [21] and [90] for deploying optimal solutions for the load balancing problem:

i. Nodes' Spatial distribution and Scalability

As cloud DCs are usually spatially diverse, load balancing algorithms need to account for some side effects such as the distance between nodes and the speed of transmission links between them, which will affect the migration time of requests between the DCs and accordingly affect load balancing decisions. Algorithms should also be scalable to balance the load between any finite numbers of nodes.

ii. Data Storage and Replication

Requests arriving to a node require information stored locally at this node to be processed. In case the load balancing algorithm decides to migrate this job, it has to make sure that either this data is also available at the other DC or that the time to transfer the request along with its data would not violate the delay limit of the request. For enhanced performance, load balancing algorithms perform full/partial data replications at the nodes ahead of time to prevent delays resulting from this task at the time of request arrival. Full replication has a higher cost as it requires more storage since all nodes will carry the same data sets. On the other hand, partial replication makes a better solution by only replicating a part of the data at each node based on its processing capabilities and storage capacity. Although this approach provides better utilization, it makes the algorithm more complex as the algorithm needs to be aware of all the data sets at each node.

iii. Algorithm Complexity

Complexity of load balancing algorithms is required to be as low as possible to avoid delays resulting from processing of complex algorithms. Delays also result from algorithms that require gathering lots of information or requiring long communications. As these delays increase, efficiency decreases along with overall system performance, which is the main reason why algorithms' complexity must be minimized.

iv. Point of Failure

Centralized load balancing algorithms are designed such that a central node takes all load balancing decisions. These central nodes are considered as single points of failure which takes down the whole system if this node is down. Although centralized algorithms are faster and more efficient at the decision making process, the single point of failure issue must be resolved. One approach is to have redundant nodes to replace the central node in case of its failure, but redundancy requires that these nodes always have the same information replicated all over them to be able to replace the central node at any point in-time. Another efficient solution for widely distributed systems are distributed algorithms, which eliminate the risk of single points of failure by allowing individual nodes to take their own load balancing decisions, as will be explained later in this section.

v. VM Migration

As load balancing decisions frequently request that a job is migrated from one DC to another, this process needs to be done without any or minimal interruption to the current running VM. As specified by the virtualization layer, a virtual machine mainly consists of a set of files that could be transferred easily to any DC. However, this transfer process could be done in two ways: Cold transfer or Hot Transfer. Cold Transfer specifies that the VM is paused, transferred, and then continue processing at its new destination; while Hot Transfer specifies that the VM is transferred without any interruption to the VM. Load balancing algorithms need to adopt either one of the two migration approaches to guarantee minimal disruption for the migrated VM.

vi. Automatic resources provisioning
One of the steps taken by load balancing algorithms is allocation and release of resources on demand based on migration decisions. This task of automatic resource provisioning should be very well handled by load balancing algorithms, quickly and efficiently.

vii. Energy consumption
While taking load balancing decision between DCs, energy consumption of DC resources must be taken into consideration. As will be later proposed by the algorithms in this thesis, the same performance level of the DC as well as service level agreements of requests must be guaranteed while reducing/maintaining the amount of consumed energy.

viii. Data Security
Security of data stored in the cloud remains one of the most important research topics in cloud computing. As user's data often have defined credentials in terms of where to be stored or processed, these credentials must be taken into account before migrating this data to other DCs.

3.2.2 Examples of Load Balancing Algorithms

This section will showcase some of the most commonly known load balancing algorithms between cloud data centers. Each algorithm is briefly explained and discussed, followed by a summary and comparison between all algorithms in Table 3-4.

- Round Robin Algorithm
  Requests arriving at the controller will get assigned to candidate DCs in a rotating order. Once a job is assigned to one DC, the other DCs take their turn in accepting requests so that a new request is not assigned again to the first DC until all other DCs have at least one request. Since requests do not necessarily have the same processing times, Round Robin algorithm doesn't maintain a fair distribution among servers/DCs [28]. The same criteria is followed when a DC needs to migrate a request, it chooses where to migrate the request by following a list of DCs in a circular manner. Figure 3-1 shows how requests represented as Rx are distributed among DCs represented as DCx according to the Round Robin Algorithm.

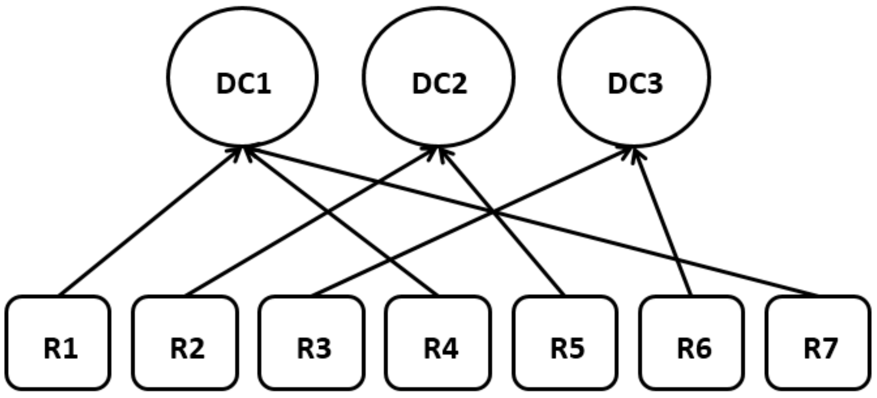


Figure 3-1: Process of Round Robin Algorithm [75]

- Weighted Round Robin Algorithm
  A modified version of round robin algorithm; where DCs are assigned weights according to their computational power and resources. DCs with higher processing capabilities are assigned more requests according to their weights as they are able to process requests quickly and efficiently. This assignment also occurs according to a list in a circular manner. Although weights are an enhancement to the algorithm, it still has the disadvantage of not taking requests' processing times into consideration. For example; a DC with weight 2 might be assigned two relatively small requests, while another DC with weight 1 is assigned one relatively large request that should have better been assigned to a DC with higher weight for providing better service.
- Dynamic Round Robin Algorithm [112]
  Another modified version of round robin algorithm with an extension for power saving. This algorithm has two additional rules which enhance power management capabilities of cloud DCs:
  1. If some requests being served on a DC have finished and some are still being processed, this DC switches to a 'retiring state'. This state means that it accepts no further requests, and that it will shut down as soon as it is done with the requests currently being processed.
  2. If a DC stays in a retiring state for a long time exceeding a defined 'retirement threshold', requests being processed at this DC are migrated to another DC so that this DC can be shutdown.
- Randomized Algorithm [75]
  This algorithm assigns VMs to DCs in a random order, without any defined order or knowledge about the status of the DCs. Although random decisions require low processing overhead and thus low response times, this algorithm might result in overload situations if jobs are assigned to overloaded DCs. Figure 3-2 shows an

example of an overload situation; where DC2 is overloaded with requests due to random assignments, while DC1 and DC3 are under-loaded.

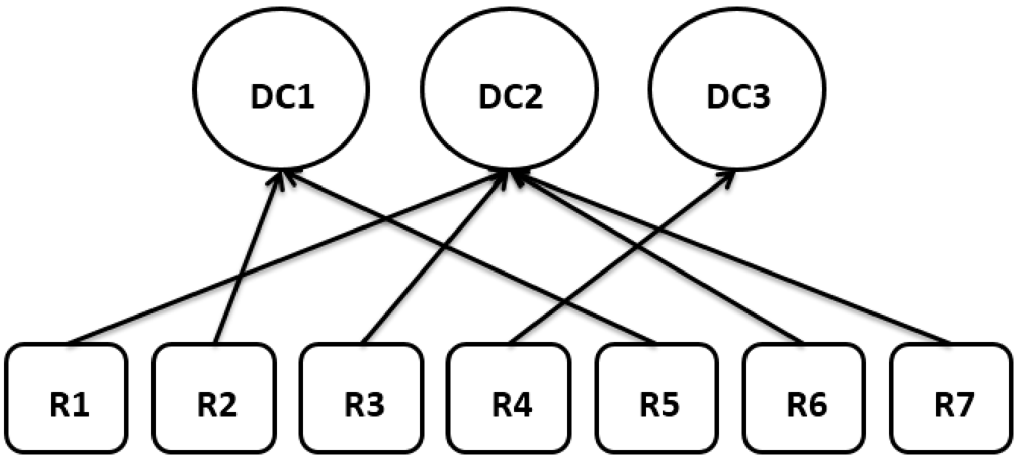


Figure 3-2: Process of Randomized Algorithm [75]

- Minimum Completion Time Algorithm

  This algorithm decides where to distribute requests according to the request completion time each DC can offer. It scans all available DCs checking the completion time of the request at each one, and chooses the DC that offers the least completion time. Figure 3-3 illustrates how the process of DC selection is done, taking into consideration that the main selection criteria for minimum execution time is the processing speed of the DC along with its current load [75]. Although this approach guarantees minimum execution time for the request, it will overload DCs with high processing capabilities, while others with low processing capabilities will receive much fewer requests [49]. According to experimental studies done by simulation in [75], Minimum Completion Time algorithm achieves almost 100% throughput at low number of requests, where throughput is defined as the number of requests processed during time specified by their SLAs. It also achieves low makespan values since this is the main algorithm criteria. As the number of requests increase, the throughput drops and makespan duration increase due to DC overload.

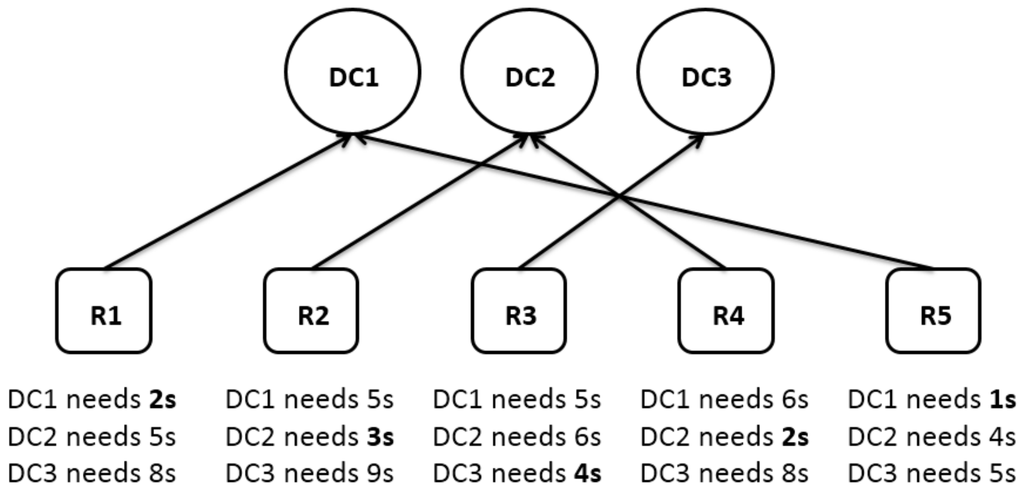


Figure 3-3: Process of Minimum Completion Time Algorithm [75]

- Opportunistic Load Balancing Algorithm
  OLB algorithm assigns requests to the first DC expected to become available. It checks all available DCs for their remaining times to become available for serving the new request, then routes the request to the one with minimum waiting time, as illustrated by Figure 3-4. In the figure each DCx load represents the time a job has to wait before being served by this DC, so the shortest waiting time determines the DC to be selected. As OLB algorithm tends to keep all DCs busy, it has a shortcoming of not considering the execution time of the request on the selected DC. So despite that a request might not wait a long time before starting to execute, it might take long execution time causing other requests to wait longer, increasing the makespan of requests [168].

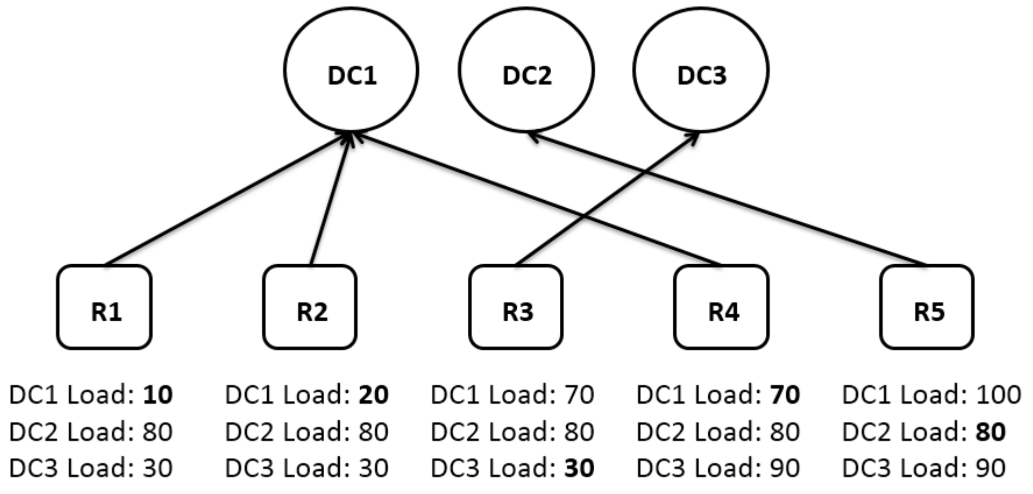


Figure 3-4: Process of Opportunistic Load Balancing Algorithm

- Min-Min Algorithm [168][42]
  This algorithm is based on Minimum Completion Time algorithm, but instead of scheduling one request at a time, it considers a set of unmapped requests. The algorithm starts by finding the minimum completion time for all requests that need scheduling, where all DCs are available to serve any of them. Once the list is sorted out, the request with minimum completion time is assigned to the corresponding DC that will provide this minimum completion time. The list then is updated again while excluding the scheduled request and adding its execution time to the execution time of all other requests at the DC busy serving it. As a result of this selection, if the number of small requests is bigger than large requests the makespan will be expanded by large requests [163].
  Authors in [42] enhance the performance of Min-Min algorithm by introducing LBIMM (Load Balancing Improved Min-Min) algorithm which reduces the execution time of each resource and improves imbalance resulting from consuming resources with high computational power in serving small requests, while large requests remain with DCs with lower resources. Another enhancement introduced in [42] is by accounting for user priorities, where users are classified into two groups, one for high priority requests and another for low priority ones. PA-LBIMM (User-Priority Aware Load Balancing Improved Min-Min) algorithm performs Min-Min algorithm on the group of high priority requests first, followed by the second group; which guarantees better offered service for high priority requests.
- 2-Phase Load Balancing Algorithm
  This algorithm was proposed by authors of [166] based on two already known load balancing algorithms: Load Balancing Min-Min (LBMM) and Opportunistic Load Balancing (OLB). In the first phase OLB is used to distribute requests among DCs keeping all DCs in working state. In the second phase LBMM is executed at each DC to make sure the execution time of the request is minimized. Benefits of the 2-phase algorithm include better overall execution times for requests as well as more efficient load balancing [167].
- Max-Min Algorithm
  It operates in the same manner as Min-Min algorithm, but Max-Min schedules larger requests with maximum execution time first to the DC that will guarantee their minimum completion time. This criterion aims at reducing penalties occurring from processing of requests with long execution time [168]. As large requests are processed first, it allows smaller requests to be processed meanwhile simultaneously on other DCs [49]. Max-Min algorithm out-performs Min-Min algorithms when short requests are much more than long requests. When this is not satisfied, the large number of large requests executed earlier will increase the system’s makespan [163].
- Resource Aware Scheduling Algorithm (Duplex)
  Duplex algorithm deploys both Min-Min and Max-Min algorithms alternatively, making use of the benefits of both algorithms [163]. Since Min-Min algorithm is more efficient when the number of small requests is less than larger ones, and Max-Min algorithm performs better when small requests outnumber larger ones, Duplex

algorithm deploys either one of these algorithms to enhance the system's performance. As a result of alternative execution of the two algorithms, neither small requests nor large ones wait for long times, thus achieving smaller makespans at various load situations and different scales of distributed systems [163].

- Join Idle Queue Algorithm
  This algorithm starts by distributing requests among idle DCs first until all of them are occupied, then requests are assigned to DCs with the least number of jobs so that average queue size at each DC is minimized [194], as illustrated in Figure 3-5. Since these decisions of finding the least occupied DC can be made ahead of a request's arrival, this algorithm significantly reduces the communication overhead during request's processing time. However, it only considers the number of queued jobs for its decisions and not their sizes, which might mislead the decision for a shorter queue having x large requests and taking more processing time than another longer queue with smaller requests. This issue is considered by the algorithm proposed in this thesis later in Chapter 5.

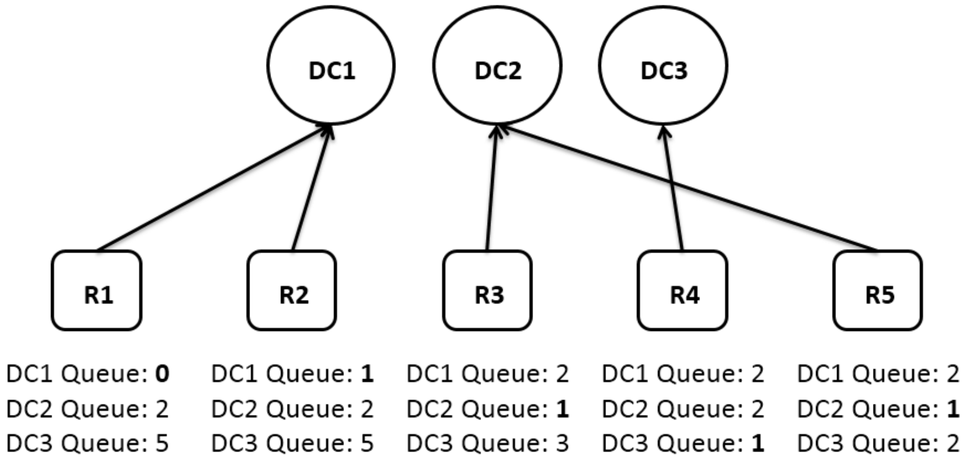


Figure 3-5: Process of Join Idle Queue Algorithm

- Power Aware Load Balancing Algorithm (PALB)
  Aiming to provide an algorithm that minimizes the amount of wasted power by unutilized algorithms, authors of [82] introduce the PALB algorithm. Their approach is to keep track of utilization percentages of all compute nodes, and accordingly decide how many DCs are actually needed for the current load. Un-needed DCs could migrate their jobs and switch-off, which according to tests done by the authors; can save up to 79% of the amount of consumed energy if compared to other non-power-aware scheduling mechanisms.
- Double Threshold Energy Aware Load Balancing Algorithm (DT-PALB)

This algorithm was proposed by authors of [78] as a means of balancing the load among DCs in a way that conserves consumed energy. The algorithm has three phases [167]:

1. Load balancing phase: The choice of where to place or migrate a request is done based on how much DCs are utilized. If nodes have a utilization percentage between 25-75%, then the least utilized DC will be chosen.
2. Upscale phase: At peak periods when all current DCs are utilized above 75%, the algorithm powers on additional DCs to serve the additional requests.
3. Downscale phase: when any of the DCs' utilization falls below 25% the algorithm migrates all the VMs on this DC to be able to switch it off.

Although this algorithm proposes an efficient method for load balancing at high and average load situations, it can cause performance degradation at low loads. Since DCs' utilization is likely to drop below 25% at low load situations, the number of migration processes will significantly increase which reduces performance and increase costs. [167]

- Enhanced Equally Distributed Load Balancing Algorithm

This algorithm proposed in [157] aims at distributing requests equally among available resources. It allocates resources to the least loaded DC aiming to achieve fastest response time for the request. Each DC is assigned a counter variable representing the number of requests currently being handled by the DC. This counter is checked by the algorithm every time a request needs to be migrated/assigned, and the chosen DC is the one with the smallest counter. Whenever a new request arrives to the DC the counter is incremented by 1, and whenever a request finishes its service the counter is decremented by 1. Although this algorithm balances the load equally among DCs providing low response times and high resource utilization, it doesn't consider requests' weights [112] (all requests affect the counter by 1 step up or down). This could lead to load imbalance when one DC carries 10 small requests while another is serving 10 larger requests.

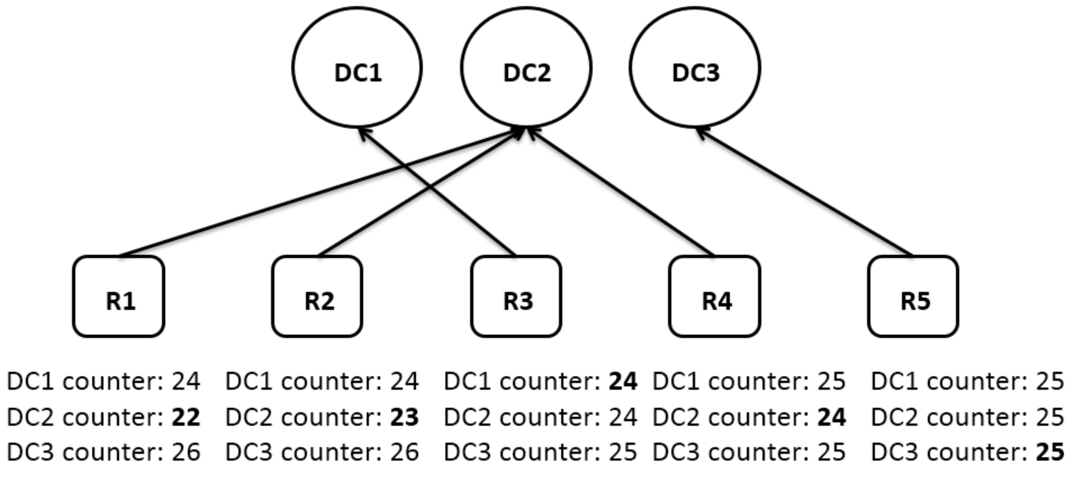


Figure 3-6: Process of Enhanced Equally Distributed Load Balancing Algorithm

- Decentralized Content aware Load Balancing Algorithm
  Based on workload and client aware policy (WCAP), this decentralized algorithm proposed in [66] distributes requests based on their content to the most suitable DC to process them. Each request is defined by its UPS (Unique and Special Property), which is used to select the target DC for the request. According to this criterion, the selected DC is the one that is able to provide best processing capabilities in lowest time for the requests, because it receives request according to its capability and specialization. Content information is used to search for the most suitable DC, which improves and speeds up the search process, improving overall system performance [126].
- Enhanced MapReduce Algorithm [106]
  MapReduce [170] is a distributed data processing architecture which follows two main steps in assigning requests to DCs: request mapping and reducing results. The algorithm implements 3 methods [90]: 'part' which initiates partitioning and mapping of tasks into smaller parts, 'comp' which compares the different parts, and 'group' which reduces tasks by grouping similar parts together. This method is enhanced by authors of [106] who propose to add a load balancing layer between the Map and Reduce tasks to prevent partitioning of small tasks and only divide large ones [90].

More algorithms for task-scheduling and load balancing could be found in literature but are not studied here for their complexity and for not being the best solutions for the situation targeted in this thesis: Balancing user requests between cloud DCs. A few of these algorithms are listed below shortly where they are also known from optimization theory:

- Genetic Algorithm : mainly used for large solution spaces [168], this algorithm subdivides a task into subtasks generating a population, then performs four main steps (Selection, crossover, mutation and evaluation) until the best solution could be found [108].
- Simulated Annealing: A recursive algorithm that also divides tasks into smaller subtasks, but considers all possible solutions for a subtask, even the poor ones. This algorithm is based on 'System Temperature' which cools down after each iteration eliminating more and more poor solutions. Algorithm stops and an ideal solution is found after the system temperature drops near zero or after 150 iterations. [168]
- Tabu Search: An algorithm based on solution space search that keeps track of already visited solution regions so that they are not repeated again. It initializes with a random mapping generated by uniform distribution, and then performs 'short hops' into unvisited solution regions trying to find better solutions [71].
- Optimal Scheduling Algorithm A*: is an algorithm based on binary tree search technique [168]. It starts with a root node that represents a void solution, and starts producing child leaves representing possible solutions. At each level of the tree the best possible solution is chosen to become parent node, produce child nodes, become inactive, and continue searching down the tree [92].
- Honeybee Foraging Algorithm

Inspired by nature for self-organization, this algorithm was introduced by authors in [158]. It resembles the behavior of honeybees when foraging bees are sent looking for food sources, then return to the hive foraging the distance to the source and its quality. Performance of this algorithm is enhanced by system's diversity, but not with the increased system size [126].

Table 3-4: Comparison and Summary of Load Balancing Algorithms

Algorithm	**Type**	**Description**	**Pros**	**Cons**
Round Robin	Decentralized [50], Static	Requests are assigned to DCs in a circular manner.	Even distribution of requests among DCs [167].	Doesn't consider tasks' processing times.
Weighted Round Robin	Decentralized, Static	DCs are assigned weights according to their processing capabilities. High weights means more received jobs.	Jobs are assigned according to DCs' capabilities. Better resource utilization [49].	Requests' processing times are not considered.
Dynamic Round Robin	Decentralized, Dynamic	Defines two rules for reducing DC's power consumption based on retiring state and retirement threshold	Allows for better power management in DCs	Not scalable for large number of DCs [112]
Randomized	Decentralized, Static	Assigns tasks in a random order	Low complexity, short response time	Status of DCs is not considered, could result in overload
Minimum Completion Time	Decentralized, Dynamic	Assigns tasks to DC offering minimum task execution time	High utilization, minimum execution time for requests	Overloads DCs with high processing power
Opportunistic Load Balancing	Decentralized, Dynamic	Assigns requests to the first DC expected to become idle	Simplicity [168]	Doesn't consider request execution time, Longer makespan

Min-Min Algorithm	Decentralized, Static	Assigns requests according to minimum completion time at corresponding DC	Fast [49], Schedules best case first	Doesn’t consider current DCs’ load, Load imbalance [42]
2-Phase Algorithm	Decentralized, Static	Based on LBMM and OLB algorithms	Better execution time, Enhanced load balancing [167]	Works only in a static environment [167]
Max-Min Algorithm	Decentralized, Static	Schedules large jobs first to the DCs guaranteeing minimum completion time	Better load balance, Larger requests get served at more powerful DCs	Small jobs wait longest
Duplex Algorithm	Decentralized, Static	Combines best of Min-Min and Max-Min Algorithm	Minimal overhead, High performance [49]	Static Algorithm
Join Idle Queue	Decentralized, Dynamic	Assigns requests to DCs with minimal number of requests	Low response time, no overhead at job arrival [112]	Requests’ sizes are not accounted for
Power Aware Load Balancer	Decentralized, Dynamic	Distributes DCs in order to maintain lowest power consumption by DCs	High availability of compute nodes, reduced power consumption [82]	Algorithm could result in unnecessary migrations at low loads
Double Threshold Energy-Aware Load Balancer	Decentralized, Dynamic	Balances load and saves energy in 3 phases: Load balancing, Upscale and downscale	Efficient Load Balancing	Unnecessary migrations at low loads degrade performance [167]
Enhanced Equally Distributed Load Balancing	Decentralized, Dynamic	Requests are assigned to the DC with lowest current load.	Low response time, increased utilization	Weights of requests are not considered.

Decentralized Content Aware Algorithm	Decentralized, Static	Distributes requests based on their content to DCs that serve them best	Improved search for DCs, Increased system performance	Static Algorithm
Enhanced MapReduce Algorithm	Decentralized, Static	Based on mapping tasks and reducing task results	Load balancing layer reduces overhead for reduce tasks	High processing time

According to tests and simulations done by [75], it can be concluded that no single algorithm would work best for all cloud environments. Although some algorithms are superior to others in many aspects; there would always be a compromise in terms of cost or performance. For example: algorithms such as round-robin, weighted round-robin and similar tactics all have the same approach of distributing load according to how much requests servers have received so far, but they all ignore the fact that the current number of received requests is no indication for the system state while a solid indication for system state is the remaining idle resources of the servers [150]. This applies to all other static load balancing algorithms, which ignore the current system state in their decisions. Dynamic algorithms provide better options in this issue, however there are very few available examples in literature, and according to [35] none of them considers the future state of the system or the effect of migrating requests to it, and only a few algorithms consider the energy efficiency of data centers.

In Chapter 6 of this thesis two novel load balancing algorithm will be introduced that avoid most of the mentioned drawbacks. Both are dynamic algorithms which distribute load based on current system state and consider the future state of the system after this request arrives. Unlike most approaches in literature that define load balancing as an approach to balance the load equally among all DCs, in this thesis a strategy of ‘load un-balancing’ is adopted. These algorithms tend to perform request consolidation on a fewer number of DCs without violating SLAs of requests in order to minimize the number of active DCs and switch-off idle ones for energy efficiency considerations. The same ‘load un-balancing’ strategy has been studied by a different approach by authors of [44] supporting the hypothesis of un-even load distribution among DCs for satisfying users’ SLAs or for enhancing DC performance. Algorithms are analyzed by studying system of two DCs operating by the respective algorithm and modeled using Markov Chains, where algorithms for solving system’s stead-state probabilities are explained. Similar work has been introduced by authors of [175] following similar analysis method for dynamic routing networks using three multi-server DCs with special mutual overflow among them.

Chapter 4 Methodology

Having explained the main concepts of cloud computing, related technologies as virtualization, migration and server consolidation as well as challenges faced by cloud developers as energy efficiency and load balancing, the second part of this thesis proposes novel models for solving these challenges. In the context of this chapter the methodology by which cloud DCs are studied and analyzed are introduced and explained. Section one explains mathematical models used to represent cloud DCs as queuing systems and perform steady state queuing analysis to study the system and predict its performance under different workload situations. Notations used within all DC queuing models will be addressed, followed by the assumptions that were made to allow for modeling data centers as queuing systems using Markov Chains. Section two introduces a simulation platform used to test and verify the modeled DCs as queuing systems using OMNeT++ simulator. Simulation defines entities that comprise a basic queuing system, such as source, queue and servers with different configurations according to each implemented model, where these entities and their interconnections are defined and explained. Contrary to the mathematical models solved by stochastic queuing theory, simulations are experimental methods for performance evaluation executed by artificially generated events and measurements by a computer program. The last Section three of this chapter a real test-bed of a minor data center set-up using two servers is illustrated, where proposed algorithms can be benchmarked to verify results of analytical models and simulations.

4.1 Mathematical Models

In this context a data center is described as a queuing system with multiple servers to serve different requests simultaneously, where all servers have a common queue at which arrivals are inserted first before a controller decided if the arrival will be served at this DC and at which sever, will be queued, or will be migrated as proposed by load balancing algorithms. This approach is accurate for modeling data centers as they have several servers hosted in racks, and requests arriving to the DC requesting SaaS, PaaS or IaaS services are hosted at any server with sufficient processing capacity by creating a virtual machine to serve this request. For illustration of the explained model components refer to Figure 4-1. The models proposed by this thesis are based on previous work by the main supervisor of this work and its author and published in [119], [134], [138], [127], [140], [120] and [141] where previous work constitutes the building blocks for the complete and more general models in this context. Models introduced within this thesis describe the DC at any time instant by a state (x, z), where x is the number of active servers in the data center and z is the number of queued arrivals waiting to be served. The control of the queuing system follows from a Finite State Machine (FSM) for states (x, z). System is initially at state $(0,0)$ where no servers are active and no arrivals are waiting for service. The state of the data center changes whenever an arrival/departure/activation event happens, as will be explained later per each model. When an arrival occurs, the DC Control decides whether this arrival will either be immediately served or queued according to the current state of the system. Buffered frames are organized in the queue in the order of their arrival, and are served strictly according to a service strategy, e.g.: First-In, First-Out (FIFO) strategy. The queuing system reaches its maximum capacity at state (n, s), where n is the number of available servers in the data center, and s is the maximum buffer capacity. Arrivals occurring when the system is full are assumed to be lost; or as some models require, migrated to another DC for load balancing purposes. As soon as a server becomes idle, it will either be re-occupied by the frame at the head of the queue, or it will be deactivated in case of an empty queue. All these decisions are derived from the FSM of the controller and communicated to the scheduler which updates the state of the queue and servers accordingly.

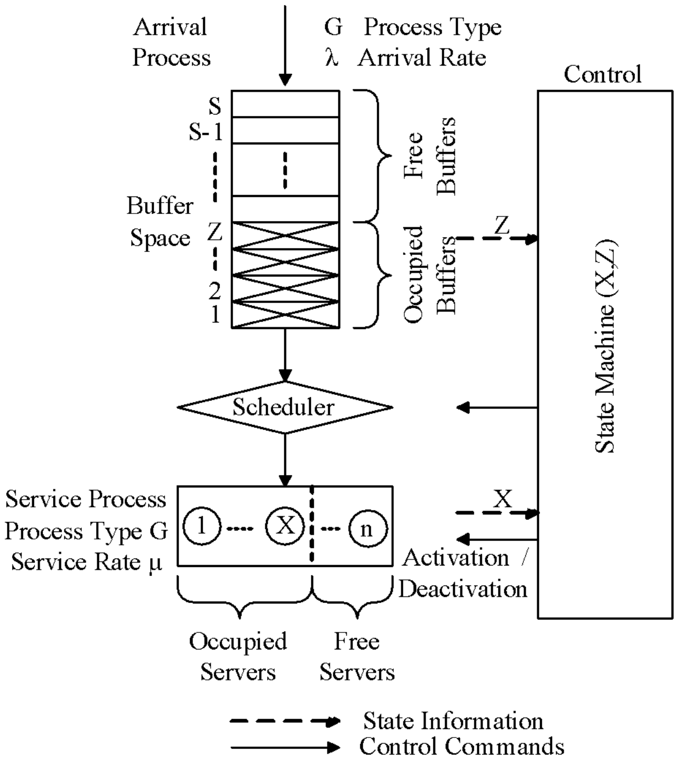


Figure 4-1: Generic Model for a Data Center with Dynamic Activation/Deactivation of Servers

At any state (x, z), if an arrival happens that will cause a server to be activated, system state changes to $(x + 1, z)$; but when a new arrival occurs and is scheduled to be buffered, system state changes to $(x, z + 1)$. When an arrival finishes its service time, the server it had occupied will be either reoccupied by another packet from the head of the queue, changing the system state to $(x, z - 1)$ as the queue size is reduced by one, or if the queue was empty the server will be deactivated and the system state will change to $(x - 1,0)$. Arrivals to the DC are assumed to occur according to a general distribution function with arrival rate λ, where interarrival times have an average value of $1/_{\lambda}$. Service times of each server in the DC are also assumed to follow a general distribution function having an average of ${}^{1}/_{\mu}$, where μ is the service rate of one server. All servers within the DC are assumed to be homogeneous, so for x activated servers the overall service rate of the DC is $x\mu$.

For the analytical solutions of the models in this thesis, Markovian assumptions are made in order to be able to solve for the probabilities of steady system states. Arrivals are assumed to occur according to a Poisson distribution, where interarrival times follow a negative exponential distribution function with mean value of ${}^{1}/_{\lambda}$. Service times are also assumed to follow a negative exponential distribution function with mean ${}^{1}/_{\mu}$, so that the basic model of a DC could be assumed as a Markovian $M/M/n/n + s$ queuing system. All proposed models of data centers as continuous-time Markov Chains are solved using Kolmogorov Equations through a novel iterative-recursive algorithm to obtain the steady state probability for each of the system states. Solutions are followed by calculations of the most significant performance metrics that describe system's behavior as average number of requests in the data center, average delays, loss probability, etc.

The use of queuing systems for modeling data centers and understanding their behavior has been studied by several authors in literature. Similar models of data centers as a two- dimensional Markov chain have been introduced in [115], [192], [113], [16], [17], and [13]. However the novelty of the algorithms proposed in this context compared to all previously mentioned algorithms is that our algorithms provide much simpler and exact solutions for solving steady state probabilities of the model via iterative recursive algorithms with consideration to all aspects describing system's behavior.

4.2 Simulation Technique

In order to verify the models solved using approximate assumptions or when mathematical solutions are too complex, a simulation tool is necessary. In this thesis all proposed models are verified by implementation of simulation models using OMNeT++ IDE tool (Objective Modular Network Integrated Development Environment) [131] and tested as well under different conditions that cannot be analyzed by theoretical analysis. For example, solving steady state probabilities of a queuing system requires Markovian assumptions to be made, which restricts interarrival and service time distributions to only one type. However as interarrival and service times are not always negative exponentially distributed, a simulation is required in order to check the model's performance under different non-Markovian realistic assumptions.

OMNeT++ is a discrete event open source simulator that has been used in a rising number of publications since its public release in 1997. It is based on C++ programming language and offers a simulation framework for several domains such as queuing networks, distributed systems, wired and wireless communication networks, sensor networks, storage area networks, etc... It provides a simple graphical user interface as well as built-in modules for constructing any desired network to be simulated rather than building a simulation model from scratch, which allows modules to be reusable and facilitates building large scale simulations for various areas. OMNeT modules are either simple modules which are dynamic modules written in C++ or a group of simple modules grouped together forming one compound module with various functionalities. Modules constructing a model communicate with each other via passing of messages through gates, where messages are forwarded to their destination module through output gates and received via input gates. Gates are interconnected according to the model's architecture via links that can be unidirectional or bidirectional. Modules also have parameters which provide configuration data such as random numbers following a defined probability distribution for defining module parameters, e.g.: server's service time. A typical OMNeT++ model is composed of the following files:

1. NED File
   Network Description File describes the structure of the implemented model to be simulated. It contains simple modules declarations as well as their gates, parameters and configurations, compound modules declarations which describe its gates and parameters and which simple modules that constitute it, and finally network definitions and interconnections between modules' gates. NED files come with a graphical editor, where users can either edit the NED file in source mode or perform modifications within the graphical user interface.
2. C++ Files
   Where each simple module needs a C++ class that dictates its functions and how it performs. Although every single module within the simulated model has its own C++ file, compound modules don't require any.
3. Initialization File
   Initialization File (.INI) contains all configurations and parameters required by the model simulation to execute. It specifies values and distributions for different variables of all modules in the model, such as distribution of interarrival times and their mean value, servers' service/activation times and their mean values as well as simulation duration or stopping condition.

Among the main advantages provided by OMNeT [25] are the ease of customization of all available modules to match the desired simulated network, as well as the ability to embed the simulation into bigger applications with data applications for accepting inputs and extracting outputs in files of common types. The modules that it already embeds also allow for simpler implementations as a user can easily build upon them and modify them as required. This ease of use and advantages have motivated the use of OMNeT in several research areas and publications. Authors of [22], [128] and [200] used OMNeT for simulating wireless and mobile networks, others used it for sensor networks [83] [102], optical networks [196][96] and related ones in the area of high performance computing [147] [43] and cloud computing simulations [114][26][27].

To simulate the basic data center explained in the previous section and illustrated in Figure 4-1, a few built in components from the queuing library of OMNeT++ are used. As shown in Figure 4-2 the cloud DC can be modeled as a source module that generates requests and forwards them into a FIFO queue, then these requests are served by any of the available servers which send a served arrival into a sink module to be destroyed. Briefly, the simple modules used perform the following functions:

1. Source Module

   The Source module generates messages to be forwarded into the passive queue according to a distribution defined in the initialization file.
2. Passive Queue

   A Passive Queue acts as the buffer for the data center which stores arrivals in a FIFO order, where in this simulation the queue is defined to have a specific finite capacity such that any arrival after this capacity is reached will be dropped. It has multiple output gates for forwarding of arrivals to servers at which they will be served. The passive queue acts as the controller deciding where arrivals should be forwarded, which could be decided according to several strategies such as priority selection or round robin. In this simulation round robin is used where any random empty server is selected to start serving an arrival when the queue controller decides upon its start service time.
3. Server

   The Server module performs the function of a server in the cloud data center that receives an arrival, starts processing it by creating the requested VM and running any necessary applications. When an arriving request finishes service it releases its resources for use by other queued arrivals. Service time of the simple server module is defined in the initialization file according to any desired distribution and mean value, also for some simulations, as will be later explained, servers are configured to have a random activation time for more realistic assumptions.
4. Merge

   A Merge module was added to the simulation model of load balancing algorithms in order to collect all arrivals of each DC and forward them to the sink module. It is necessary as it dramatically reduces the number of input gates at the sink module to only one per each DC instead of having one input gate per server per DC.
5. Sink

   Sink module is the final destination for all serviced arrivals, and it represents the departure of a request from the data center.

For each model among those presented in later chapters a compound module is implemented with the respective functionalities, specifications and variables. Several experiments are performed for each model in order to predict its performance and explore the effect of each parameter on its behavior. For accuracy of results, all measurements are repeated several times with the same parameters and different seeds used for generating random numbers in order to obtain multiple readings and deliver an average result with a defined 95% confidence interval.

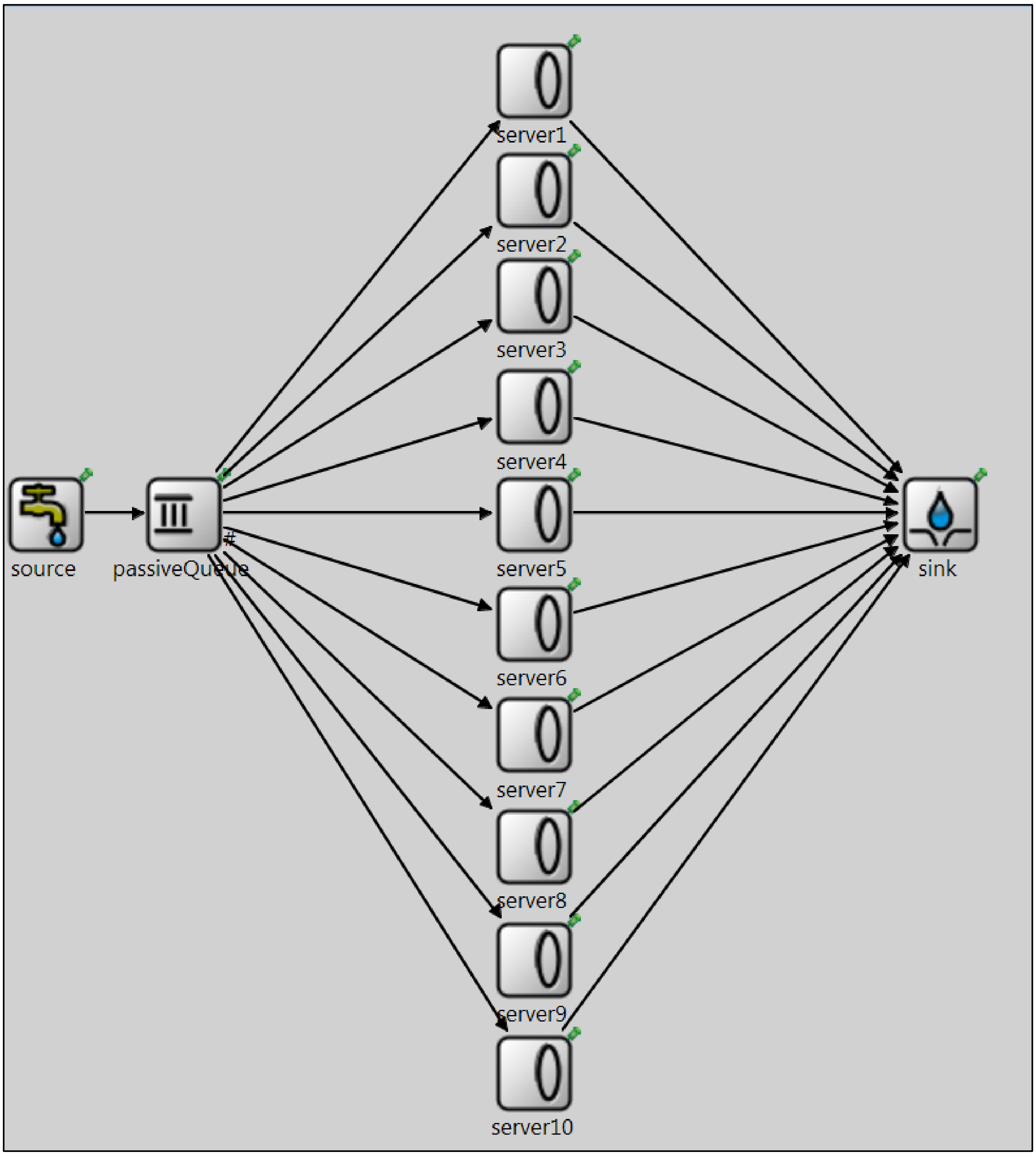


Figure 4-2: Basic DC Simulation Model

4.3 Experiment Setup and Measurements

For further proofing of the obtained results from analytical solutions and simulations of the proposed models, a test-bed is implemented for a sample data center consisting of two servers in order to test and run algorithms using VMware tools [190] to identify how they perform in real-life scenarios. This verification is of high importance as despite assuming realistic assumptions for most system aspects through simulations, not all effects encountered by a real-life running system can be accounted for. Effects such as how system scales with increasing its size, types of cabling or connections among its entities and the protocols they run, efficiency and response times of a centralized controller, etc. are all aspects that are best to draw conclusions based upon real-life experimentation.

The test bed used is constructed using three ESXi servers with one data store connected within one LAN via ISCSI, as illustrated in Figure 4-3. One of the three servers is configured with vCenter [186] to act as a centralized controller for the data center providing proactive management over the other servers and their hosted virtual machines. The data store holds all virtual machine files and was supplied with operating system images to be deployed upon them. Test bed setup was performed by the aid of previous research work done by authors of [153], [130] as well as several virtualization software listed below:

1. ESXi Hypervisor [60]
   ESXi is a bare-metal hypervisor that installs directly on the server and performs the function of virtualizing the server for hosting several logical virtual machines.
2. VMware vCenter [186]
   vCenter acts as a platform for managing all ESXi servers, data stores and virtual machines within the data center. It was installed on one of the servers in order to remotely manage other servers for performing functionalities such as deploying, shutting down or migrating a virtual machine or monitoring the performance of servers and each of their hosted virtual machines.
3. VMware vSphere [188]
   vSphere is the software package through which ESXi is deployed on servers and vCenter is installed for centralized management of the data center. It acts as a user interface to ease the configuration process by the end user via connecting it to vCenter, where this can be performed through installing vSphere client on any windows machine. An important add-on of vSphere is vMotion which is used for testing the proposed load balancing algorithm for migration of virtual machines between the two servers.

After the test bed was installed the proposed algorithms and their test cases were deployed using scripts that are automatically run by ESXi servers using PowerCLI software [187].

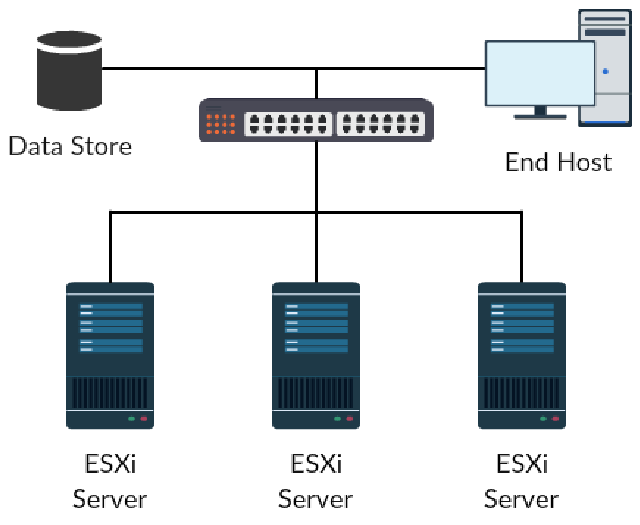


Figure 4-3: Test Bed Architecture

Chapter 5 Energy-Efficient Cloud Data Centers

This chapter introduces the proposed Multiple Parallel Hystereses model which allows for achieving an energy efficient operation of cloud data centers while maintaining users' service level agreements. The algorithm is based upon the idea of server consolidation where the number of active servers in the data center is adapted to its current load, so that under-utilized servers can be turned-off or put into sleep mode to save energy consumed while running idle. The novelty of this model is illustrated in its ability to model the data center with consideration to realistic aspects such as server activation overhead and reduced service rates during sleep modes, performing automatic sever consolidation and accurately predicting the performance of the data center with upper bounds for users' delay using an iterative-recursive algorithm for solving system state probabilities at any given load situation.

Although the idea of modeling DCs using queuing systems with hystereses thresholds for servers' activation/deactivation have been previously introduced in literature, the proposed algorithm introduced in this chapter for solving queuing systems with hystereses is the simplest among all previous work as it is the first to discover the recursive nature of steady state equations used for solving state probabilities. Keilson et al. [129] were the first to introduce the idea of setting activation and deactivation hystereses thresholds for servers based on the number of requests in the system. Authors provided exact analysis for the multi-server threshold queues with hysteresis using Greens methods developed by the same author in [86]. Due to complexity of the analysis method, authors were not able to extend their solutions beyond systems with 2 heterogeneous servers. The same queuing model was analyzed by authors of [79] for instantaneous server activations and further extended in [41] to include server activation overhead and bulk arrivals; where their analysis was based on the method of stochastic complementation for solving steady state probabilities using a closed-form solution. Complexity of stochastic complementation method has also limited their analysis to systems with limited number of heterogeneous servers and limited bulk size. Explicit mathematical analysis by methods of Green Function [129] or by Stochastic Complement analysis [79] lead to rather complex equations which are difficult to evaluate and have been applied to very small systems. Novelty of the recursive algorithm proposed in this thesis is shown in its ability to easily compute steady-state probabilities of arbitrarily large heterogeneous systems without any stability problems.

This chapter starts with an explanation for the parallel hysteresis model and how hysteresis allows for implementing server consolidation and adaptation to the DC's current load. The second section explains the recursive algorithm used for solving the model along with mathematical analysis for its most important performance metrics. Section two also introduces the OMNeT++ simulation model for a data center operating under the parallel hysteresis algorithm as well as the architecture and implementation of the model on the cloud DC test-bed. Results obtained from the recursive algorithm are presented in Section 3 and verified in comparison to those obtained by OMNeT++ simulator model and testing the model on a data center test bed using VMware.

5.1 Model Explanation

The Multiple Parallel Hystereses Model uses the hysteresis behavior previously introduced by the thesis' author and main supervisor in [119] and [133] to provide the DC with load-dependent behavior. It throttles activation/deactivation of servers by defining upper and lower thresholds for activation and deactivation of each server; respectively, to avoid frequent oscillations between on/off states in servers, and thus automatically adapting the number of active servers in the DC to its current load. Upper and lower thresholds constitute a hysteresis where the upper threshold for server activation is defined by the number of buffered arrivals $w^{(x)}$ for each number of active servers x, so that when the system is running with x active servers it is required to hold $w^{(x)}$ arrivals in its buffer to make sure that load has increased to the limit that a new server is required to be activated for $x = 1,2,\dots,n$. These hystereses thresholds force the DC to activate only the amount of servers required to serve the current load, while keeping the rest of the servers inactive or in sleeping mode, thus automatically consolidating requests into minimal acceptable number of servers. Multiple thresholds are specified – one per server - to avoid frequent oscillations between activations and deactivations of servers, where hystereses thresholds allow server activations to be throttled upon short load bursts by buffering arrivals until reaching a certain buffering threshold indicating that the current load has increased to a limit that a new server activation is required. When a new arrival occurs at this threshold level, one new server activation is triggered. Values for hysteresis thresholds are set with consideration to maximum delays that could be tolerated by the arriving request, which could be derived from arrival's Service Level Agreement (SLA). Another characteristic of the model to guarantee minimal delays is that deactivation of servers do not occur except when a server has no more queued arrivals to serve, which ensures that arrivals are served at the maximum service rate of activated servers.

The model considers two different server activation delays which model the durations required by a switched-off server to change its state into switched-on or by a server in sleep mode to change its state into active. It also models DVFS strategies where servers work at a reduced service rate by reducing their operational frequencies for efficient energy consumption. These two aspects will be explored separately by case studies at the end of this chapter. The state transition diagram explaining the system behavior is shown in Figure 5-1, where the two types of states (shaded/Un-shaded) represent the system states with/without activation overheads, respectively. Un-shaded states represent a model with instantaneous activations of servers, where a new arriving request occurs at the border states $(0,0), (x, w^{(x)}), \dots$ for $x = 1,2,\dots,n-1$, where $w_0 = w^{(0)} = 0, w^{(1)} = w_1 + w_0$, etc. would lead to an immediate server activation by a horizontal transition arrow into state$(1,0), (x+1, w^{(x)}), \dots$ for $x = 1,2,\dots,n-1$, respectively. These horizontal transitions are not shown in the model, but rather the shaded states are introduced to indicate that the activations of a server is not an instantaneous process, but takes a period of time that is assumed in this model to be negative exponentially distributed with an average of 1/α.

As shown in Figure 5-1, the system starts at the bottom left idle state, represented as $(0,0)$ where the system has null active servers and buffered arrivals. At this idle state when an arrival occurs at rate λ it triggers a server to be activated, but since activation times are not negligible, the arrival will be buffered until the triggered server is activated with an average activation time of $1/\alpha$, thus the state of the system changes to $(0,1)$. At this instant two events may occur: either another new arrival happens before the server gets activated, or the server activates. If a new arrival occurs first it gets buffered until the server gets activated and the system's state changes to $(0,2)$ where further arrivals will be buffered until a hysteresis threshold is reached to trigger activation of another server, or in case the triggered server becomes active then the x component of the state increases by one indicating an active server and the z component reduces by one as one arrival starts its service time. If the server activates first, then it will start servicing the arrival in the buffer leaving the buffer empty, and the state of the system changes to $(1,0)$. Another two events are possible at this state: either a new arrival occurs and the system changes to state $(1,1)$ by buffering this arrival until the server is finished with the arrival it is currently serving, or if the server finishes first then the system turns into an idle state $(0,0)$, and so on….

In Figure 5-1 the events of triggering a server to be activated are marked as 'A' with bold arrows. Servers are only triggered for activation when an arrival occurs at defined system states: $(0,0), \left(1, w^{(1)}\right), \left(2, w^{(2)}\right), \left(x, w^{(x)}\right), \ldots$ for $x = 0,1,2, \ldots, n-1$ and $w^{(x)} = w_1 + w_2 + \cdots + w_x$where $w^{(x)}$ indicate the queue thresholds for triggering activation of a new server x, while w_x indicates the increase in threshold for new arrivals to be buffered before activating a new server x relative to threshold of server $x+1$, respectively. Boundary values for w_x are $w_0 = 0$ and $w_n = s$. Server deactivations are indicated in the figure also by bold arrows and marked as 'D', where deactivation of a server is only allowed when a server becomes idle, and there are no more arrivals to be served in the queue. Server deactivations are only triggered when an arrival finishes services at any of these system states: $(1,0), (2,0), \ldots, (n, 0)$.

Generally, when the system is at any general state (x, z), expected events are either an arrival event, departure event, or server activation event. The system state changes according to each of these events as follows:

1. If an arrival event occurs:
   The current system state has to be checked:
   - If the arrival occurs at any of the defined border states at which the arrival exceeds a defined queuing threshold and a new server must be activated, i.e. states $(0,0), \left(1, w^{(1)}\right), \ldots, (n-1, w^{(n-1)})$, then a server activation is triggered.
   - If the arrival occurs at any state where the buffer is full, i.e. $(0, s), (1, s), \ldots. (n, s)$, the arrival will be lost since the queue is fully occupied.
   - If the arrival occurs at any other state other than the states mentioned in the above two conditions, the arrival will be queued.

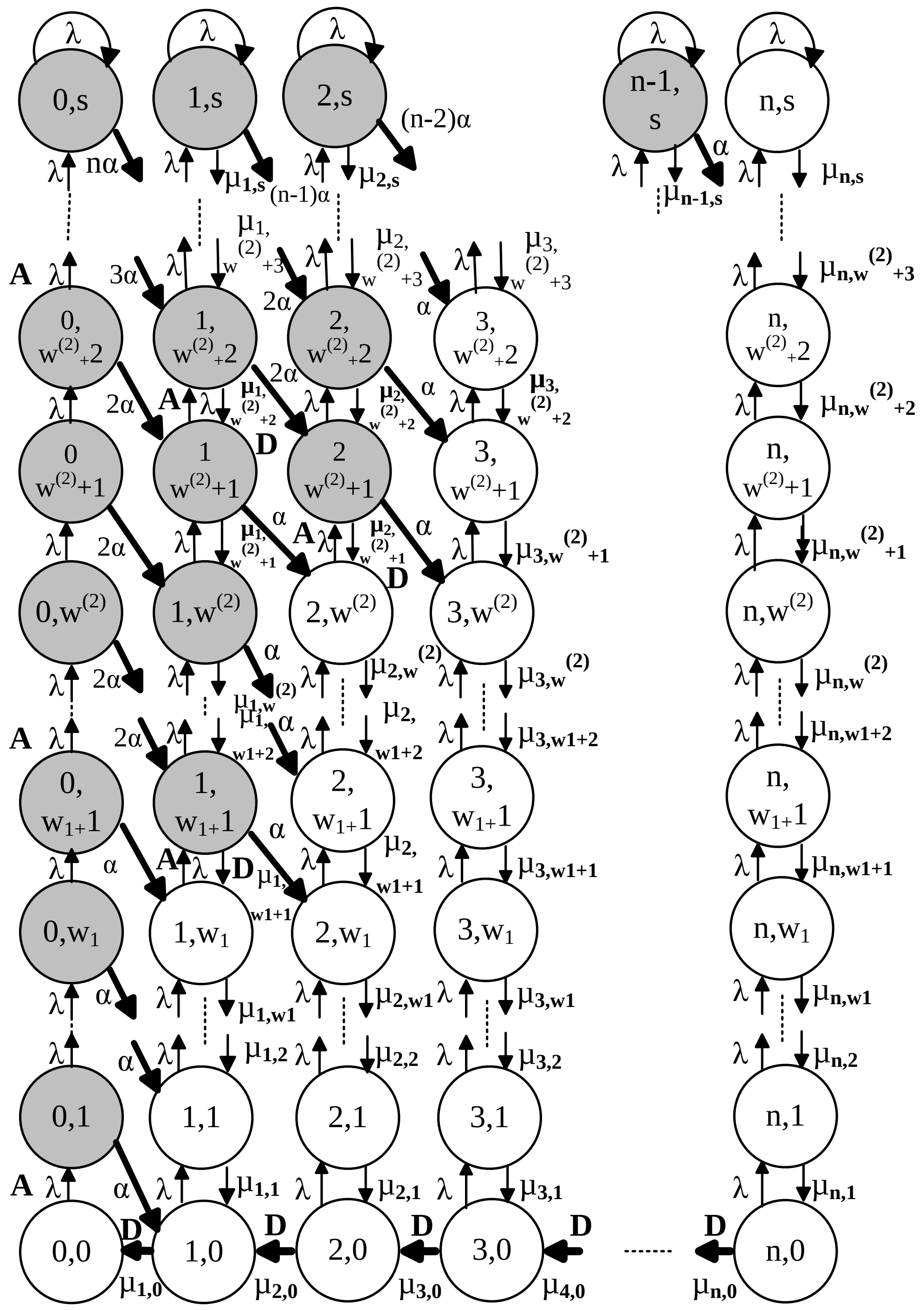


Figure 5-1: State Transition Diagram of Multiple Parallel Hystereses Model with Activation Overhead

2. If a departure event occurs:
   Departure event occurs after an arrival has completed its service time at one of the systems' servers, and the system state has to be checked:
   - ➢ If the departure occurs and the buffer is still occupied with arrivals, the server will be re-occupied with the arrival at the head of the queue according to FIFO policy.

- Otherwise if the departure event occurs and the queue is empty, then the server at which departure occurred will be deactivated.

3. If a server is activated:
   If a server had been triggered for activation and its activation time has ended, then this server will accept the arrival at the head of the queue, thus reducing the queue size by one and increasing the number of active servers by one.

In Figure 5-1, service rates at each state are indicated as $\mu_{x,z}$ to represent the service rate at the respective state (x, z). In case of homogeneous service rates, $\mu_{x,z}$ is equivalent to $x\mu$, which is the assumption in the analytical solution. However, the model is adaptive to different service rates at each server, and also to different service rates of the same server at different stages, i.e. active, sleeping, switched-off, etc... Activation rates also may exceed α (activation rate of one server) in case a new activation is triggered before the previously triggered server is activated. This adds more α to the activation rate, and it is allowed to increase only up to $n\alpha$ when all the servers in the system are triggered for activation, but none is active yet.

Hystereses threshold values for the model are derived such that an arrival's SLA is not violated in terms of average response time. For each number x of active servers where $x = 1,2, \dots n$, a threshold $w^{(x)}$ is specified such that the worst case mean delay with the current service load would still be tolerated under the specified SLA. This worst case mean delay could be explained as follows: for an arrival that meets the system at state $(x, w^{(x)} - 1)$, this arrival will change system state to $(x, w^{(x)})$ and will experience the longest mean delay. Compared to arrivals before and after this arrival, the ones arriving before it experienced a shorter queue and thus a shorter mean delay, and the arrival after it will exceed the threshold value and trigger a new server activation which will increase the service rate so it will also experience a shorter mean delay. Worst case mean delay of this arrival can be calculated knowing that there are $w^{(x)} - 1$ customers ahead of it in the queue, added to them one already being serviced in the server at which the arrival will be serviced. These $w^{(x)}$ arrivals are being serviced at a rate of $x\mu$ (in case of homogeneous servers), so the worst mean delay time could be calculated by the following equation:

$$t_{wo} = \frac{w^{(x)}}{x\mu} \tag{5-1}$$

Accordingly, since the worst case mean delay t_{wo} must be smaller than or equal to the delay bound specified by the SLA, by setting t_{wo} to the SLA threshold value the hysteresis threshold could be obtained using Equation 5-2 and rounding down to the nearest integer value.

$$w^{(x)} \leq t_{wo} * x\mu \tag{5-2}$$

Another approach for setting hysteresis thresholds according to SLA is by not violating percentiles of the response time distribution function. Percentiles defined by SLAs require that for a defined percentage of time, the delay should not exceed a certain maximum delay.

In this case a different calculation approach should be adopted, which is not considered in the scope of this thesis. The model of Figure 5-1 describes all cases where arriving jobs are served without any I/O interruption (as in case of paged computer systems). The model can also be applied to paged computer systems; in that case the arrival rate λ includes all returns after a page fault interruption.

5.2 Model Analysis

Analysis for the multiple parallel hystereses model explained above is performed using two methods as will be explained in this section. First the model is analyzed mathematically by solving the Markov chain shown in Figure 5-1 using a novel iterative recursive algorithm for solving state probabilities of the system by which performance metrics could be calculated to evaluate the model under Markovian assumptions. In the second subsection a simulation model is constructed for a DC operating under multiple hystereses model in order to verify analytical results as well as testing the model's performance under non-Markovian assumptions to verify its validity under general conditions.

5.2.1 Mathematical Analysis

Analysis and calculation of state probabilities for the Markov chain of the Multiple Hystereses model with activation overhead is performed using standard Kolmogorov Forward Equations in a recursive manner. A recursive solution is adapted here to avoid numerical instabilities that occur when using explicit formulas and numerical solutions. The model is solved under the Markovian assumptions that all inter-arrivals and service times follow a negative exponential distribution, as well as servers' activation times. At any state (x, z) in the Markov Chain; the rate of arrivals is λ, total service rate of the system is $\mu_{x,z} = \sum_{i=1}^{x} \mu_i$, and activation rate is $\alpha_{x,z}$ which is the summation of activation rates for all servers triggered for activation at the respective state.

The algorithm for calculating state probabilities $p(x, z)$ for $x = 0,1, \dots n$ and $z = 0,1, \dots s$ follows the following steps:

- Assume $p(0,0) = 1$
- At all states $(0, j)$, where $j = 1,2, \dots s - 1$, Kolmogorov forward equation follows from Equation 5-3

$$\left(\lambda + \alpha_{0,j}\right) p(0, j) = \lambda\, p(0, j - 1) \tag{5-3}$$

- Using equation 5-3, states $(0, j)$ where $j = 1,2, \dots s - 1$ can be calculated relative to $p(0,0)$
using Equation 5-4

$$p(0, j) = \frac{\lambda}{\lambda + \alpha_{0,j}}\, p(0, j - 1) \tag{5-4}$$

- Probability of state $(0, s)$ can be calculated according to Equation 5-5

$$p(0, s) = \frac{\lambda}{\alpha_{0,s}} \, p(0, s-1)$$

(5-5)

- Using the balance equation at state $(0,0)$ expressed in Equation 5-6, $p(1,0)$ can be calculated exactly relative to the assumed $p(0,0)$

$$p(1,0) = \frac{\lambda}{\mu_{1,0}} \, p(0,0)$$

(5-6)

- Assume $p(1, s) = y$
- Using balance equation at state $(1, s)$, probability of state $(1, s-1)$ can be calculated according to Equation 5-7

$$p(1, s-1) = \frac{\mu_{1,s}+\alpha_{1,s}}{\lambda} \, p(1, s)$$

(5-7)

- Probabilities of states $(1, j)$ for $j = s-2, s-3, \dots 1$ can be calculated recursively using Equation 5-8

$$p(1, j) = \frac{1}{\lambda}\left[(\lambda + \mu_{1,j+1} + \alpha_{1,j+1})p(1, j+1) - (\mu_{1,j+2})p(1, j+2) - (\alpha_{0,j-2})p(0, j - 2)\right]$$

(5-8)

- As Equation 5-8 can be used to solve state $(1,0)$ in terms of variable y, it can be equated to the numerical value of state $(1,0)$ obtained by Equation 5-6 to solve for y
- Substitute the obtained value of y in states $p(1, j)$ for $j = s-2, s-3, \dots 1$
- Remaining system states can be solved in the same manner recursively according to the following general equations for each i, where $i = 2,3, \dots n$

i. Assume $p(i, s) = y$ (5-9)

ii. $p(i, s-1) = \frac{\mu_{i,s}+\alpha_{i,s}}{\lambda} \, p(i, s)$ (5-10)

iii. For all states (i, j) where $j = s-2, s-3, \dots 0$

$$p(i, j) = \frac{1}{\lambda}\left[(\lambda + \mu_{i,j+1} + \alpha_{i,j+1})p(i, j+1) - (\mu_{i,j+2})p(i, j+2) - (\alpha_{i-1,j-2})p(i-1, j - 2)\right]$$

(5-11)

iv. Probability of state $(i, 0)$ can be found numerically using balance equation at state $(i-1,0)$, equate numerical value to the value obtained from Equation 5-11 to evaluate y

v. Substitute the value of y into states $p(i, j)$ for $j = s-2, s-3, \dots 1$

- After obtaining all probabilities of state, calculate normalization factor

$$\text{Normalization Factor}=1/\sum_{all\ i}\sum_{all\ j}p(i,j) \tag{5-12}$$

- Multiply all probabilities of state by the normalization factor

Having obtained all steady state probabilities of the DC model under Multiple Parallel Hystereses algorithm, performance metrics are necessary to evaluate and study the system performance. Most significant performance metrics of the DC are derived using equations below:

- State distribution of busy servers

$$P(x)=\sum_{z=0}^{s}p(x,z) \tag{5-13}$$

where the probability of x active servers can be obtained by adding up all probabilities of states where x servers are active.

- Average number of busy servers, also indicating the amount of carried traffic

$$Y_s=\sum_{x=1}^{n}xP(x) \tag{5-14}$$

carried traffic by the average number of servers is calculated by summing the multiplication of the probability of each states by the number of active servers in this state.

- State distribution of buffered arrivals

$$Q(z)=\sum_{x=0}^{n}p(x,z) \tag{5-15}$$

- Mean queue length, i.e. average number of buffered arrivals

$$L=\sum_{z=0}^{s}zQ(z) \tag{5-16}$$

- Probability of an arrival to be lost (Blocking Probability)

$$B=\sum_{x=0}^{n}p(x,s) \tag{5-17}$$

since arrivals are lost when the system's buffer reaches its capacity, loss probability of the DC follows from the summation of all states where $z = s$.

- Probability of an arrival to be delayed upon arrival

$$W = 1 - B \tag{5-18}$$

Due to activation delay of servers, all arrivals to the system experience delays except those who are lost.

- Mean waiting time with respect to all arriving requests following from Little's law [80]

$$E[T_w] = L/\lambda \tag{5-19}$$

- Mean waiting time with respect to buffered requests (excluding those arrivals that are lost)

$$E[T_w | T_w > 0] = L/\lambda W \tag{5-20}$$

- Activation rate of idle servers

$$R_A = \lambda * \sum_{x=0}^{n-1} \sum_{i=x}^{n-1} p(x, w^{(i)} + i - x) \tag{5-21}$$

where Equation 5-21 sums up all states at which a new server activation is triggered due to exceeding buffering threshold by a new arrival request, and multiplies them by λ as triggering a new server activation occurs as a result of an arrival event.

- Deactivation rate of active servers

$$R_D = \sum_{x=1}^{n} p(x, 0) * \mu_{x,0} \tag{5-22}$$

since deactivation of a server occurs only when a server turns idle, i.e., after an arrival is served and the queue is empty, Equation 5-22 adds up all states where the queue size is null, and each state is multiplied by its respective service rate.

- Average number of servers in activation phase

$$Y_A = \sum_{x=0}^{n-1} \sum_{z=0}^{s} \frac{\alpha_{x,z}}{\alpha} * p(x, z) \tag{5-23}$$

where Equation 5-23 calculates the average number of servers in activation phase by multiplying the probability of each state in the system by the number of servers triggered for activation at this state represented as $\alpha_{x,z} = i\alpha$, where $i = 0,1,\dots n$. Another more general calculation is shown in Equation 5-24:

$$Y_A = \sum_{x=0}^{n-1}\sum_{i=x}^{n-1}\sum_{j=1}^{w^{(i+1)+i-x+1}} (i - x + 1) * p(x, w^{(i)} + i - x + j)$$

(5-24)

- Power consumption of servers inside the DC

$$P_s = Y_s P_{running} + Y_A P_{running}$$

(5-25)

In the simplest form of this model servers that are not yet triggered for activation remain switched-off, thus eliminating the power consumed by running idle and limiting the power consumption to servers running at their full speed or being activated. Power consumed by running servers can be computed by multiplying the average number of active servers by the power consumed by a running server $P_{running}$, which is estimated to be equal to 25W as reported in [5] for an Intel Pentium M 1.6 GHz Processor. Power consumed by servers in the activation phase can be calculated by multiplying the average number of servers in activation phase times the power spike caused by activation, which is assumed to be equivalent to that consumed by a server running at full speed.

- Power-Saving Efficiency

$$\eta = (n * P_{running} - P_s)/(n * P_{running})$$

(5-26)

in this context power saving efficiency is measured as the amount of power saved by applying the hystereses algorithm to data centers instead of operating them by the always-on strategy. This definition implies that the highest efficiency can be achieved at low load situations where power consumed by idle servers can be saved, whereas at high load situations when the DC is fully utilized the space for power saving efficiency is minimal.

5.2.2 **Simulation Model**

For testing the Multiple Hystereses Model for achieving energy efficiency introduced in this chapter, a compound OMNeT++ module is implemented as shown in figure 5.2. As earlier explained in Section 4.2 the source module generates arrivals with a random interarrival time following a negative-exponential distribution function, where the average time between arrivals depends on the load experienced by the DC. Arrivals are forwarded from the source to the passiveQueue module which decides whether the arrival should be immediately served thus triggering the activation of a new server for serving it, or will be queued until the next activation threshold is reached. In case an activation threshold is reached which is determined by the number of buffered arrivals, the criterion of choosing which server to be activated is done using a round-robin approach, and the time required by a server to be activated is randomly selected according to a negative-exponential distribution. When a server is activated a message is passed from the server to the queue requesting to send the arrival buffered at the head of the queue, where service times of arrivals at servers are negative-exponentially distributed with an average of one unit time for each server in the DC. When the service time of an arrival ends, it is forwarded to the sink module where it will be discarded from the DC, and the server at which it had been served at will pass a message to the queue asking for the arrival at the head of the queue to start its service time. In case of an empty buffer the server will be deactivated or put into a sleep mode.

The algorithm is tested under several parameters specified in the initialization file of the model (.ini), where the mean value of interarrival, service and activation times are specified for generating exponential random numbers with respective mean values. The duration of the simulation is also specified, where it is set to end after the millionth arrival leaves the DC. To test the performance of a DC operating under the proposed algorithm and under diverse load conditions, the simulation runs under load value per server ranging from 0.1 to 1, where a load of 0.1 per each server indicates long interarrival times in comparison to service times, and a load of 1 indicates high load at servers as arrival rate is equal to the service rate. Throughout this simulation the average service time for each server in the DC is assumed to be 1, so for simulating the DC under different loads the interarrival times are varied according to equation 5-27:

$$\rho = \frac{A}{n} = \frac{\lambda}{n\mu} = \frac{average\ service\ time}{n{*}average\ interarrival\ time}$$

$$average\ interarrival\ time = \frac{average\ service\ time}{n\rho} \tag{5-27}$$

Figure 5-2 illustrates the (.ned) file for the simulation, which shows all simple modules and their interconnections. The procedure followed by the simulation is explained through flow charts in Figure 5-3 which explain the span of arrivals and servers inside the data center, respectively.

Most important performance metrics for evaluating DC's performance under Multiple Hystereses Model are calculated using the following equations:

- State distribution of busy servers

$$P(x) = \frac{Number\ of\ arrival\ instances\ \ where\ x\ servers\ were\ active}{Number\ of\ arrivals} \tag{5-28}$$

Where at the instance of each arrival the number of active server is recorded, so that the probability of x active servers can be measured by counting the times x servers were on at arrival instants divided by the total number of arrivals.

- Mean delay of delayed arrivals

$$E[T_w|T_w > 0] = \frac{\sum Delays\ of\ delayed\ arrivals}{Number\ of\ delayed\ arrivals}$$

(5-29)

The number of delayed arrivals is determined by a counter that is incremented every time a job enters the queue, which represents all arrivals that are accepted in the DC and not lost. All accepted arrivals are considered to be delayed due to activation delay of servers. Delay of each delayed arrival is measured as the difference between its arrival time and the time it started service.

- Power saving efficiency

$$\eta = \frac{\Phi - \sum_{all\ servers} Active\ time\ of\ server * Power\ consumption\ per\ unit\ time}{\Phi}$$

$$\Phi = Number\ of\ servers * Total\ simulation\ time * Power\ consumption\ per\ unit\ time$$

(5-30)

where the activity period of each server in the DC is measured and multiplied by the amount of power consumption per unit time, then added together to calculate the total power consumption of the DC. Subtracting the amount of consumed power using hystereses model from the amount of consumed power by servers operating under the always on strategy and dividing by it results in the power saving efficiency factor.

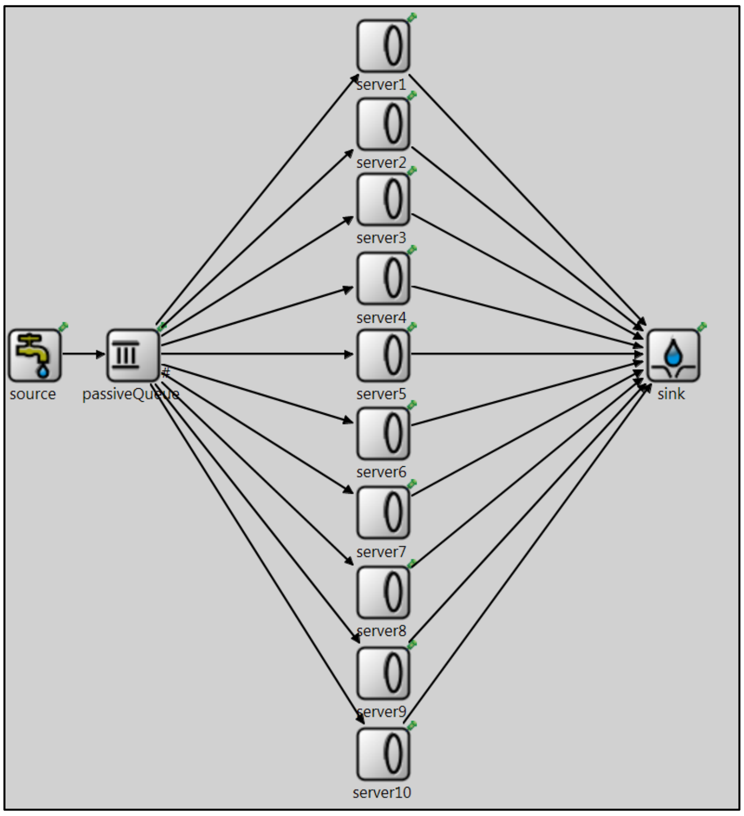


Figure 5-2: Basic DC Simulation Model

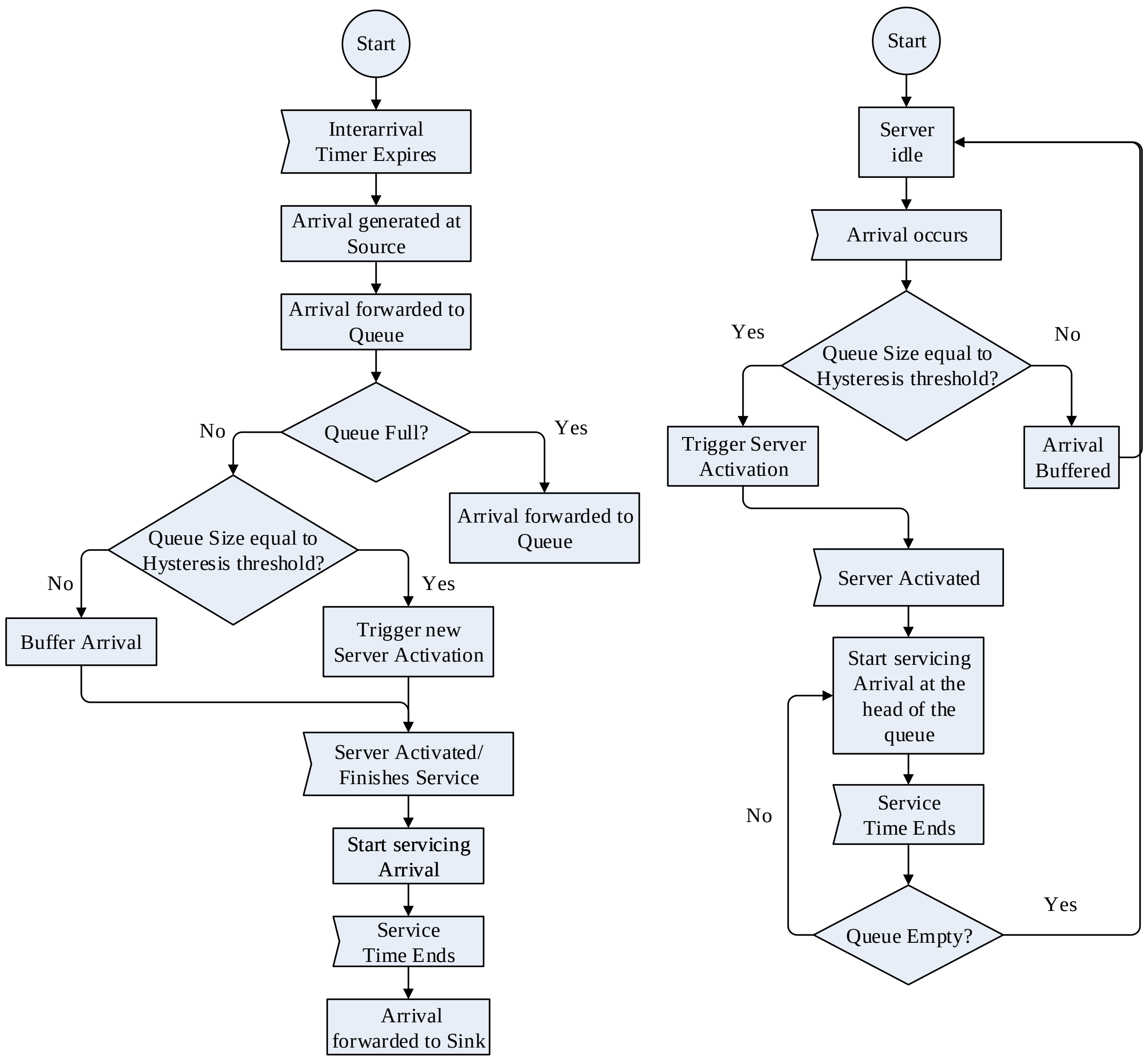

Start
Interarrival Timer Expires
Arrival generated at Source
Arrival forwarded to Queue
Queue Full?
No
Yes
Arrival forwarded to Queue
Queue Size equal to Hysteresis threshold?
No
Yes
Buffer Arrival
Trigger new Server Activation
Server Activated/ Finishes Service
Start servicing Arrival
Service Time Ends
Arrival forwarded to Sink
Start
Server idle
Arrival occurs
Queue Size equal to Hysteresis threshold?
Yes
No
Trigger Server Activation
Arrival Buffered
Server Activated
Start servicing Arrival at the head of the queue
Service Time Ends
Queue Empty?
No
Yes

Figure 5-3: Flow Charts explaining Life Cycle of Arrivals and Servers inside the DC

5.3 Results

5.3.1 Performance Evaluation

This section introduces the results obtained by testing the performance of a data center under the proposed Multiple Hystereses Model. Results in this section are obtained by solving the recursive solution introduced in Section 5.2.1 using MATLAB tool, from the implemented simulation using OMNeT++ software as well as from PowerShell scripts running on the DC test bed. The model is tested under various loads to see how its performance changes under light/heavy load conditions. All figures are shown for load value $\rho = \frac{A}{n} = \frac{\lambda}{n\mu} = 0, \dots, 1$ with variable arrival rate $\lambda = 0, \dots, 100$ arrivals per unit time and constant service rate per server μ fixed to 1 request per unit time for each of the 100 servers. Parameters such as hystereses width and activation rate are also varied to see how they affect system performance. An identical queuing system without the algorithm is also presented for comparison and to clarify the improvements achieved by implementing the Multiple Hystereses model on the DC's performance. For all results introduced in this section the activation delay of sleeping servers is assumed to be of a minimal value in order to study the effects of varying other parameters, where the effect of varying activation times will be separately studied in Section 5.3.2.

5.3.1.1 Probability of State P(x)

The main target for introducing the hysteresis behavior in this model is to adapt the number of servers in the DC to its current load, thus reducing the frequency of activations/deactivations of servers and being able to consolidate the load onto the lowest number of active servers to switch-off idle servers or put them into sleep mode. This is illustrated in Figure 5-4 which shows probability of state $P(x)$ for $x = 0, 50$ and 100 active servers for a date center with $n = 100$ servers, $s = 300$ buffer places and hysteresis width $w = 3$. A hysteresis width of 3 implies the following: after the first server is triggered for activation when the first request arrives, the second server is triggered for activation when an arrival occurs to a queue size of 3; the third server is triggered for activation when an arrival occurs to a queue size of 6, etc. The figure shows that $P(0)$, i.e. probability of having zero active servers is highest when the load approaches zero, and decreases rapidly to zero as load increases and more servers are turned on to accommodate the increased load. An exact opposite behavior occurs at $x = 100$, where the probability of having 100 active servers is zero for all small and intermediate load values, then increases to 1 as the system load approaches $\rho = 100$ when arrival rate and service rates have equivalent values. For intermediate load values as $x = 50$, 50 active servers are only needed when the system is half loaded, which is shown in the figure by a probability peak around $\rho = 50$ and zero probability at other smaller or higher loads. Figure 5.4 also compares the model results to the results of standard
M/M/n/n+s queue with the same parameters without server consolidation whose results are shown in dashed lines. Proposed model shows that it enhances the automatic adaptability of the number of active servers in the data center to the load value by concentrating the probability of $P(x)$ around respective load value $\rho = x$, for $x = 0,1,2, \dots . n$.

5.3.1.2 Server Activation Rates R_A

Another effect of increasing hystereses widths and increasing number of buffered arrivals before activating a new server is reducing server activation/deactivation rates. This effect is one of the main effects of the hystereses model by which automatic server consolidation is achieved. Figure 5-6 compares server activation rates of an $M/M/n/n + s$ queue vs. one with hystereses having identical parameters $n = 100$ and $s = 300$. The figure shows that servers' activation rates decreases rapidly as hysteresis width increases, which reduces the oscillation between on/off states in servers thus avoiding delays and power spikes experienced by activating switched-off/sleeping servers. This occurs because as more arrivals are buffered, new server activation will be delayed more until load increases to a defined level by hysteresis thresholds. Varying hysteresis thresholds results in a corresponding variation of the servers' activation rates as increasing the number of buffered arrivals delays the activation of servers and thus reduces the activation rate, as shown in Figure 5-7.

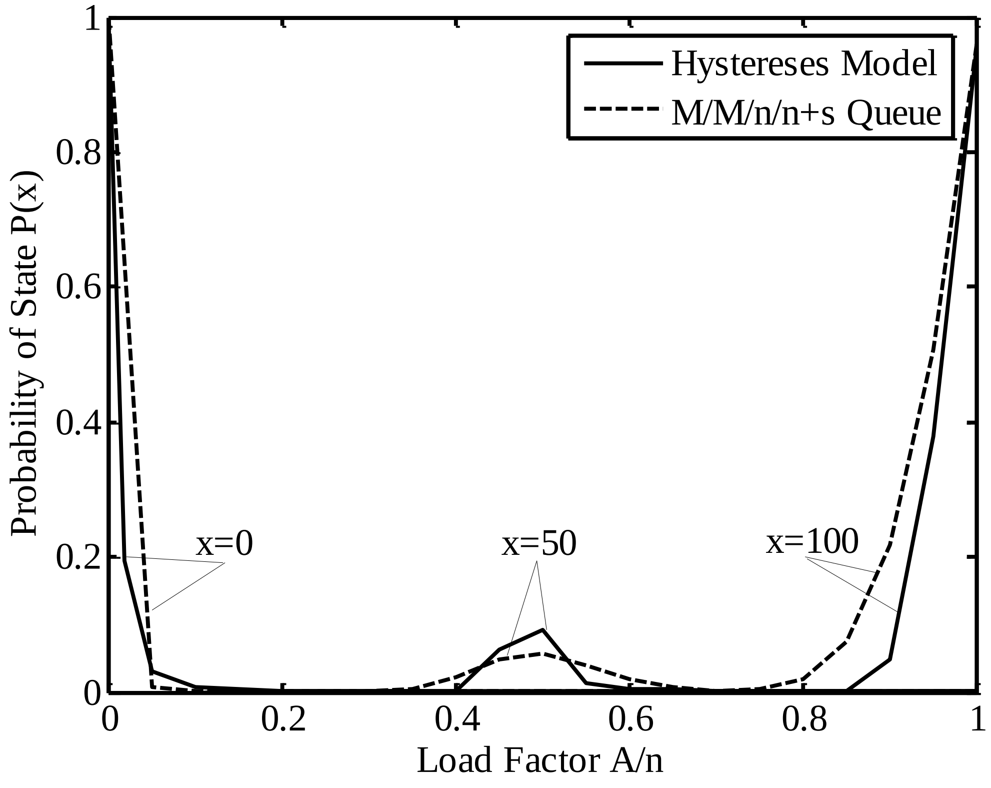


Figure 5-4: Probabilities of State P(x) of the Server Group inside Cloud Data Center

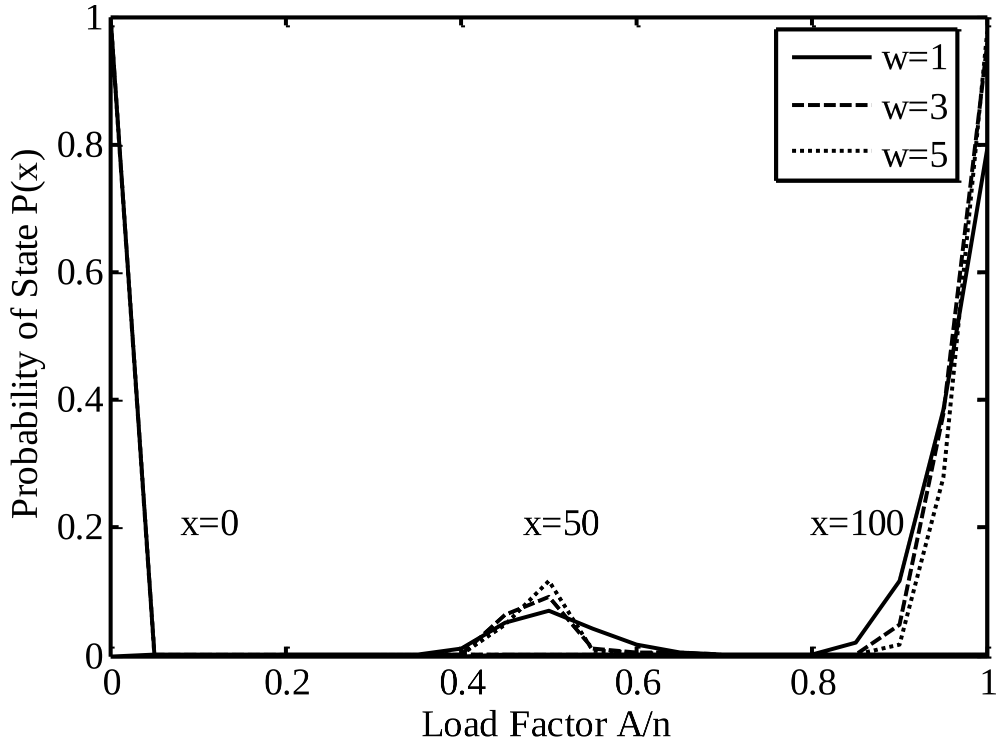


Figure 5-5: Probabilities of State P(x) of the Server Group for variable Hysteresis Widths

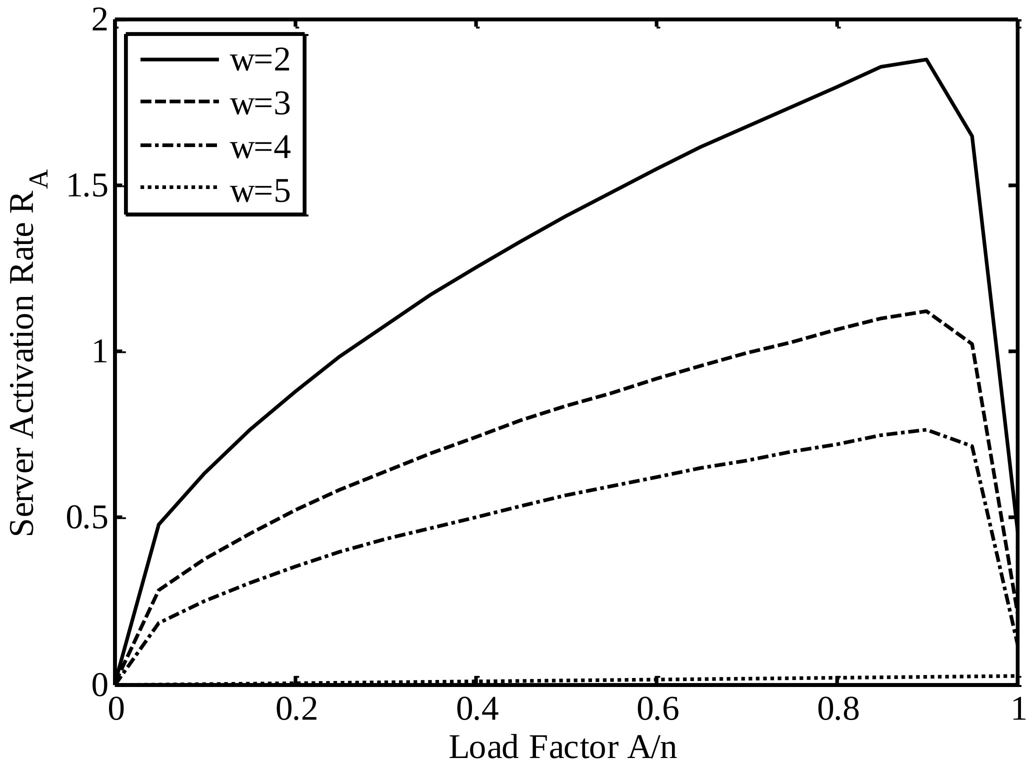


Figure 5-6: Server Activation Rate RA for variable Hystereses widths

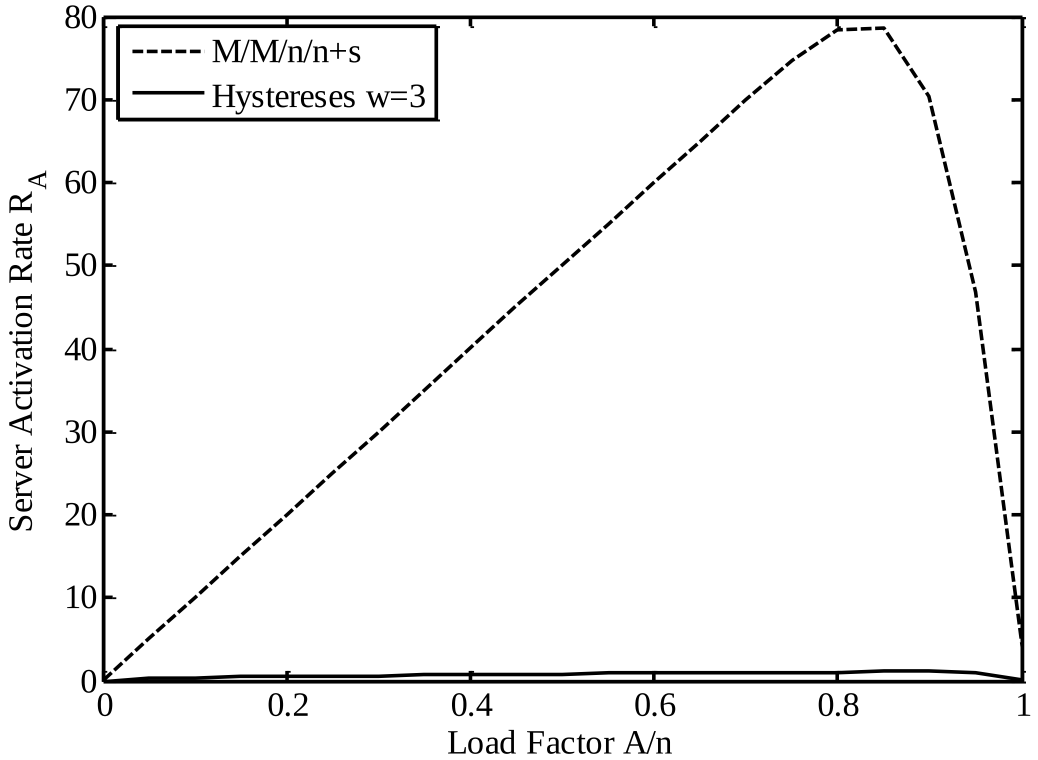


Figure 5-7 Server Activation Rate R_A for Hystereses Model vs. an equivalent $\boldsymbol{M/M/n/n+s}$ Queue

5.3.1.3 Power Consumed by Servers P_S

For evaluating the model's efficiency at reducing power consumption, Figure 5-8 shows the amount of consumed power for a DC operating under hystereses model with $n = 100$ and $w = 2,3,4$ and 5 versus an $M/M/n/n + s$ queue operating under always-on strategy. Power consumed at different server states is adopted from [5], where a server operating at full capacity consumes 25W as well as those being activated, and servers that are in-active are assumed to be switched-off with zero power consumption in the hystereses model, where in the always-on strategy they are idle with power consumption of 6W per unit time. Figure 5-8 shows a huge reduction in consumed power achieved by the hystereses model at low loads and decreasing as the load on the DC increases. At a load value approaching zero, the hystereses model keeps all servers switched-off thus consuming zero power compared to the $M/M/n/n + s$ queue without hystereses which keeps all servers on and idle consuming 600W without any useful output. As the load increases, hystereses model tends to have lower server activation rate thus achieving lower power consumption. At significantly high loads, e.g. $\rho = 1$, all servers at the DC tend to be switched-on to serve the increased load of incoming requests, thus diminishing the effect of any power saving algorithm. The figure also shows that the hystereses width has almost no effect on the amount of consumed power, which limits the variation in power consumption to the status of idle servers being on and idle or switched-off. Results for a hysteresis width of 1 were not included in the figure due to the lack of a significant effect on the power consumption.

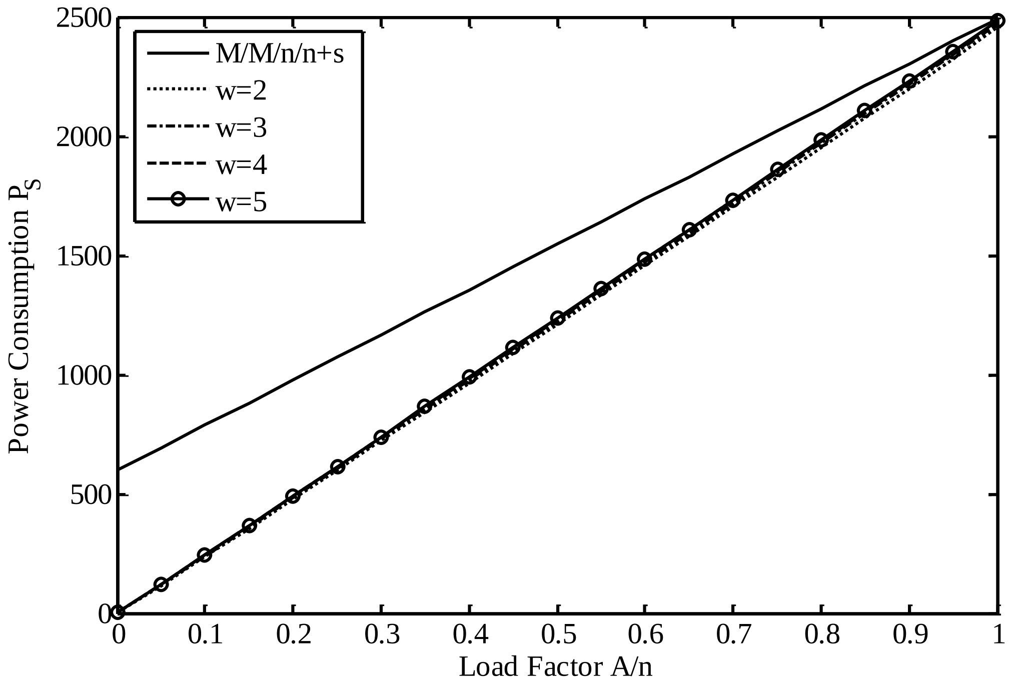


Figure 5-8: Power Consumption of Servers inside the DC for various Hystereses Widths vs. $\boldsymbol{M/M/n/n + s}$ Queue

5.3.1.4 Power Saving Efficiency η_s

Another measure of model's power saving capability is shown in Figure 5-9, which shows the power-saving efficiency of a DC with 100 servers operating under the hystereses model and how efficiency varies with varying the hystereses thresholds. The figure also shows power saving efficiency of an equivalent $M/M/n/n+s$ queue without hystereses for comparison. As elaborated earlier in equation 5-26 the figure shows the amount of power saved by applying hystereses model compared to always-on strategy, where the highest efficiency value would be 1 when all servers are switched-off and the least value approaches zero when the DC is fully operative. As the figure shows; highest efficiency is achieved only when the DC is under low load where the hystereses model saves energy consumed by idle equipment. Power saving efficiency diminishes as the load increases and the DC tends to have all its equipment under high load. Figure 5-9 also shows that efficiency is not affected by hystereses widths.

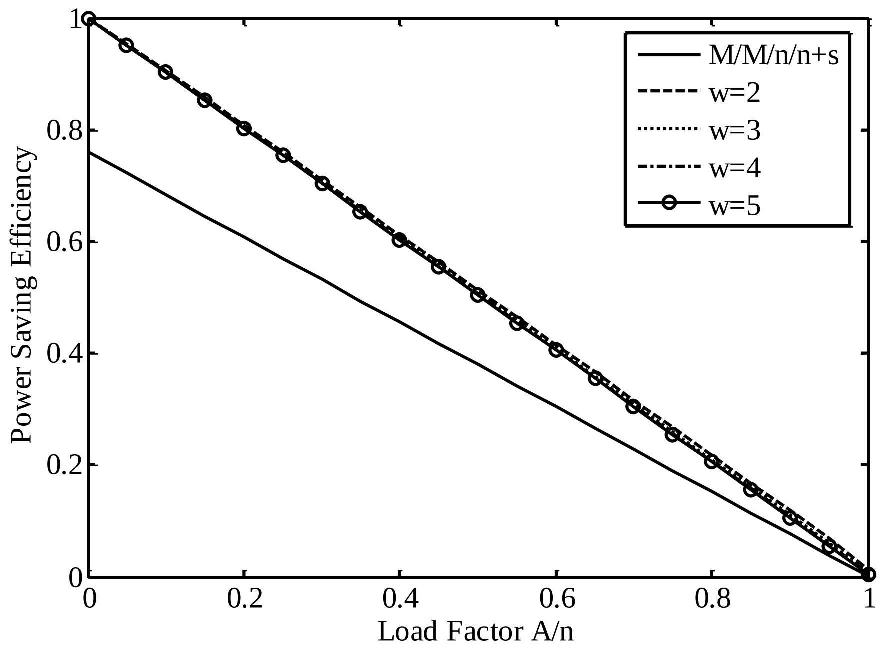


Figure 5-9: Power Saving Efficiency for Hystereses Model with various Widths vs. $\boldsymbol{M/M/n/n+s}$ Queue

5.3.1.5 Mean Delay of Delayed Frames $\boldsymbol{E[T_w|T_w > 0]}$

The main side effect of the hystereses model for reducing power consumption and achieving server consolidation is the increased delays. As more arrivals tend to be buffered in a system with hystereses instead of immediately being served, the mean delay time will increase according to the increase in the hysteresis width, as shown in Figure 5-10. Results are shown for variable hysteresis widths $w = 1,2,3$ and 4 vs. a system without hystereses ($M/M/n/n+s$ queue) to show that although introducing the hystereses improves system performance, it increases the mean waiting time of buffered requests. As explained earlier, this delay increase is the main criteria to be taken into account while designing the system parameters in order not to violate customers' SLAs. In this context the mean delay is calculated for only those arrivals that experience delay in the DC's queue, not with respect to all DC's arriving requests. This is intended as the second calculation method would lower the average delay value and provide a misleading average value for delay, whereas calculating the average while considering only those who wait provides a more realistic measure and could be used as a Quality of Experience 'QoE' metric while the mean delay of all arriving requests $E[T_W]$ is used as a Quality of Service 'QoS' metric. Despite the increased delay, applying hystereses model to a DC achieves an interesting effect of almost constant delays for a huge load range, i.e. average delay is almost constant for all requests while load on the data center is 5-95% loaded. This 'plateau' of delays is highly beneficial for administrators for provisioning required configuration parameters and required hardware for their DC. As the proposed hystereses model and its analytical solution can provide an exact estimate for the average delay of delayed requests, administrators can tune their configuration parameters for hystereses widths as well as buffer sizes to match the delay value required such that users' SLAs are not violated while the load to the DC varies from 5% to 95%.

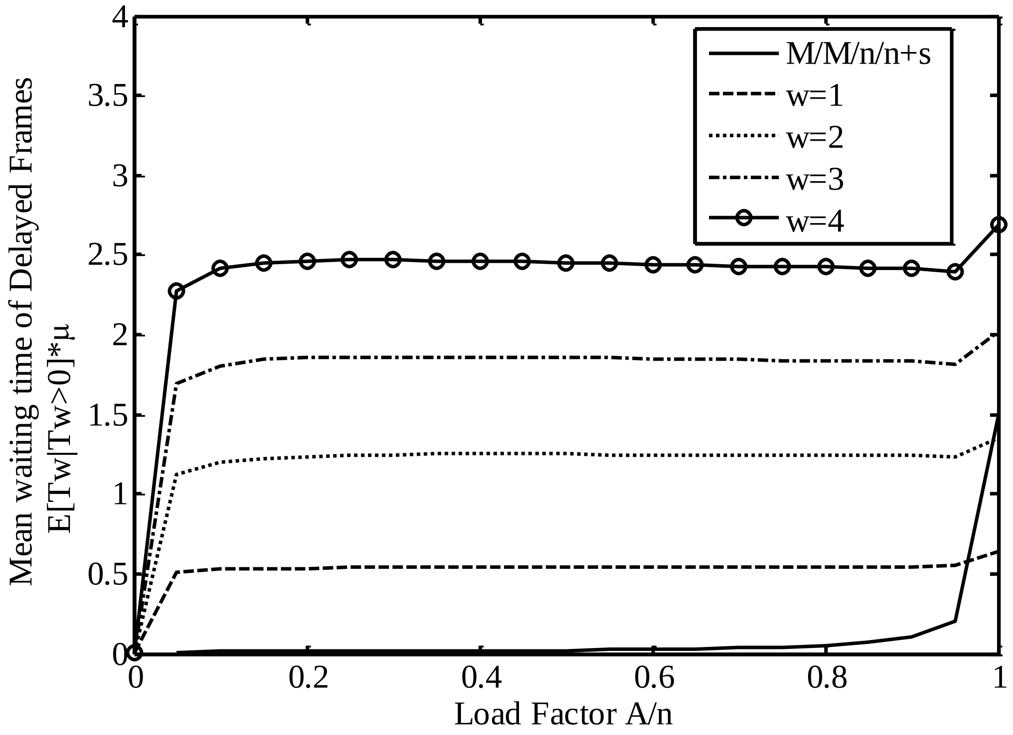


Figure 5-10: Mean Waiting Time of Buffered Requests
$\boldsymbol{E[T_W \mid T_W > 0]}$ for variable Hystereses Widths vs. $\boldsymbol{M/M/n/n+s}$ Queue

Results obtained by the analytical solution shown in Figure 5-10 are supported by results from OMNeT++ simulations for the same test cases. Figure 5-11 shows the delay measured by the simulation model with a 95% confidence interval compared to the average delay calculated by mathematical analyses for delayed arrivals. The figure shows that the analytical solution provides an upper limit for the average delay, where delays measured from the simulation is slightly lower[1]. Simulation tends to provide lower delays due to the nature of the hystereses model; as a new arrival to the queue could increase its length to reach a threshold value at which a new activation is triggered, thus increasing the service rate of the system and reducing delay of buffered arrival. This effect could not be accounted for in the calculated mean delay using Little's Law [80] in equation 5-20.

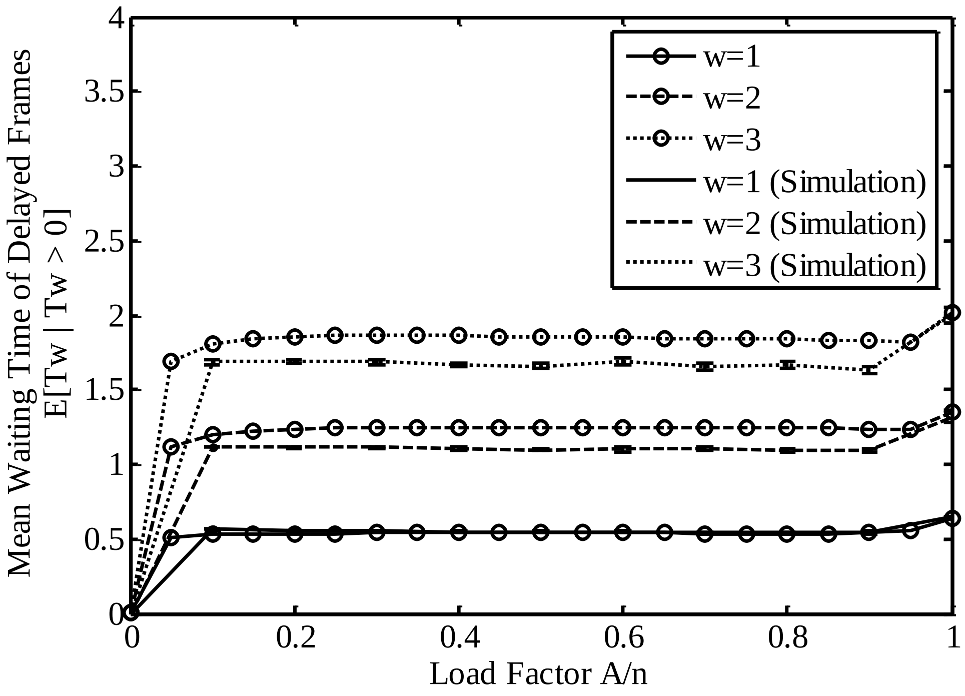


Figure 5-11: Mean Waiting Time of Buffered Requests $\boldsymbol{E}[\boldsymbol{T_W} \mid \boldsymbol{T_W} > \boldsymbol{0}]$ for variable Hystereses Widths (Analytical Solution vs. Simulation)

5.3.1.5 Model Validation

This section provides validation for the proposed hystereses which has proved its efficiency in enhancing DC's performance by achieving server consolidation and reduced power consumption. However the analytical results provided for the model is done under Markovian assumptions, i.e., negative-exponential inter-arrival times and service times as well as negative-exponential activation times for servers. As these assumptions do not apply to all types of traffic or all types of hardware, the model was tested under different distributions for inter-arrival, service and activation times in order to show its validity and applicability to any DC type disregarding the distributions of times mentioned earlier. Various distributions are applied in the OMNeT++ simulation model for a system with parameters $n = 100$, $s = 200$, $w = 2$ and $\alpha = 0.1$ while measuring the mean delay of delayed frames for being the most critical parameter affected by the hystereses model.

[1] The differences between analytic and simulation results originate from the calculation of the mean number of delayed arrivals used in Equations 6.16 and 6.19 as they include cases where servers are in an activation phase but may be deactivated when no new server activation is required, c.f. state transitions indicated by "D" in Figure 5-1.

5.3.1.5.1 Various Inter-arrival Time Distributions

Figure 5-12 shows mean delay of delayed frames for a DC operating under hystereses model while using different distributions for generating random inter-arrival times while simulating the model using OMNeT++. Different mean values for inter-arrival times are used to reflect load variation experienced by the DC, e.g.: for a load value $\rho = x$ per server, the average time between two requests generated at the source module of the simulation would be equal to $t_{inter-arrival} = 1/nx$ as per equation 5-27. Figure shows results while using the following distributions with respective configurations for generating inter-arrival times:

- Exponential distribution with mean $t_{inter-arrival}$
- Constant interarrival times = $t_{inter-arrival}$
- Uniform distribution with limits $[0\,,2 * t_{inter-arrival}]$

For observing effects of variable inter-arrival time distributions, service times' and activation times' distributions were both generated using a negative-exponential distribution function with means 10s and 0.1 seconds, respectively. As the figure shows; measured mean delay is not affected by varying the distribution used for generating inter-arrival times between requests, which proves the validity of the model and its applicability to different distributions of interarrival times rather than Markovian.

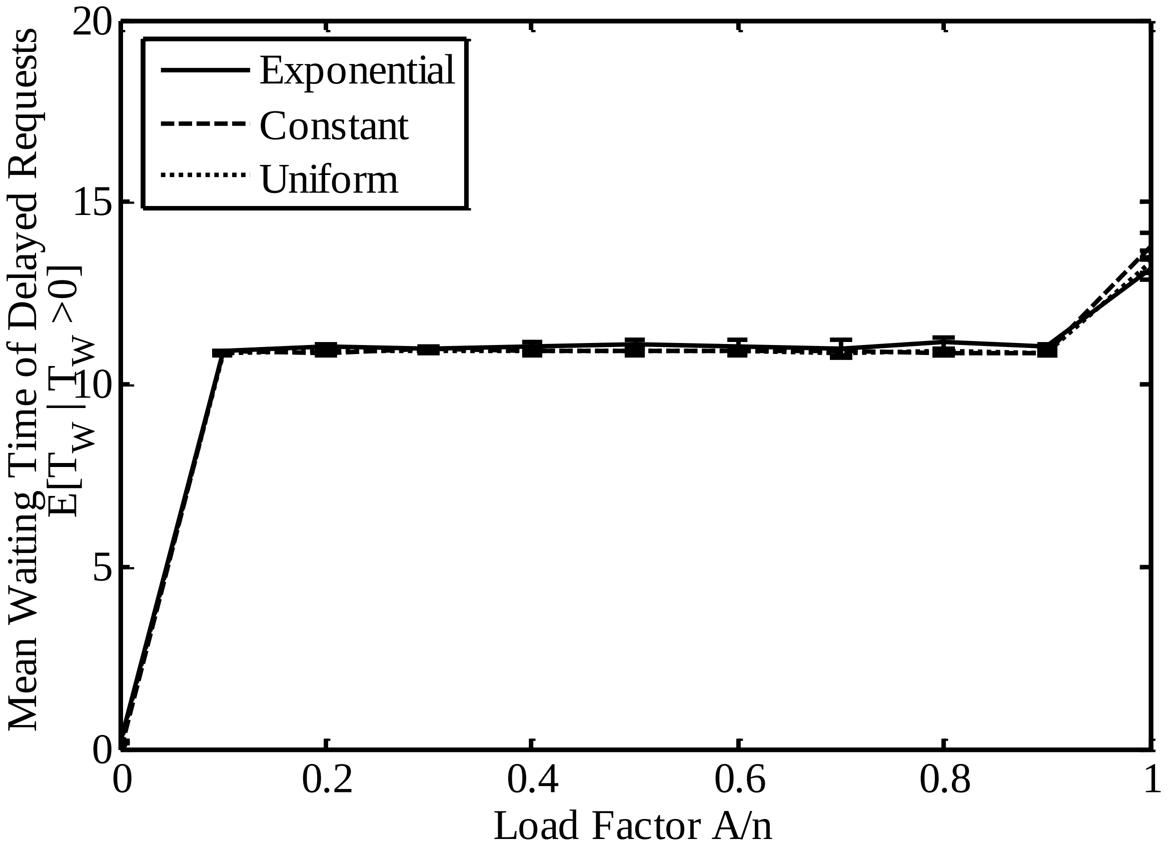


Figure 5-12: Mean Delay of Delayed Requests E [Tw |Tw>0] for different Inter-Arrival Time Distributions

5.3.1.5.2 Various Service Time Distributions

For testing the effect of altering the distribution of service times experienced by requests at servers inside the DC, OMNeT++ simulations were conducted using the following distributions for generating service times for each arrival upon entering the server at which it will be served:

- Exponential distribution with mean 10
- Constant interarrival times = 10
- Uniform distribution with limits [0,20]

Figure 5-13 shows that the mean delay experienced by users while generating exponential service times is not much affected while using different distributions, which proves applicability of the model for any service time distributions and validity of the Markovian assumption used by the analytical solution. Results in the figure were obtained while keeping the generating negative exponentially distributed inter-arrival times with respective means to the required load value as well as negative exponentially distributed server activation times with mean value of 1 which raises the mean delay value to 1s at load values approaching zero. The small dependence of the mean waiting time on the service time distribution is (at first sight) counter-intuitive, but can be explained by making use of the results of Figure 5-7: with increasing values of the hysteresis width the server activation rates decrease rapidly and thus the mean waiting time of delayed frames is less affected by the service time distribution.

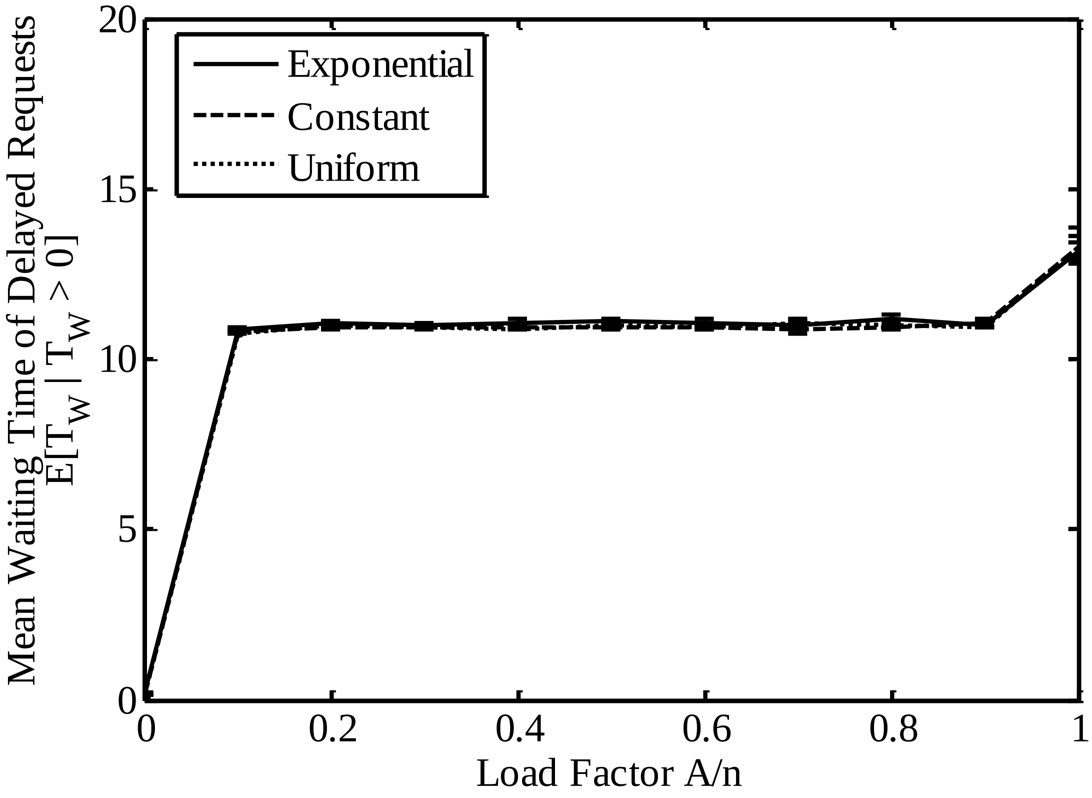


Figure 5-13: Mean Delay of Delayed Requests E [Tw |Tw>0] for different Service Time Distributions

5.3.1.5.3 Various Activation Time Distributions

Similar to the previous two test-cases Figure 5-14 shows the validity of the assumption of negative-exponentially distributed servers' activation times and the applicability of the hystereses algorithm to model any type of DC with any distribution for servers' activation times. Results in the figure were obtained while generating service times according to a negative-exponential distribution with mean value of 10s and inter-arrival times according to the same distribution but with mean values retrieved from Equation 5-27 to reflect the DC's load while distributions for generating servers' activation times varied as follows with $t_{activation} = 0.1$:

- Exponential distribution with mean $t_{activation}$
- Constant interarrival times = $t_{activation}$
- Uniform distribution with limits $[0,2 * t_{activation}]$

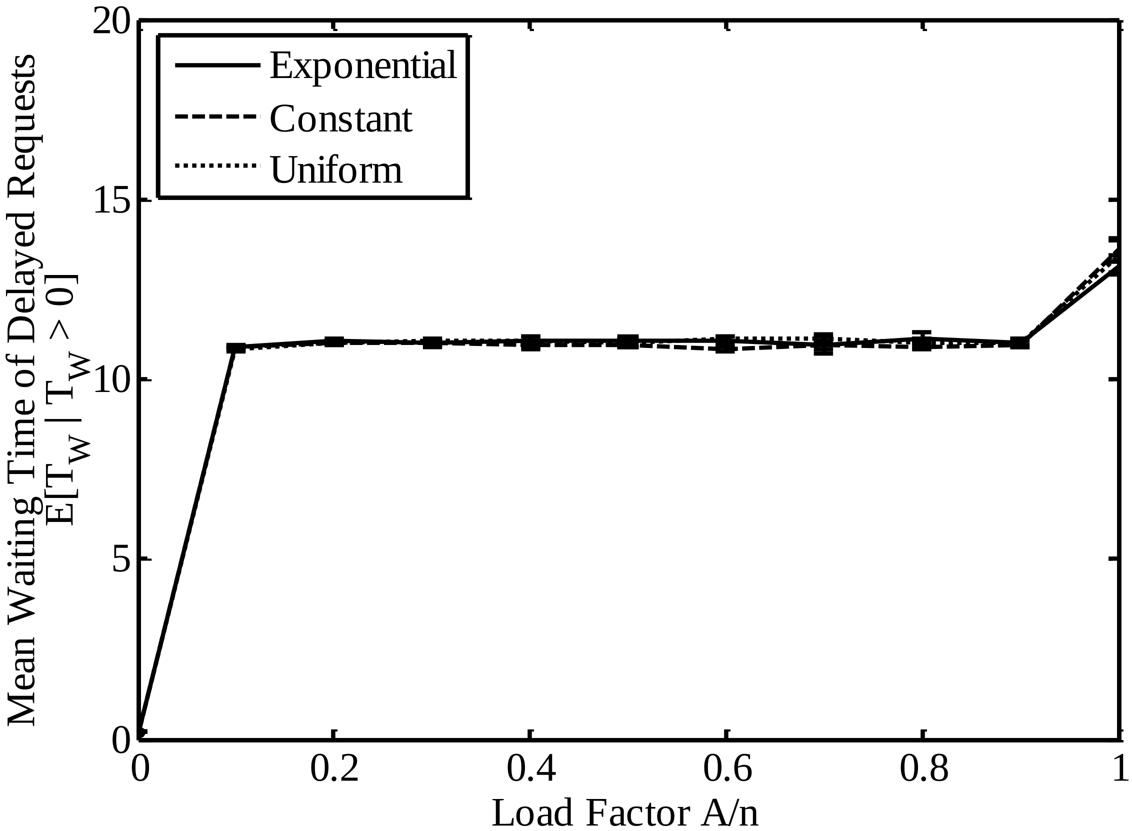


Figure 5-14: Mean Delay of Delayed Requests E [Tw |Tw>0] for different Server Activation Times Distributions

5.3.2 Case Study 1: Sleep Modes and Effect of Server's Activation Rates

One of the novelties of the proposed hystereses model is its ability to model realistic properties of servers such as the time required for activation of switched-off servers and its associated power spike. This subsection compares two approaches taken when a server finishes its service and the DC's queue is empty: Cold Stand-by (CSB) and Hot Stand-by (HSB). CSB is the basic approach where a server is completely deactivated when it becomes idle, and has to be booted when a server activation is triggered. HSB avoids this booting delay and instead puts servers in a sleep mode where they are in-active, consuming less energy and can be activated again faster than CSB mode. HSB mode provides several sleep states depending on which components of the server are put to sleep, where deep sleep states with more inactive components require longer activation times than light sleep states. This can be represented in the proposed hystereses model by varying the activation rate of servers and thus their average booting/wakeup times. As CSB requires long activation times to boot servers, servers in CSB mode require a smaller α.; whereas servers in HSB mode have large α as they require shorter wake-up times.

In addition to the variation in activation times between turned off/sleep states, a variation in power consumption must also be considered. Totally switched-off servers in CSB mode consume no power; however sleep states specified by HSB mode still consume power as not all components of the server are put to sleep. For evaluating the efficiency of sleep modes integrated into the proposed hystereses model, power consumed by DC with/without adopting sleep states are compared as well as the average delay experienced by delayed arrivals. The power consumed by servers in CSB or HSB mode is calculated according to the following equations:

$$P_{S,CSB} = Y_s * P_{running} + Y_A * P_{running} \tag{5-31}$$

$$P_{S,HSB} = (Y_s + Y_A) * P_{running} + (n - Y_s - Y_A) * P_{sleeping} \tag{5-32}$$

$P_{running}$ follows from [5] to be equal to 25W as the energy consumed by a server operating at full capacity, while $P_{sleeping}$ is assumed to have an average value of 1.56W according to [53] where sleeping states consume an average of 26% of idle power consumption. This value is assumed for all sleep states since the difference in power consumption between different sleep states is minor as components that are already idle have low power consumption. Following from Equations 5-31 and 5-32 power-saving efficiency for CSB and HSB modes by which the amount of power saved by these power saving modes compared to always-on strategy can be calculated as follows:

$$\eta_{CSB} = (n * P_{running} - P_{S,CSB})/n * P_{running} \tag{5-33}$$

$$\eta_{HSB} = (n * P_{running} - P_{S,HSB})/n * P_{running} \tag{5-34}$$

Figure 5-15 shows plots for power saving efficiency obtained using Equations 5-33 and 5-34 for a DC with parameters $n = 100$, $s = 200$ and $w = 2$ by varying the value of activation rate α as follows: Since CSB mode shuts down any idle server and boots it again when a server activation is triggered, it requires long mean activation times and thus smaller activation rates shown in Figure 5-15 as $\alpha = 0.25$. For HSB mode idle servers are put to sleep at deep/light sleep modes, thus require shorter wake-up times and larger activation rates varying between 0.5,1,4 and ∞ where deeper sleep states have smaller α and lighter sleep states have higher α. As the figure shows power saving efficiency of CSB mode is higher at low loads when the system is lightly loaded and not fully utilized, so it saves power consumed by idle servers by turning them off. As load increases; HSB mode outperforms CSB mode as DC utilization increases and the average number of idle servers as well as their idle durations are highly reduced which eliminates the power saving effect by CSB mode. At these load ranges and according to Figure 5-7 servers are likely to have high activation rates where power-saving efficiency of HSB dominates since it eliminates the power spikes associated with reactivation of sleeping servers. As α increases and time required to wake up a sleeping server is reduced, power saving efficiency of HSB mode at relatively high loads also increases. For validating power saving capabilities of the hystereses model using either CSB or HSB modes, an equivalent $M/M/n/n+s$ queue operating at always-on strategy is shown in Figure 5-15 to have much lower power-saving efficiency especially at low load regions when all servers are kept on without being utilized.

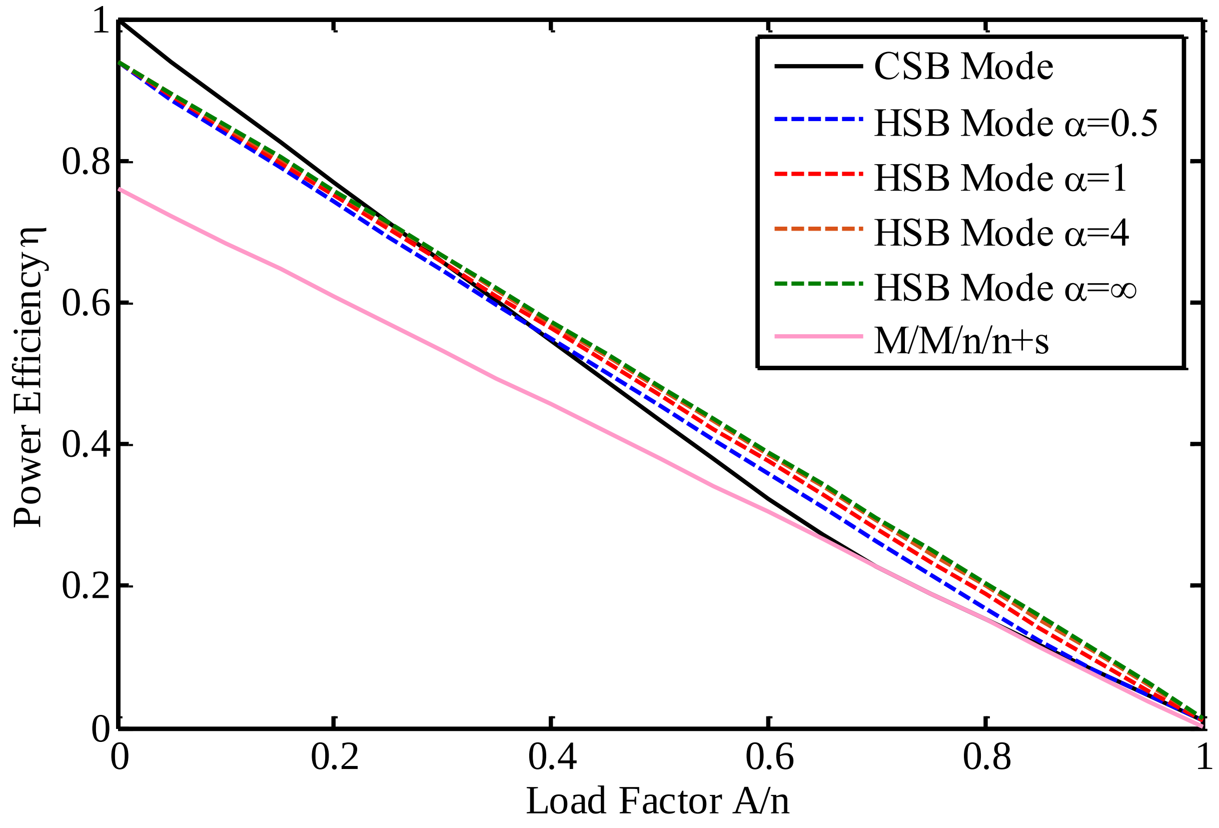


Figure 5-15: Power Saving Efficiency for CSB Mode vs. HSB Mode with various Sleep States

Having explored the efficiency of the proposed sleeping modes within the hystereses model in saving power consumed by the DC, their effect on the mean delay of requests at the

DC is shown in Figure 5-16. Figure 5-16 shows mean delays for variable server activation rates, where the average time taken by a server to be activated could be obtained as $1/\alpha$. CSB mode shows the highest average activation time; i.e. 4 seconds corresponding to an average activation rate of 0.25 per server which is visible at load values approaching zero where the only delay experienced by an arriving request is that while a server is being activated. As load increases and more severs are activated the mean delay value drops and stabilizes until the DC is almost 95% loaded. HSB mode have shorter wake-up times depending on which C state the server entered after being idle, where deep sleep states require longer wake-up times which is represented in the figure by an average activation rate of 0.5,1 and 4 for deeper and lighter sleep states, respectively. As the figure shows, small activation rates result in longer activation times, which increase the average delay of buffered requests. This effect is also mostly obvious at small loads, when almost all requests arrive at the DC require a server to be activated. When compared to an equivalent $M/M/n/n + s$ queue with parameters, it can be noticed that the proposed hystereses model and its sleep modes highly increase the mean delay value, which is the main drawback for power-saving efficiency. However, as the model and its recursive solution can accurately predict the average delay of delayed requests according to system parameters, they can be easily tuned to match the required delay value according to users' SLAs'

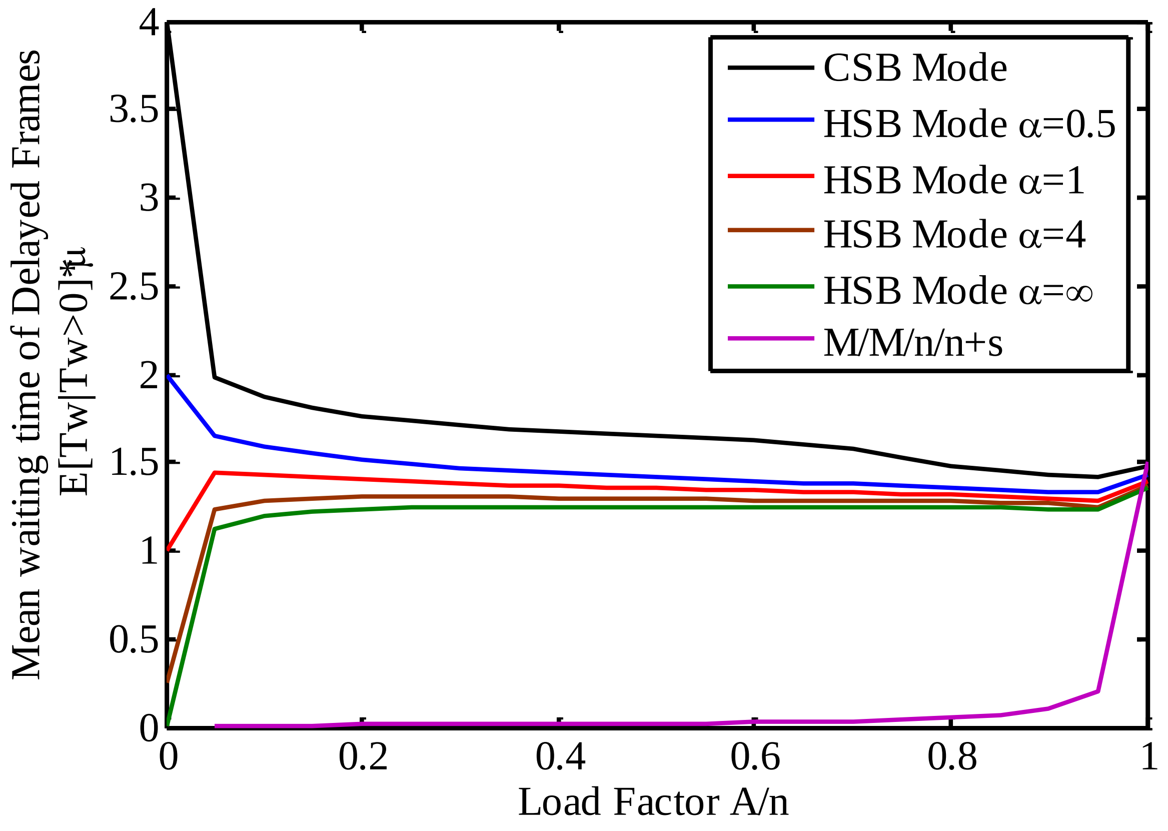


Figure 5-16: Mean Waiting Time of Buffered Requests $\boldsymbol{E}[\boldsymbol{T_W} \mid \boldsymbol{T_W} > \boldsymbol{0}]$for variable Activation Rates vs. $\boldsymbol{M/M/n/n + s}$ Queue

5.3.3 Case Study 2: Reduced Service Rates by DVFS

As explained briefly in sub-Section 3.1.3, DVFS is an approach that reduces the frequency by which a server operates by specifying various P-states each with a defined operating frequency and voltage. Various P-states correspond to different power consumption modes, where the reduction in operational frequency results in a corresponding reduction in consumed power. P-states have a main difference to C-states as illustrated in the previous section: a server enters a P-state when it is busy serving a request, but reduces its service rate for energy efficiency purposes, whereas C-states are only entered by servers when they are completely idle.

In the STD of the hystereses model shown in Figure 5.1 the service rate of any state (x, z) is represented as $\mu_{x,z}$, which follows from Equation 5-27 in the basic cases without DVFS as the total service rate of a homogeneous server system is the summation of service rates of all servers.

$$\mu_{x,z} = x\mu \tag{5-35}$$

To reflect a DC implementing DVFS technology, $\mu_{x,z}$ can be varied for each state of the DC to represent different service rates for different P-states servers can operate at. Typically, reduced service rates are adopted when the DC is experiencing low loads to avoid severe performance degradation. A relatively low load situation can be represented in the hystereses model by a defined low queue threshold z^*; such that when the queue size is lower than the buffering threshold servers operate at a reduced service rate of μ^* per server; according to Equation 5-36:

$$\mu_{x,z} = x\mu \quad for\ z > z^*$$

$$\mu_{x,z} = x\mu^* \quad for\ z \leq z^* \tag{5-36}$$

For calculating the consumed power by servers operating under DVFS strategy, the following equation is used:

$$P_S = P_{running} * \sum_{x=1}^{n} \sum_{z=z^*+1}^{s} x * p(x, z) + P_{DVFS} * \sum_{x=1}^{n} \sum_{z=0}^{z^*} x * p(x, z) \tag{5-37}$$

The first part of the equation calculates the power consumed by servers operating at full speed when buffer size is above the threshold by multiplying the average number of servers operating at full power by the power consumption of a server operating at full speed $P_{running}$. The second part of Equation 5-37 calculates the reduced power consumed by servers with reduced service rates when buffer capacity is less than or equal to the defined buffering threshold for DVFS, where P_{DVFS} varies depending on the P-state at which a server operates and μ^* varies accordingly:

$$\mu^* = P_{DVFS}/P_{running} \tag{5-38}$$

Figures 5-16, 5-17and 5-18 shows the performance of a DC operating under hystereses model while implementing various P-states a server can enter by reducing its operational frequency to reduce its power consumption. P-states used in these test cases are adopted from [5] where P1 is the state with highest frequency and performance and P5 is the state with lowest frequency and lowest performance. Figure 5-17 shows mean delays of buffered arrivals for a DC operating in CSB mode with parameters $n = 100$, $s = 200$, $w = 2$, $\alpha = 1$ and $z^{*} = 5$ where servers are only allowed to reduce their operational frequency when the queue size is lower than or equal to 5 buffered arrivals. As this buffering threshold is only reached at low load values, it results in a delay spike as shown in Figure 5-17 which increases as frequency decreases. This increase in delay is caused by the reduction in the operational frequency of the server which elongates the time it requires to finish servicing an arrival, thus increasing service time and mean waiting time of arrivals. Reduced service rates result also in an increased buffer size which triggers more servers to be activated while hystereses thresholds are reached. The increased activation rate of servers around load regions where queue size is equal to z* illustrated by Figure 5-18 causes more servers to be brought into service than the number required to serve the current load value. As the load value increases with a corresponding increase in buffer size, the increased number of activated servers exit P states and increase their service rates to accommodate the increasing load, which is reflected in Figure 5-17 by a rapid decrease in the mean delay of delayed arrivals as the service rate of the DC rapidly increases. Figure 5-17 also shows that the more frequency is reduced, the more average delay increases when servers operate at P states causing system buffer to be filled faster thus triggering more servers for activation and increasing server activation rate.

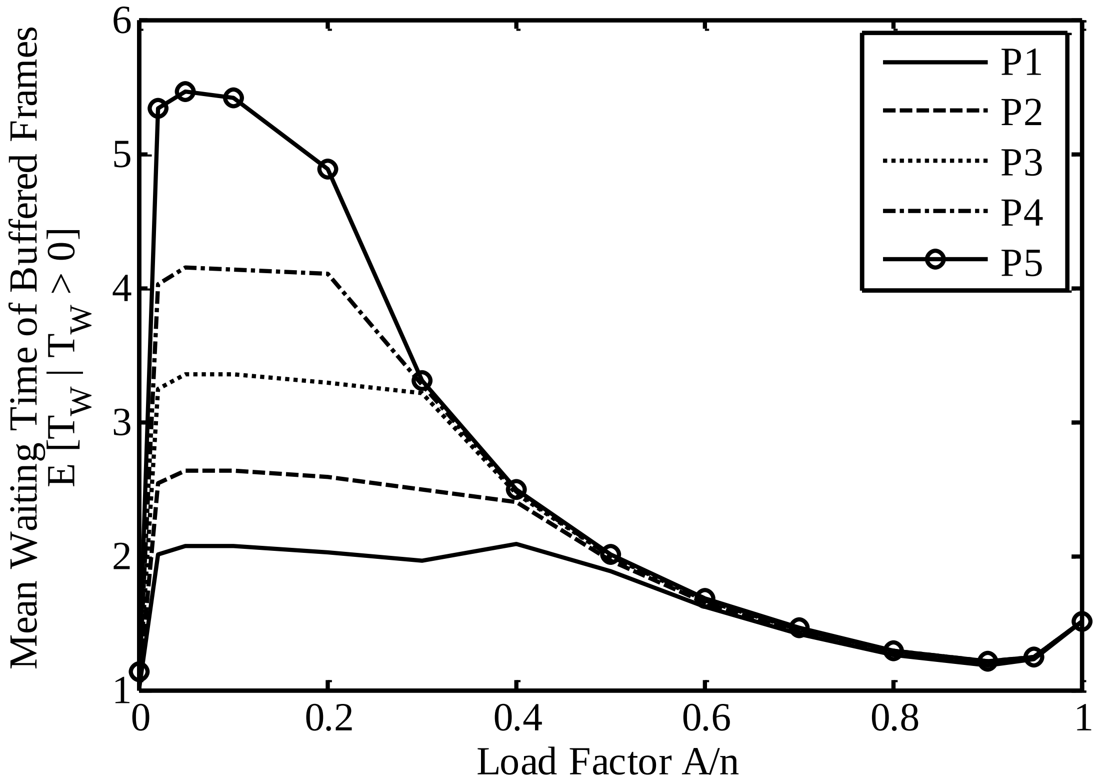


Figure 5-17: Mean Waiting Time of Delayed Frames for Hystereses Model under different P-States

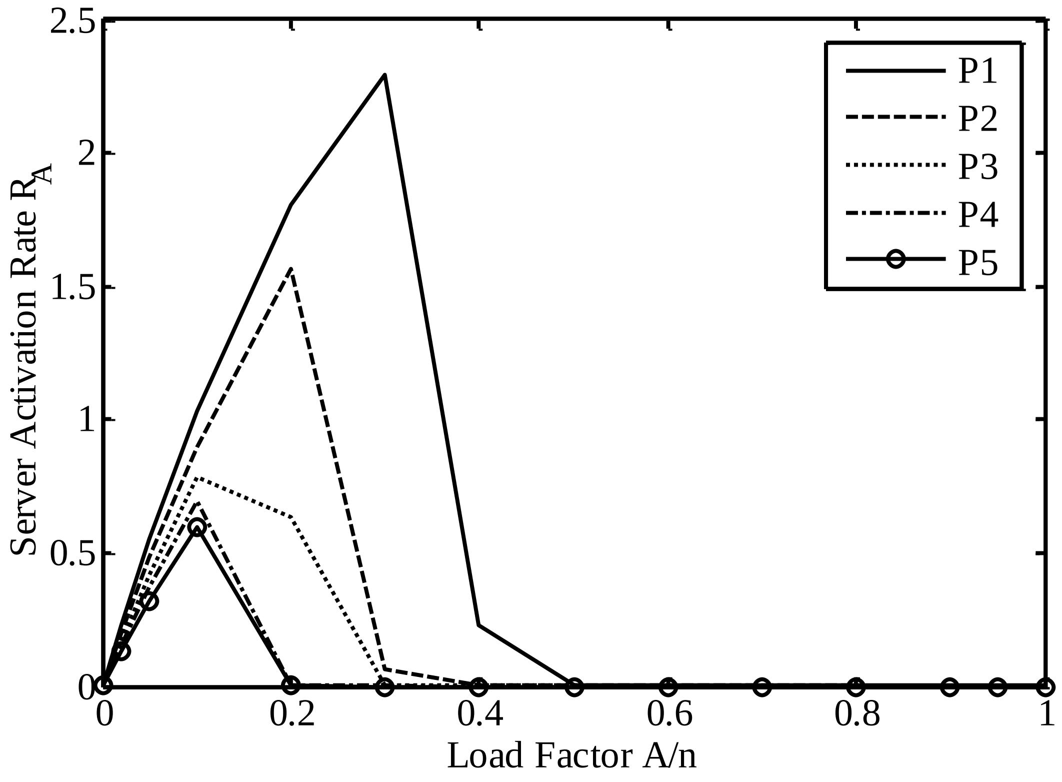


Figure 5-18: Server Activation Rate for Hystereses Model with DVFS under different P-States

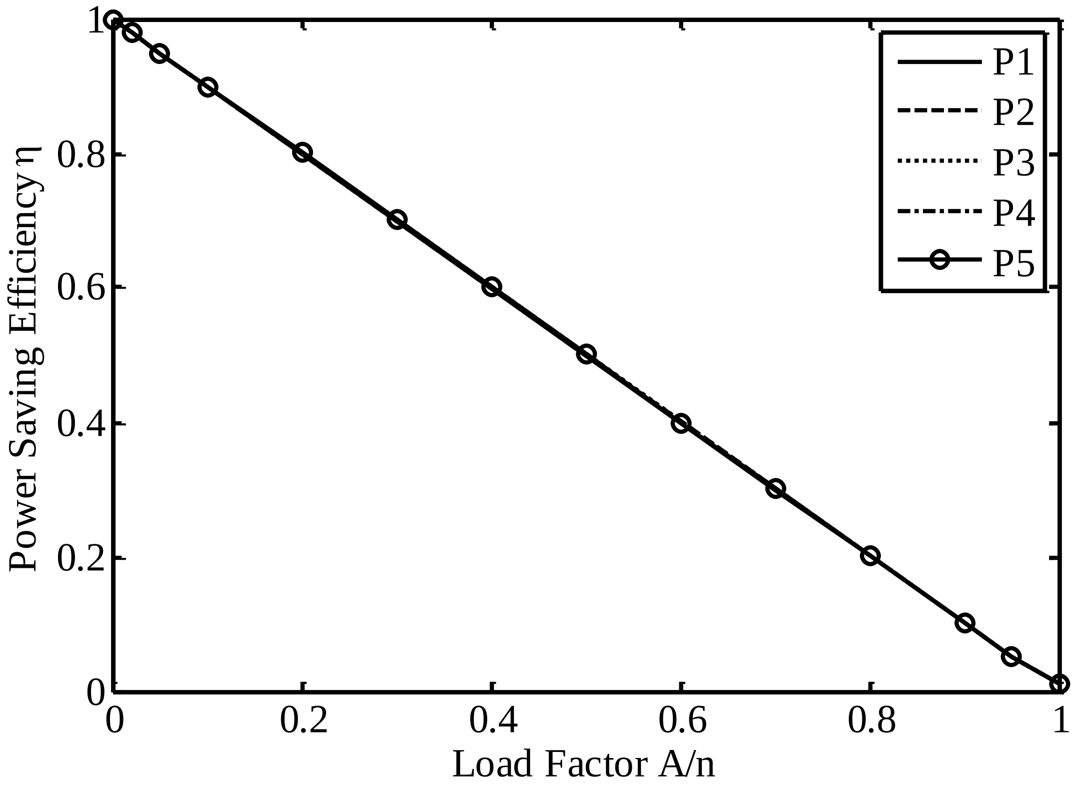


Figure 5-19: Power Saving Efficiency of Hystereses Model under different P-States

Despite the increase in mean delay noted in Figure 5-17, the corresponding power saving efficiency due to dynamic frequency scaling is minimal. Figure 5-19 shows the efficiency of hystereses algorithm with DVFS at saving power for different P-states, where graph lines overlap on the figure due to the minimal differences in efficiency. This is explained as follows: despite the expectation that the amount of power consumption should be reduced by reducing the operational frequency of servers, this reduction is power consumption by active servers is compensated by more servers triggered for activation and becoming active due to the increased service time, mean delay and accordingly buffer size which triggers more servers to be activated thus consuming more power. This hypothesis is supported by the shifted peak of server activation rates in Figure 5-18 compared to a hystereses system without DVFS in Figure 5-7.

Another parameter of interest is when to put servers in the DC into P-states. Figures 5-20 and 5-21 showcase results of a DC whose servers operate at P3 state when z^* reach different queuing thresholds $z^* = 5, 50, 100, 150$. Despite power saving efficiency is not increased due to the increase in number of activated servers compensating the energy saving by reduced operational frequency as Figure 5-20 shows, the mean delay of delayed requests is highly affected by increasing z^*. Figure 5-20 illustrates that as z^* increases delay rapidly increases due to the reduction in service rates over longer load values which elongates durations spent by arrivals in the DC's buffer. This increased mean delay value is maintained over longer load ranges as servers keep operating in P-states for longer times until buffering threshold z^* is reached.

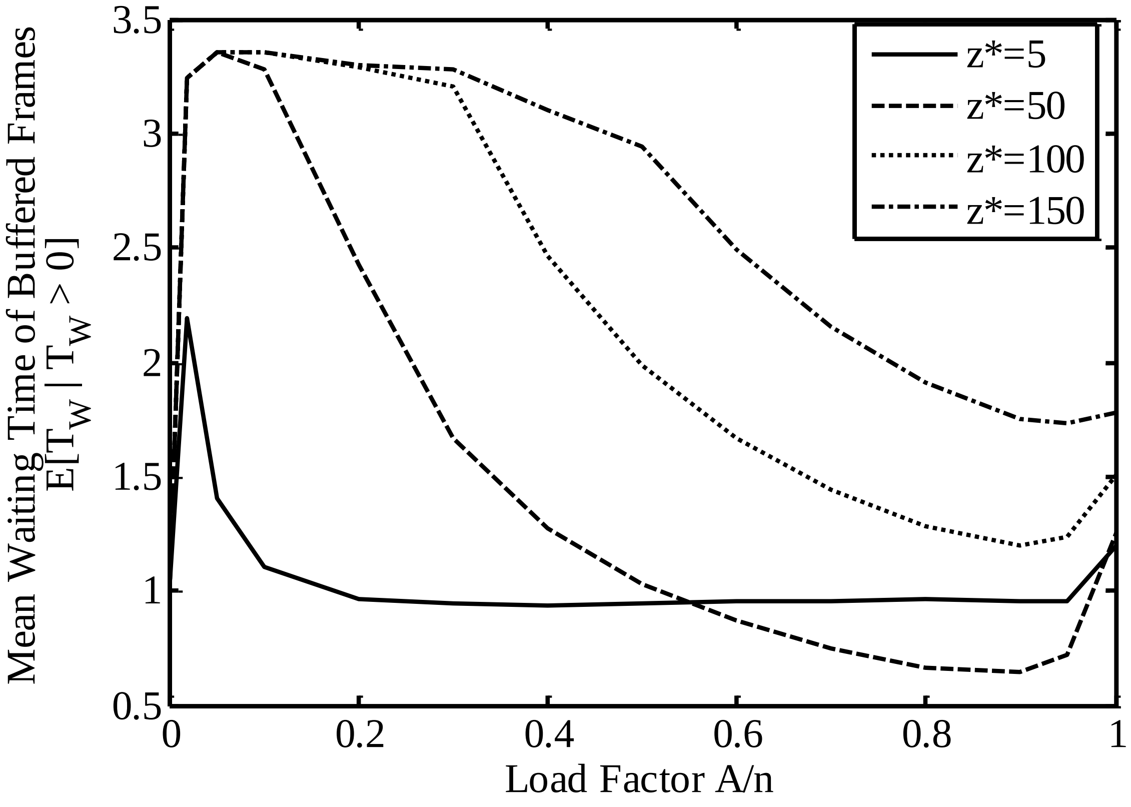


Figure 5-20: Mean Waiting Time of Buffered Frames for Hystereses Model under DVFS with different values for Parameter z*

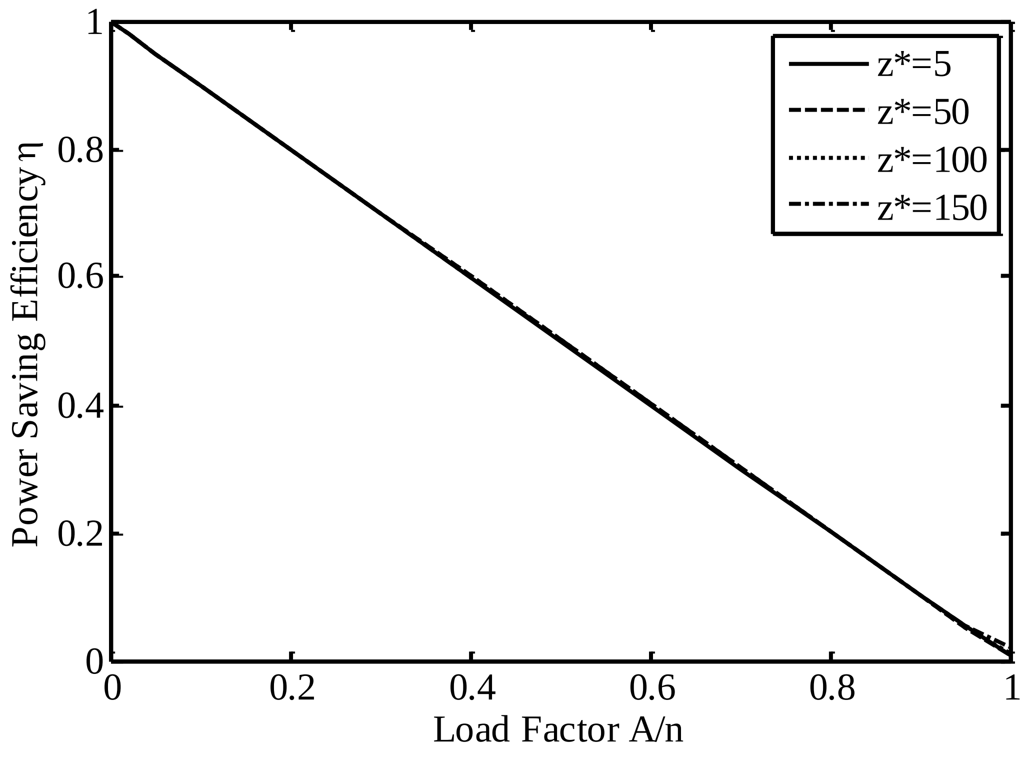


Figure 5-21: Power Saving Efficiency of Hystereses Model under DVFS with different values for Parameter z*

Choosing among different parameters for z* and P-states under test in this section should always be done while considering users' SLAs, as attempts to reduce power consumption or number of activated servers always result in degradation in DC's performance in terms of increased delays for delayed arrivals. Important to be mentioned also is that all before mentioned results show that application of DVFS in a DC under hystereses model and the reduced service rates due to frequency reduction alter its server consolidation effect. This is most visible in altering peak values of server activation rates and ant which load values they are achieved, as well as altering the concentration of probabilities of x active servers at respective load x, which defies the goal of the model to consolidate the DC's load on the least number of possible servers to maintain users' SLAs.

Chapter 6 Load Balancing Algorithms between Cloud Data Centers

This chapter introduces two algorithms for the purpose of balancing the load dynamically among data centers in the cloud to maintain an efficient performance during peak-load periods. As previously explained in Chapter 3, load balancing between cloud DCs is required in order to avoid in-efficient load distribution where some DCs suffer an overload that can result in performance degradation and violation of its customers' SLAs while other DCs are under-utilized. Dynamic Load balancing also helps in providing better utilization of resources as well as maintaining prescribed SLAs for customers.

The novelty of the algorithms proposed in this thesis results from the fact that they are using decentralized, dynamic algorithms that perform load balancing decisions at the instant of each request's arrival. As concluded from the discussion provided in Chapter 3 for existing load balancing techniques in literature most algorithms had drawbacks of being static, centralized, having long processing times or result in reduced resources' utilization and long average response times for requests. The two proposed algorithms avoid mostly all of these issues by having the following properties:

- Algorithms operate separately at each DC avoiding the need for any centralized nodes that may cause bottlenecks or single points of failure.
- All load balancing decisions are taken considering the current state of the local DC as well as foreign DCs to which requests could be migrated to. Future states of the DCs after requests arrive to them are also considered to make sure that requests are balanced optimally between DCs without causing overload at any of them.
- Unlike the common convention of load balancing algorithms to balance the load evenly between DCs, the proposed algorithms do not encourage migrations between DCs unless needed for providing better service for users. The two algorithms compare between the approach of balancing the load only at overload situations and thus, keeping the un-balanced load situation as long as performance is not degraded, and balancing the load at low and high load regions for achieving lowest average waiting time possible.
- Maintaining service level agreements of requests in terms of average response times is the main criteria for the load balancing process. If at the instance of each arrival a request cannot be served at its own DC according to prescribed SLAs, it will be migrated to another DC offering better response time.
- Algorithms account for overheads resulting from migrating requests between DCs, such as transmission delays due to geographical distances between DCs which must be taken into account to determine whether migration would be beneficial or not, as well as for the overhead required for communicating the actual load situations at different DCs.

This chapter models the load balancing process between two DCs by two different algorithms: Local Server System First (LSSF) and Shortest Response Time First (SRTF), each defining a preference for where to schedule a request at its arrival instant. For each algorithm, the first section models the system using a two-dimensional Markov Chain representing the states of the two DCs, and illustrating all the possible scenarios that could occur to a DC, from accepting a request, migrating a request, or declining to serve it. Theoretical analysis for solving the Markov Chain represented by the state transition diagram of the model using a simple recursive solution is explained along with computations of significant performance metrics for evaluating the system. Section two in each algorithm explains a simulation model of the algorithm using OMNeT++, where the algorithm is simulated for two DCs to verify analytical solution under Markovian assumptions, and is then extended to more than two DCs and general interarrival and service distributions. In the third section each algorithm is implemented on the test bed described in Chapter 4 for actual testing of the algorithm using real traffic requests.

6.1 Algorithm 1: Local Server System First (LSSF)

Local Server System First algorithm is based on the idea of maintaining unbalanced load among cloud DCs as long as required to meet SLAs. The algorithm balances the load by migrating arriving requests from highly loaded DCs to lightly loaded ones in case of overload situations only if the user SLAs' cannot be maintained at the arriving request's local DC. At low and average load situations, an arriving request would always be directed to its local server system first, only until the local server system is totally occupied at which case the algorithm is triggered. If an arrival doesn't find a place to occupy at its local DC, foreign DCs are checked if they can accommodate this arrival; if a match is found while considering migration overhead to this DC then the arrival will be migrated and served, otherwise the arrival will be lost.

6.1.1 Model Definition

For modeling the Local Server System First (LSSF) load balancing algorithm DCs are abstracted as multi-server queuing models, as explained previously in Chapter 5. The most basic case of the algorithm is illustrated in Figure 6-1 in the form of two DCs with mutual overflow of requests between them. Each DC is assumed to be aware of its own current state along with current state of the foreign DC. This could be implemented by having the two DCs exchange their current states via periodic update messages, or through a central controller to which current states are sent then it broadcasts them back to all nodes. For ease of analysis, the DCs studied in the basic case illustrated in this section are assumed to be homogeneous where each DC has the same number of homogeneous servers having equal service rates, and queues of all DCs operate according to FIFO queuing discipline. Parameters for the model in the Figure 6-1 are explained in the following Table 6-1 for $DC_i, i = 1,2$.

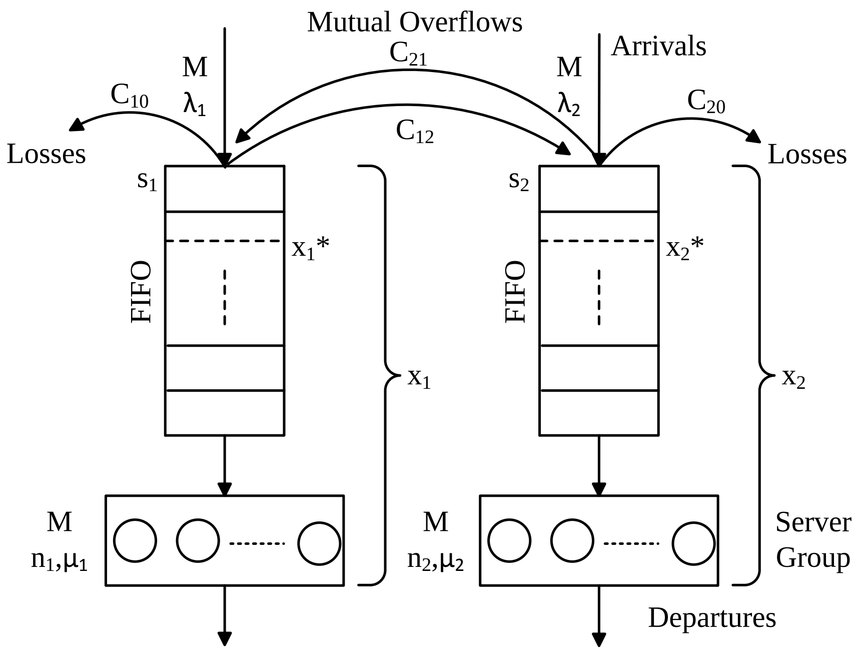


Figure 6-1: System Model for two Data Centers under LSSF Algorithm

Table 6-1: Definition of Model Parameters

n_i	Number of homogeneous servers in each DC_i
μ_i	Service rate of each one of the homogeneous servers in DC_i
s_i	Buffer size at each DC_i where buffer sizes are dimensioned such that arrivals' SLA is maintained even for the last arrival in the buffer in terms of average response time. In this model with Markovian service times and for defined delay threshold $t_{W,T}$ the mean response time for requests t_{wi} should never exceed the delay threshold, as explained by Equation 6-1. Accordingly, s_i is dimensioned by the following Equation 6-2: $$t_{wi} = \frac{s_i}{n_i\mu_i} \le t_{W,T} \quad i=1,2 \qquad (6\text{-}1)$$ $$s_i \le t_{W,T} * n_i\mu_i \qquad (6\text{-}2)$$ Equation 6-1 is derived from the following reasoning: the average waiting time for any arrival before it could start being serviced can be derived by dividing the number of arrivals ahead of it in the queue by the service rate by which these arrivals are served. When the last arrival occurs to the queue at position s_i, this arrival will have $s_i - 1$ customers ahead of it in the queue, plus one customer in the server at which this arrival will be served. The average service rate of the system at the instance of an arrival catching the last place in the queue is $n_i\mu_i$ as all servers are active and each is serving

	requests at a rate of μ_i. By dividing the number of arrivals ahead of the arrival occupying the last place in the queue by the DC's total service rate its average waiting time can be obtained.
λ_i	Arrival rate of requests to DC_i
M	Refers to Markovian interarrival and service times which are negative-exponentially distributed
x_i	Number of customers at each DC_i, either being served or in the queue
x_i^*	Threshold for accepting any arrivals migrated from an outside DC. If the current system state (i.e. number of customers in the system) is less than or equal to x_i^* then DC_i will accept a migrated arrival, otherwise only local arrivals are accepted until reaching buffer capacity. x_i^* is dimensioned such that the average response time of a migrated arrival to DC_i added by the migration time t_M from its overloaded DC_j to DC_i could still meet the average response time defined by SLA: $t_{W,T}$, as illustrated by equation 6-3. $t_M + \frac{x_i^* - n_i + 1}{n_i \mu_i} \leq t_{W,T}$ (6-3) Equation 6-3 follows the following reasoning: an arrival to DC_i when it is occupied by x_i^* customers will find $x_i^* - n_i$ customers ahead of it in the queue, so the arrival has to wait for these customers to become scheduled for service plus the residual time until the first service termination occurs. By dividing the number of customers in the foreign DC by the service rate $n_i \mu_i$ at which the migrated arrival will be served, the average response time of the migrated arrival could be calculated. As the average response time met by a new arrival is constituted by cumulating service times of arrivals ahead of it in the system, it is determined to follow an Erlangian distribution of order $x_i^* - n_i + 1$. The migration time is assumed in this context for simplicity to have a constant duration, which results in a shifted Erlangian distribution for the total average response time by t_M. Average response time added to the migration time from the overloaded DC_j to DC_i should be less than the average response time specified by SLA. Accordingly, x_i^* could be calculated using Equation 6-4: $x_i^* \leq (t_{W,T} - t_M) * n_i \mu_i + n_i - 1$ (6-4)
C_{i0}	Logical condition for rejection of an arrival at DC_i
C_{ij}	Logical condition for migrating an arrival from DC_i to DC_j

The load balancing model operates according to the following strategy: At the instance of an arrival occurring at any DC_ii from its own customers, one of the following cases will occur according to the system state:

- If at least one server is idle then the arrival will be immediately served at its local DC
- If all servers are occupied and $x_i < s_i$, the arrival will be buffered at its local DC
- If the local DC of the arrival is completely occupied, then the arrival could be migrated to another DC where its SLA could be maintained. Thus, the local server system is always preferred by an arrival unless the system is totally occupied. Job migration between DCs has an overhead represented by migration time from one DC to another, indicated as t_M. If no available DC is suitable for the job to migrate to then the arrival will be dropped.

If an arrival didn't find a place at its own server system, then a foreign DC will decide whether to accept or reject the arrival based on the following conditions illustrated in Figure 6-1 for the basic case of only two data centers:

- C_{12}: $(x_1 = n_1 + s_1)$ AND $(x_2 \leq x_2^*)$

  An arrival will be migrated from DC_1 to DC_2 iff DC_1 is fully occupied, and the number of requests in DC_2 is less than or equal to the defined threshold x_2^*.
- C_{10}: $(x_1 = n_1 + s_1)$ AND $(x_2 > x_2^*)$

  An arrival will be dropped from DC_1 iff DC_1 is fully occupied, and the number of requests in DC_2 is greater than the defined threshold x_2^*.
- C_{21}: $(x_2 = n_2 + s_2)$ AND $(x_1 \leq x_1^*)$

  An arrival will be migrated from DC_2 to DC_1 iff DC_2 is fully occupied, and the number of requests in DC_1 is less than or equal to the defined threshold x_1^*.
- C_{20}: $(x_2 = n_2 + s_2)$ AND $(x_1 > x_1^*)$

  An arrival will be dropped from DC_2 iff DC_2 is fully occupied, and the number of requests in DC_1 is greater than the defined threshold x_1^*.

After the handling of arrivals is decided upon (admitted to the queue/migrated/rejected), accepted requests are served in a First-In First-Out (FIFO) order. By implementing this algorithm SLAs of requests are guaranteed to be maintained in terms of average response times; losses are reduced since arrivals that arrive at a fully occupied DC have an opportunity to be migrated and served elsewhere, and finally, DCs are protected from reaching overload states by rejecting migrating requests upon a defined threshold so that local arrivals have priority to be served at their local DCs.

6.1.2 Model Analysis

In order to analyze the proposed LSSF algorithm and understanding its behavior, the basic case of balancing the load among two DCs is first studied analytically using Markov Chains. A state transition diagram representing system's states is introduced and explained followed by an algorithm for solving the steady state probability for each system state which are then used to calculate significant performance metrics to evaluate the algorithm such as loss, delay and migration probabilities as well as mean waiting times. Following sub-sections introduce a simulation model for the algorithm to verify the analytical model and extend it beyond Markovian assumptions for interarrival and service times, as well as testing the algorithm between two servers in a DC using the test bed explained earlier in Chapter 4.

6.1.2.1 Mathematical Analysis

For evaluating the model's performance, this section shows a representation of the model using two-dimensional Markov Chains. Figure 6-2 shows the state transition diagram for two data centers while applying the previously explained LSSF load balancing algorithm between them. Each state in the state transition diagram is represented by $(x_1\ , x_2)$, where x_1 represents the number of requests in DC_1, and x_2 represents the number of requests in DC_2, either being served or queued while waiting for service. Transitions between states are indicated by directed arrows from one state to another labeled by the transition rate that caused this transition. Migration thresholds X_j* and X_i* for migration from DC_i to DC_j and vice versa, respectively are indicated in the diagram by dashed lines.

Generally, when the system is at any general state $(x_1\ , x_2)$, expected events are either an arrival event or a departure event at either one of the DCs. The system state changes according to each of these events as explained below for DC_1 and the same applies at DC_2:

1. In case of an arrival event of a local customer occurring at DC_1 at rate λ_1:
   If $x_1 < n_1$ then the arrival will be served immediately by one of the available servers at the DC. If $n_1 \leq x_1 < n_1 + s_1$ the arrival will be accepted and queued as all servers are busy and the queue is not yet fully occupied. In both before-mentioned cases the state of the system will change to $(x_1 + 1, x_2)$. If $x_1 = n_1 + s_1$ then the arrival cannot be accepted at its local DC, so other DCs are checked for availability to accommodate this arrival so that it is not lost. Availability is determined by migration thresholds of foreign DCs, so if the number of customers at one of the DCs is lower than its defined migration threshold, the arrival is migrated to it and the system state changes to $(x_1\ , x_2$+1). If no suitable DC could be found, the arrival would be lost without a change in the system change.

2. In case a server completes serving an arrival at DC_1 :
   If $x_1 \leq n_1$ then the arrival served at a rate of $x_1\mu_1$ will leave the system and the server would be switched-off as the queue size is zero. If $n_1 < x_1 \leq n_1 + s_1$ the served arrival at a rate of $n_1\mu_1$ will leave the system and the server would be occupied by a request from the head of the queue. In both cases the system state to change to $(x_1 - 1, x_2)$.

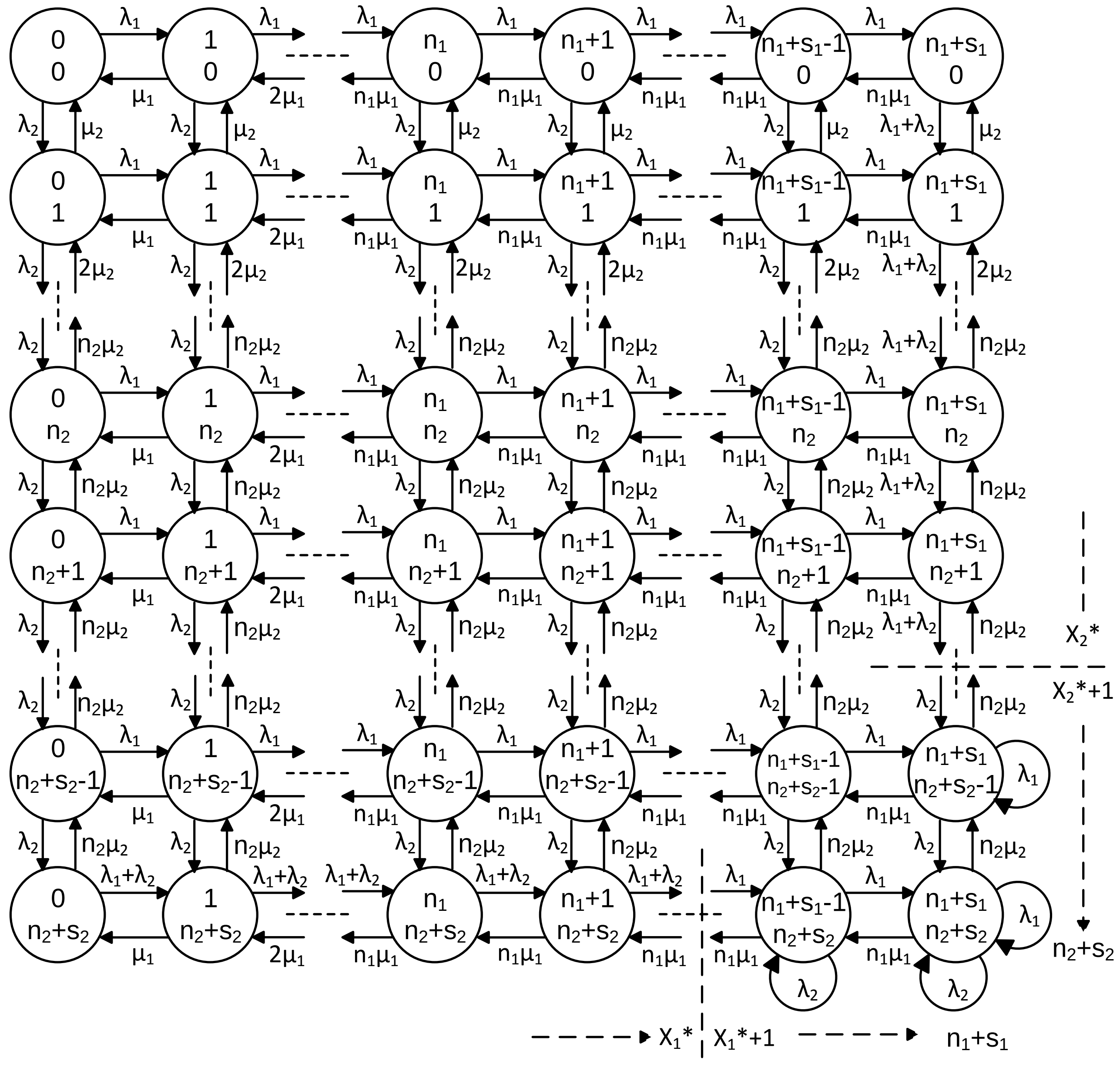


Figure 6-2: State Transition Diagram for two DCs operating under LSSF Algorithm

For solving the steady state probabilities of the Markovian queuing model explained above a system of linear balance equations has to be solved according to the method of Markov Chain analysis (see solution for the service strategy SRTF). In this section an approximate solution based on product-form state distribution for non-border states of the state transition diagram is presented following a step-wise algorithm, as explained below:

- Assume the probability of state (0,0)

$$p(0,0) = 1 \tag{6-5}$$

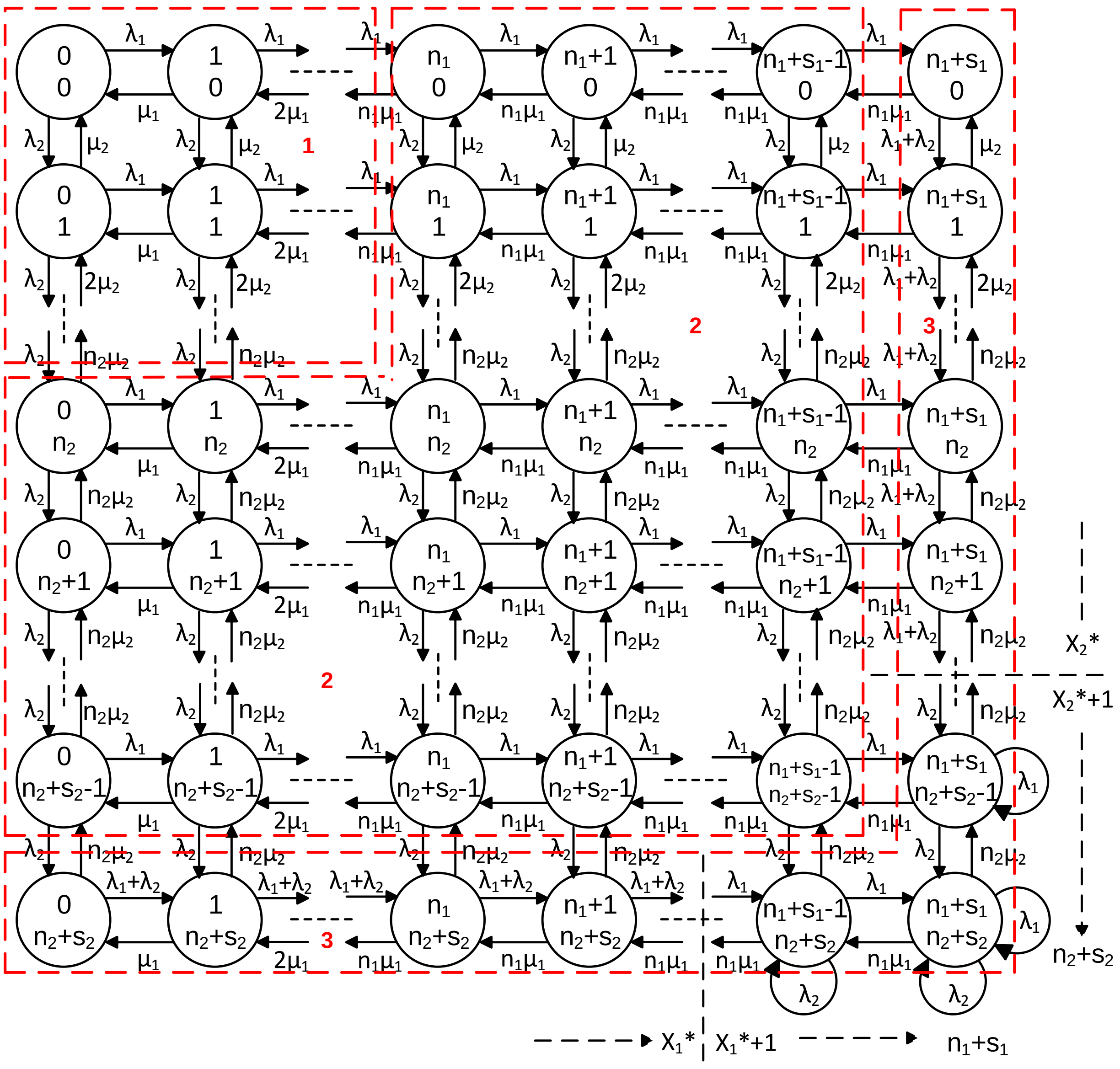


Figure 6-3: Solution Sub-Spaces for LSSF Algorithm

- Consider sub-space 1 shown in Figure 6-3:
  Sub-space 1 includes all states where $x_1 < n_1$ and $x_2 < n_2$. For these specific states, no overflow occurs between the two DCs as not all servers are fully occupied, so any new local arrival to any of the two DCs will be immediately served by its own DC without any need to be migrated to the foreign DC. Accordingly, each state in this sub-space 1 can be solved using the product-form solution in equation 6-6.

$$p(x_1, x_2) = p(0,0) * \frac{\lambda_1^{x_1}}{\mu_1^{x_1} * x_1!} * \frac{\lambda_2^{x_2}}{\mu_2^{x_2} * x_2!}$$

for $x_1 < n_1$ and $x_2 < n_2$

(6-6)

- Consider sub-space 2 shown in Figure 6.3:
  States in sub-space 2 are the states where an arrival to a DC is queued at its own DC. No mutual overflows occur at these states because local DCs are not completely occupied yet, so the two DCs could be treated as independent. These states could be solved also using product form solution illustrated in equation 6-7.

$$p(x_1, x_2) = p(0,0) * \frac{\lambda_1^{n_1}}{\mu_1^{n_1} * n_1!} * \frac{\lambda_2^{n_2}}{\mu_2^{n_2} * n_2!} * \left(\frac{\lambda_1}{\mu_1 n_1}\right)^{x_1 - n_1} * \left(\frac{\lambda_2}{\mu_2 n_2}\right)^{x_2 - n_2}$$

  for $n_1 \leq x_1 < n_1 + s_1 - 1$ and $n_2 \leq x_2 < n_2 + s_2 - 1$

(6-7)

  Subspaces 1 and 2 both rely on product-form solution based on independence of DCs, which is a valid approximation at small load values where the mass of probability lies in these two subspaces. Whereas for heavy load situations and large values of λ_1 and λ_2, the mass of the probability will be shifted to the border states, since DCs tend to be completely occupied and mutual overflows occur between them which makes the product-form solution invalid.

- Consider sub-space 3 shown in Figure 6-3 indicating all border states:
  As product form solution cannot be applied to solve these states due to mutual overflows between DCs, each state has to be solved separately using local balance equation between each state and its neighbor states. At this point in the algorithm, all inner states from sub-spaces 1 and 2 have been calculated relative to state (0,0) using the previously explained product-form solution. Following local balance equilibrium rules, each of the border states is in statistical equilibrium with its lower neighbor states, so all border states can be calculated relative to inner states according to the following equations:

  For lower border states $(0, n_2 + s_2)$, $(1, n_2 + s_2)$, $(2, n_2 + s_2)$,…. $(n_1 + s_1 - 1, n_2 + s_2)$:

  - State $(0, n_2 + s_2)$ can be calculated according to Equation 6-8:

$$p(0, n_2 + s_2) = \frac{\lambda_2}{n_2 \mu_2} p(0, n_2 + s_2 - 1)$$

(6-8)

  - States $(1, n_2 + s_2)$, $(2, n_2 + s_2)$,… $(n_1 - 1, n_2 + s_2)$ are calculated using Equation 6-9:

$$p(x_1, n_2 + s_2) = \frac{1}{x_1 \mu_1 + n_2 \mu_2} \left((\lambda_1 + \lambda_2) p(x_1 - 1, n_2 + s_2) + \lambda_2 p(x_1, n_2 + s_2 - 1)\right)$$

(6-9)

- States $(n_1, n_2 + s_2)$, $(n_1 + 1, n_2 + s_2)$,… $(x_1^*, n_2 + s_2)$ can be calculated using Equation 6-10:

$$p(x_1, n_2 + s_2) = \frac{1}{n_1\mu_1 + n_2\mu_2}\big((\lambda_1 + \lambda_2)p(x_1 - 1, n_2 + s_2) + \lambda_2 p(x_1, n_2 + s_2 - 1)\big) \tag{6-10}$$

- States $(x_1^* + 1, n_2 + s_2)$, $(x_1^* + 2, n_2 + s_2)$,… $(n_1 + s_1 - 1, n_2 + s_2)$ can be calculated using Equation 6-11:

$$p(x_1, n_2 + s_2) = \frac{1}{n_1\mu_1 + n_2\mu_2}\big(\lambda_1 p(x_1 - 1, n_2 + s_2) + \lambda_2 p(x_1, n_2 + s_2 - 1)\big) \tag{6-11}$$

For right border states $(n_1 + s_1, 0)$, $(n_1 + s_1, 1)$, $(n_1 + s_1, 2)$… $(n_1 + s_1, n_2 + s_2 - 1)$:

- State $(n_1 + s_1, 0)$ can be obtained using Equation 6-12:

$$p(n_1 + s_1, 0) = \frac{\lambda_1}{n_1\mu_1} p(n_1 + s_1 - 1, 0) \tag{6-12}$$

- States $(n_1 + s_1, 1)$, $(n_1 + s_1, 2)$, ... $(n_1 + s_1, n_2 - 1)$ can be calculated using Equation 6-13:

$$p(n_1 + s_1, x_2) = \frac{1}{n_1\mu_1 + x_2\mu_2}(\lambda_1 p(n_1 + s_1 - 1, x_2) + (\lambda_1 + \lambda_2)p(n_1 + s_1, x_2 - 1)) \tag{6-13}$$

- States $(n_1 + s_1, n_2)$, $(n_1 + s_1, n_2 + 1)$ … $(n_1 + s_1, x_2^*)$ can be calculated using Equation 6-14:

$$p(n_1 + s_1, x_2) = \frac{1}{n_1\mu_1 + n_2\mu_2}(\lambda_1 p(n_1 + s_1 - 1, x_2) + (\lambda_1 + \lambda_2)p(n_1 + s_1, x_2 - 1)) \tag{6-14}$$

- States $(n_1+s_1, x_2^*+1), (n_1+s_1, x_2^*+2)$ … (n_1+s_1, n_2+s_2-1) can be calculated using Equation 6-15:

$$p(n_1+s_1, x_2) = \frac{1}{n_1\mu_1+n_2\mu_2}\left(\lambda_1 p(n_1+s_1-1, x_2) + \lambda_2 p(n_1+s_1, x_2-1)\right) \tag{6-15}$$

- After calculating lower and right border states; bottom right corner state (n_1+s_1, n_2+s_2) can be calculated using Equation 6-16:

$$p(n_1+s_1, n_2+s_2) = \frac{1}{n_1\mu_1+n_2\mu_2}\left(\lambda_1 p(n_1+s_1-1, n_2+s_2) + \lambda_2 p(n_1+s_1, n_2+s_2-1)\right) \tag{6-16}$$

- After all state probabilities have been calculated relative to the assumption made by Equation 6-5, the assumed state probability $p(0,0)$ can be calculated by obtaining the normalization factor from Equation 6-17:

$$Normalization\, factor = p(0,0) = \frac{1}{\sum_{x_1=0}^{n_1+s_1}\sum_{x_2=0}^{n_2+s_2} p(x_1,x_2)} \tag{6-17}$$

Using Equation 6-17, exact values of remaining state probabilities are calculated by multiplying obtained values by normalization factor.

Having all state probabilities of a system of two DCs with mutual overflow calculated, the most significant performance metrics that allow studying the behavior of DC_i, $i = 1,2$ are derived from the following equations:

- Carried traffic by each DC_i Y_i indicating the average server occupancy at each DC_i

$$Y_1 = \sum_{x_1=0}^{n_1-1}\sum_{x_2=0}^{n_2+s_2} x_1 p(x_1,x_2) + \sum_{x_1=n_1}^{n_1+s_1}\sum_{x_2=0}^{n_2+s_2} n_1 p(x_1,x_2) \tag{6-18}$$

$$Y_2 = \sum_{x_1=0}^{n_1+s_1}\sum_{x_2=0}^{n_2-1} x_2 p(x_1,x_2) + \sum_{x_1=0}^{n_1+s_1}\sum_{x_2=n_2}^{n_2+s_2} n_2 p(x_1,x_2) \tag{6-19}$$

where the average number of active servers is calculated by adding up the multiplication of the probability of each state by the number of active servers during this state.

- Probability of a request being lost due to overload B_i

$$B_1 = \sum_{x_2=x_2^*+1}^{n_2+s_2} p(n_1+s_1, x_2)$$

(6-20)

where an arrival to DC_1 is lost iff DC_1 is completely occupied, and the number of customers at DC_2 exceed the migration threshold. Thus any new arrival at these states cannot be accepted by both DCs and will be lost. Probability of loss at DC_2 follows the same reasoning as shown in equation 6-21. States at which losses occur at either DCs are shown in Figure 6-4.

$$B_2 = \sum_{x_1=x_1^*+1}^{n_1+s_1} p(x_1, n_2+s_2)$$

(6-21)

- Probability of a request being migrated from DC_i to a less loaded DC M_i

$$M_1 = \sum_{x_2=0}^{x_2^*} p(n_1+s_1, x_2)$$

(6-22)

$$M_2 = \sum_{x_1=0}^{x_1^*} p(x_1, n_2+s_2)$$

(6-23)

where an arrival to a fully occupied DC can be migrated to another DC if the number of requests at the foreign DC doesn't exceed its defined migration threshold. States at which migration between DCs occur are indicated in Figure 6-4.

- Mean number of buffered arrivals L_i

$$L_1 = \sum_{x_1=n_1+1}^{n_1+s_1} \sum_{x_2=0}^{n_2+s_2} (x_1-n_1)p(x_1, x_2)$$

(6-24)

$$L_2 = \sum_{x_1=0}^{n_1+s_1} \sum_{x_2=n_2+1}^{n_2+s_2} (x_2-n_2)p(x_1, x_2)$$

(6-25)

where an arrival to a DC will be buffered if all servers were busy at its arrival instance.

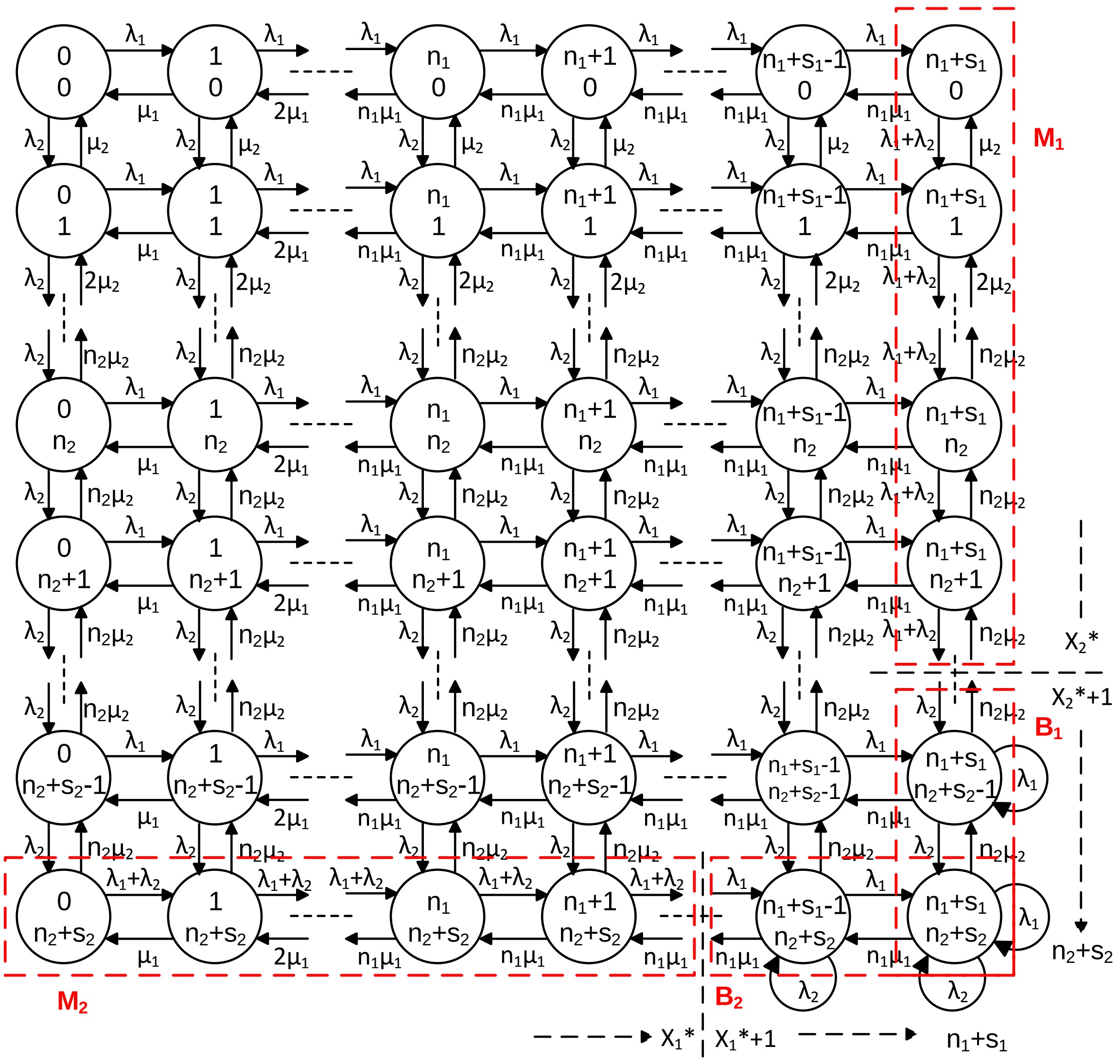


Figure 6-4: Sub-space indicating Loss and Migration Probabilities for DC_1 and DC_2

- Probability that a request is immediately served upon arrival to its own DC I_i

$$I_1 = \sum_{x_1=0}^{n_1-1} \sum_{x_2=0}^{n_2+s_2} p(x_1, x_2)$$

(6-24)

$$I_2 = \sum_{x_1=0}^{n_1-1} \sum_{x_2=0}^{n_2+s_2} p(x_1, x_2)$$

(6-25)

where immediate service only happens if a request arrives to a DC with idle severs. Probabilities of immediate service for DC1 and DC2 are illustrated in Figure 6-5.

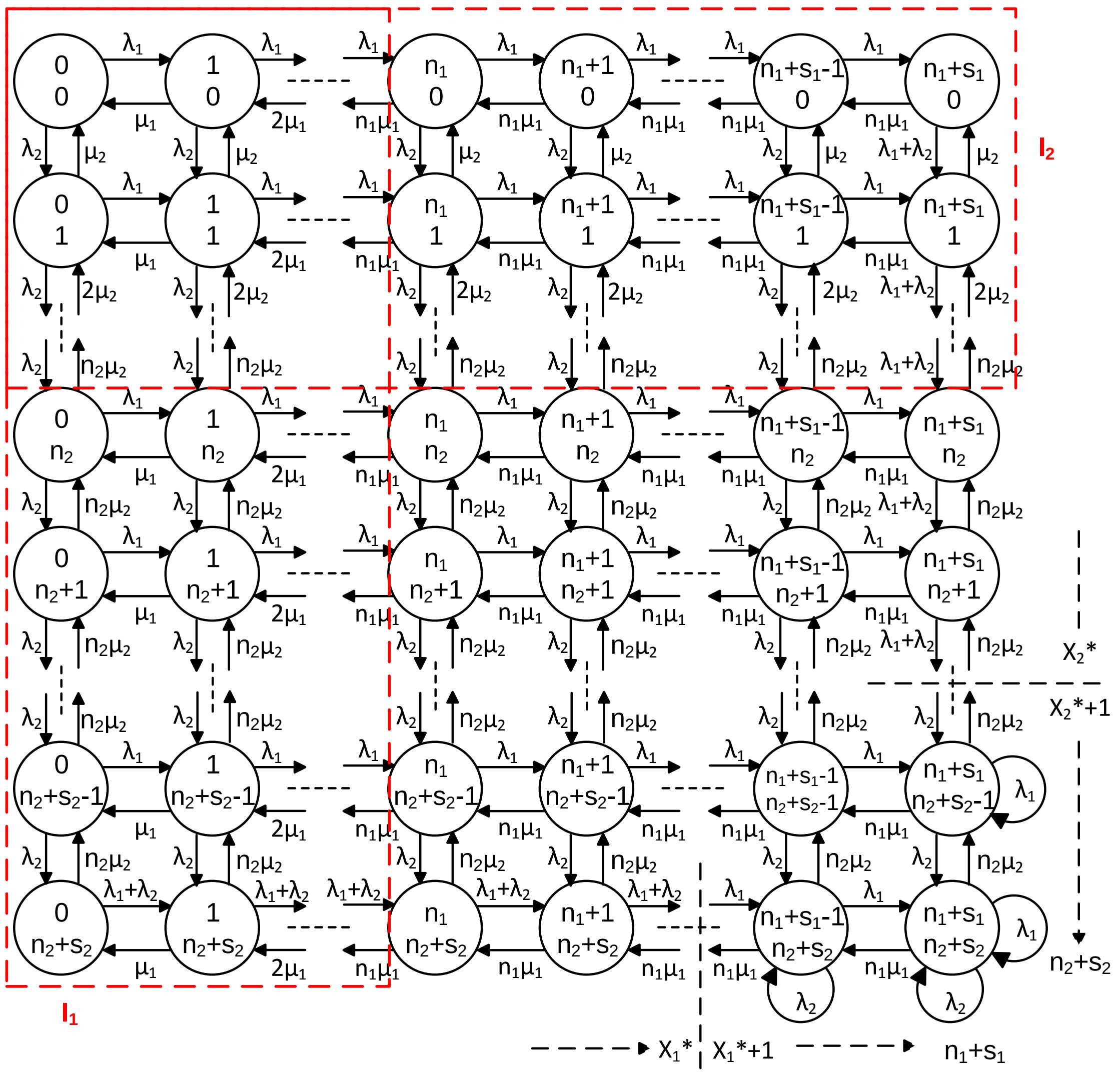


Figure 6-5: Sub-space indicating Probability of Immediate Service for DC$_1$ and DC$_2$

- Probability of a request being delayed upon arrival to its own DC W_i

$$W_1 = \sum_{x_1=n_1}^{n_1+s_1} \sum_{x_2=0}^{n_2+s_2} p(x_1, x_2) - B_1 - \sum_{x_2=0}^{n_2-1} p(n_1+s_1, x_2) \tag{6-26}$$

$$W_2 = \sum_{x_1=0}^{n_1+s_1} \sum_{x_2=n_2}^{n_2+s_2} p(x_1, x_2) - B_2 - \sum_{x_1=0}^{n_1-1} p(x_1, n_2+s_2) \tag{6-27}$$

Equations 6-26 and 6-27 follow from the fact that an arrival to a DC will be buffered if at the instant of its arrival all servers in this DC are busy, and it either finds a place in the queue to be buffered, or migrated to a remote DC where it might also be buffered there if all servers in the remote DC are occupied.

- Mean waiting time of a request w_i

$$w_1 = \sum_{x_1=n_1}^{n_1+s_1-1} \sum_{x_2=0}^{n_2+s_2} p(x_1,x_2) * (x_1 - n_1 + 1) * \frac{h_1}{n_1}$$
$$+ \sum_{x_2=n_2}^{x_2^*} p(n_1+s_1,x_2) * [t_M + (x_2 - n_2 + 1) * h_2/n_2]$$

(6-28)

The mean waiting time of arrivals depends whether an arrival is served at its local DC or migrated to a foreign one. The first part of Equation 6-28 calculates the delay an arrival would experience if it was served at its own DC according to Little's Theorem [80], which is the service time of all arrivals ahead of it in the queue, in addition to one arrival in the server at which it will be served. The second part of Equation 6-28 describes the delay of migrated arrivals, calculated as the summation of service times of all arrivals in the foreign queue, plus service time of one arrival at the server at which it will be served added to them the migration time from local to foreign DC.

- Mean waiting time of a delayed request t_{wi}

$$t_{wi} = w_1/W_1$$

(6-29)

where the mean waiting time of delayed arrivals is calculated by dividing the mean waiting time of all arrivals by the waiting probability.

Results for all above explained performance metrics will be shown in the following Section 6.1.3 and verified against results from simulations and test bed experimentation.

6.1.2.2 Simulation Model

Besides the analytical solution explained in the previous sub-section for mathematically analyzing the LSSF load balancing algorithm a simulation model has also been implemented using OMNeT++. Figure 6-6 shows the NED file for the simulation model which shows two DCs with mutual overflow of requests between them operating under the LSSF strategy. Each DC is composed of simple and compound modules: source, passive queue, servers and a merger for collecting served requests before being disposed at the combined sink module. Each DC has a separate source module for generating arrival requests according to each DC's type of requests, load value and interarrival time distribution. When an arriving request is generated at a source module it has to decide whether the request will be served at its own DC thus exiting via the gate connecting to its own passive queue; or if the local passive queue is full then the request could be migrated to the foreign DC and exits the source module via the output gate connected to foreign passive queue if its size is below the defined migration threshold level. If the local passive queue was full and the foreign passive queue length exceeds the defined threshold then the arrival is considered to be lost and discarded at the source module. Requests are generated at the two source modules at arrival instances following a Poisson distribution by generating negative-exponentially distributed inter-arrival times between requests.

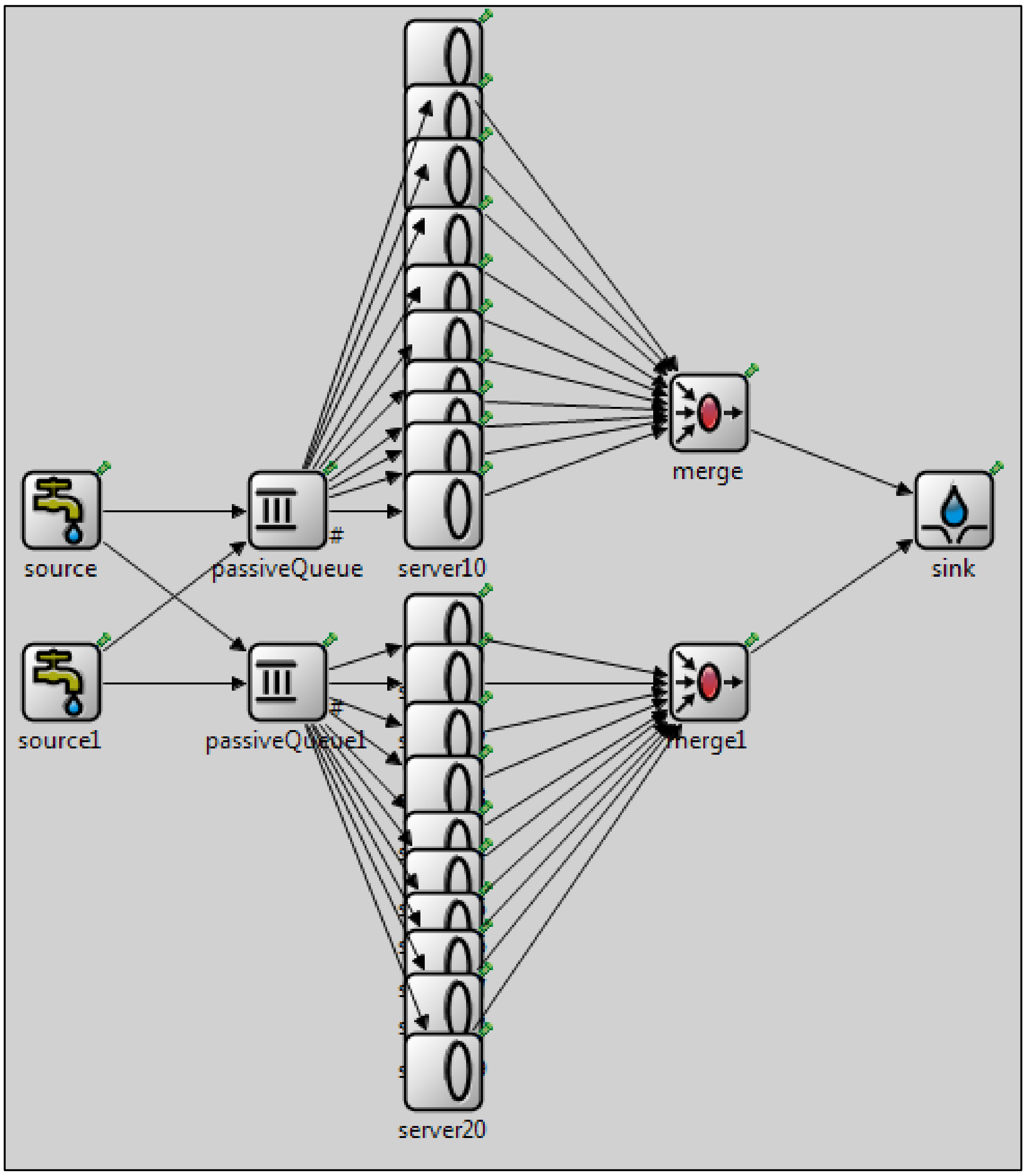


Figure 6-6: Simulation Model for testing LSSF Algorithm between two DCs

The passive queue module forwards arriving requests to idle servers in a FIFO order where idle servers are selected in a round-robin strategy. When a request is forwarded to the passive queue it has to check whether any of the DC's servers are idle so that the request can immediately forwarded to the idle server to start being serviced; or if all servers are busy then the arriving request will be queued. At each server requests' service times are generated according to a negative-exponential distribution; where after service times are elapsed requests are forwarded to the merge module to exit the DC by forwarding them to the sink module. Figure 6-7 shows a flow chart explaining the operation of the LSSF simulation program.

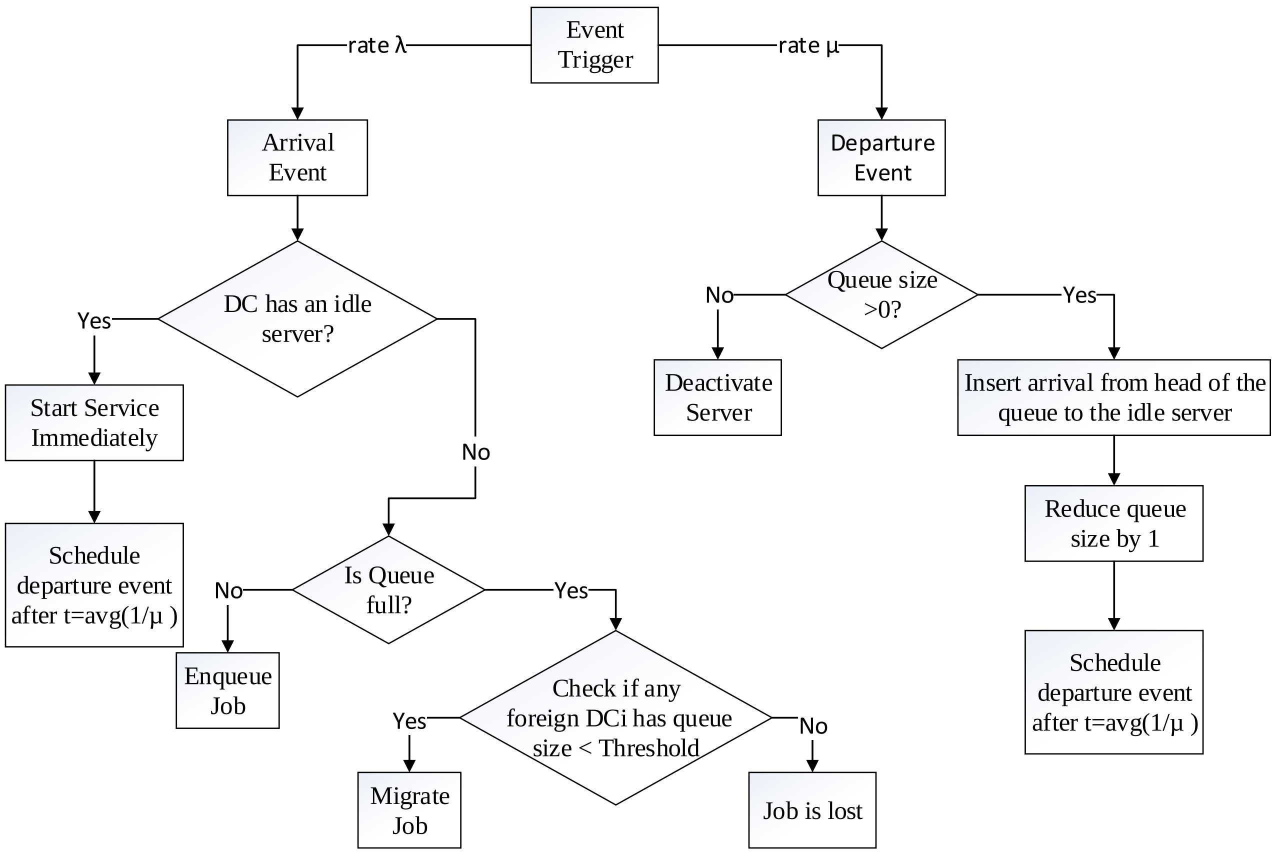

Event Trigger
rate λ
rate μ
Arrival Event
Departure Event
DC has an idle server?
Yes
No
Start Service Immediately
Schedule departure event after t=avg(1/μ)
Is Queue full?
No
Yes
Enqueue Job
Check if any foreign DCi has queue size < Threshold
Yes
No
Migrate Job
Job is lost
Queue size >0?
No
Yes
Deactivate Server
Insert arrival from head of the queue to the idle server
Reduce queue size by 1
Schedule departure event after t=avg(1/μ)

Figure 6-7 Flow Chart for OMNeT ++ Simulation Model for LSSF Load Balancing Algorithm

Significant performance metrics of DCs operating under LSSF strategy such as loss, migration and delay probabilities as well as mean delay of delayed requests are calculated according to the equations below for each DC:

- Loss Probability B

$$B = \frac{Number\ of\ lost\ requests}{Total\ number\ of\ arriving\ requests} \tag{6-30}$$

- Migration Probability M

$$M = \frac{Number\ of\ migrated\ requests}{Total\ number\ of\ arriving\ requests} \tag{6-31}$$

- Delay Probability W

$$W = \frac{Number\ of\ delayed\ arriving\ requests}{Total\ number\ of\ arriving\ requests} \tag{6-32}$$

where

$$\begin{aligned} &Number\ of\ delayed\ arriving\ requests \\ &\quad = Number\ of\ delayed\ arrivals\ in\ local\ DC \\ &\quad + Number\ of\ migrated\ and\ queued\ arrivals \end{aligned}$$

- Mean Delay of Delayed Frames $E[T_W \mid T_W > 0]$

$$E[T_W \mid T_W > 0] = \frac{\sum Delays\ of\ delayed\ arriving\ requests}{Total\ number\ of\ delayed\ arriving\ requests} \tag{6-33}$$

where

$$\begin{aligned} &\sum Delays\ of\ delayed\ arriving\ requests \\ &\quad = \sum Delays\ of\ delayed\ arriving\ requests\ in\ local\ DC \\ &\quad + \sum Delays\ of\ delayed\ migrated\ requests\ \ in\ foreign\ DC \end{aligned}$$

Due to the increased flexibility of the simulation at testing different parameters of the algorithm that cannot be altered in the analytical solution, various distributions for inter-arrival, service and activation time have been tested to eliminate the dependability of the algorithm on Markovian assumptions. All simulation results presented in the upcoming results section are based upon data gathered from 20 simulation runs and presented with a 95% confidence interval.

6.1.2.3 Experimental Setup

For testing its effectiveness in real-time environments, LSSF algorithm was tested using the cloud DC test-bed explained previously in sub-Section 4.3. Each of the physical servers running ESXi hypervisor emulates a DC with 10 servers in the proposed algorithm, where each one of the physical servers can host and serve up to 10 VMs simultaneously. Configuration scripts specifying the algorithm steps were written and tested first using Windows' PowerShell to troubleshoot any errors during script execution; afterwards scripts were imported to PowerCLI for automatic deployment on servers. Scripts were configured to run for the duration required to create and serve 100,000 requests; attempts for producing results for higher number of requests required more memory space and time than possible and resulted in running errors. Scripts were supplied with data arrays for interarrival and service times for each arriving request, which were generated by Matlab according to specified random number distributions. During scripts' execution data is gathered in order to evaluate model's performance, such as number of delayed requests and delay time of each, number of migrated arriving requests and number of lost arriving requests in order to calculate the performance metrics in the same manner as specified in the previous simulation section. As the placement of the VM whether to be hosted at its local DC or migrated to a foreign DC is performed upon its arrival, vSphere and vMotion allow for deploying the virtual machine on its local server system or migrate it to the other server, respectively. Scripts are set to create and deploy 100,000 VM requests where throughout the experiments statistics are collected such as number of delayed/ migrated VMs and average delay of a VM request.

6.1.3 Results

This section shows numerical results for the most significant performance metrics by which the system performance can be depicted and service level agreements of users are determined; such as average delays, loss, migration and delay probabilities. The algorithm is tested between two data centers under different load values to indicate how the algorithm performs at low, average and high load situations. Figures below show the results for each of the defined performance metrics calculated at a DC implementing the LSSF algorithm vs. DC without load balancing to report the enhancement done by algorithm. Results are also shown for cases with/without migration overhead where queue thresholds are adjusted accordingly to maintain users' SLAs. For the test cases shown below both DCs are allocated the same parameters in terms of number of homogeneous servers, queue size and load value per server, thus the values for performance metrics calculated at any of them and presented below will be equivalent.

Test cases in this section show results for two identical DCs each having $n = 10$ homogeneous servers where each server has an average service rate $\mu = 1$ arrival /second, and all servers in the DC have a common buffer place of size $s = 30$. System's load value is varied between 0 till 1.2 per server to show how the algorithm will affect the system performance at low-load / overload situations. Delay limit defined by the users' SLAs is set to $t_{WT} = 3$, which indicates that a user served at its own DC will always experience a delay that is at most equal to that defined by SLA even if it occupied the last available buffer space upon its arrival. In case the arrival had to migrate to a foreign DC, delay limit also defines that the waiting time at the foreign DC along with the migration overhead to it should not exceed the defined SLA value. The two test cases illustrated in this section show cases for different values of migration time. First is $t_M = 0$ indicating cases where migration time is small or negligible, typically in cases of migration between server groups within the same DC or between nearby DCs. Second case with $t_M = 2$ indicating relatively long migration times where transmission time between DCs require more time relative to the average service time of requests.

Case 1: Two identical DCs with $n_i = 10$, $s_i = 30$, $t_M = 0$ and defined threshold migration x_i^*=39.

Case 2: Two identical DCs with $n_i = 10$, $s_i = 30$, $t_M = 2$ and defined threshold migration x_i^*=19.

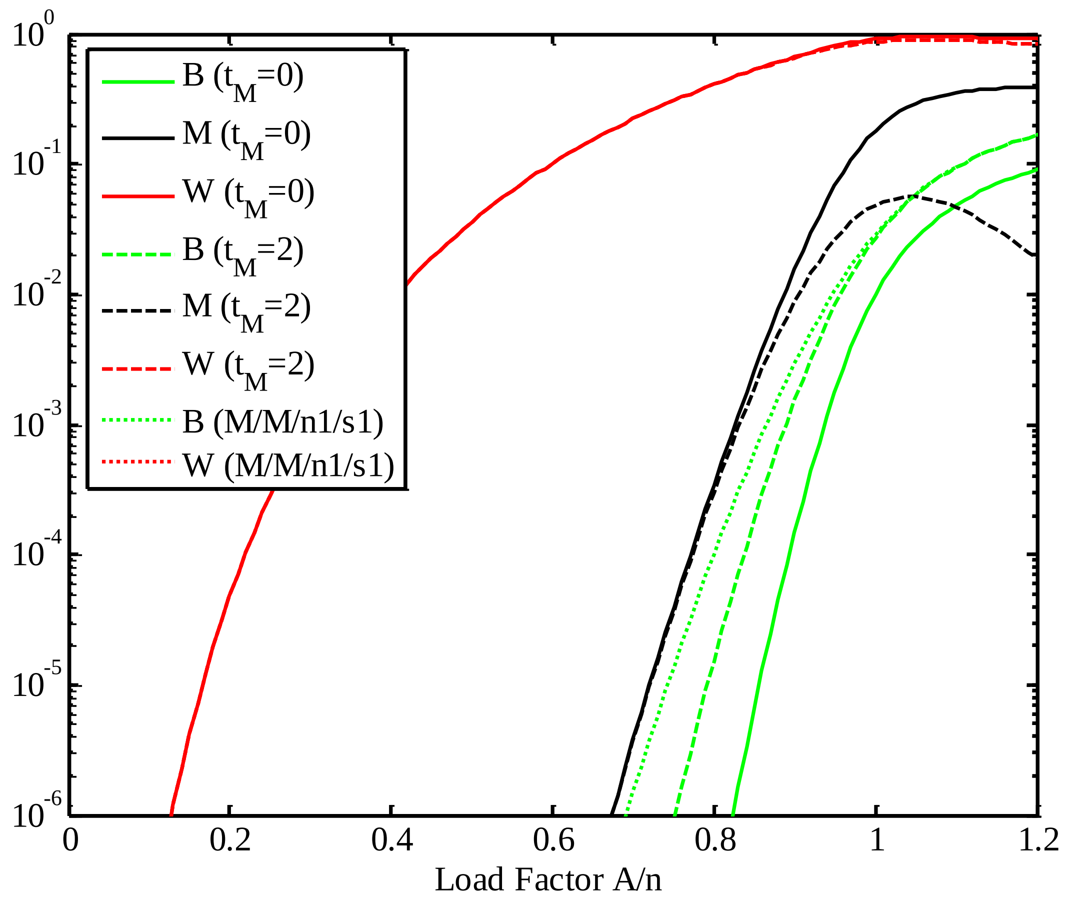


Figure 6-8 Loss (B), Migration (M) and Delay (W) Probabilities vs. Load (A/n) for LSSF Algorithm

Test case 1 of the LSSF algorithm sets the migration threshold at each DC to x_i^*=39, which allows the DC to accept foreign arrivals until only one buffer place is empty, this remaining place can only be occupied by a local arrival so any foreign arrivals will be lost. Whereas case 2 sets the migration threshold to x_i^*=19, allowing less buffer places at the DC to be occupied by foreign arrivals to compensate the delay due to migration overhead. The selection of parameters for the test cases is reasoned by the SLAs delay limit. For case 1, the threshold is high to allow for more migrations since there is no migration overhead, and the only delay an arrival experiences is due to buffering at the foreign DC. While for case 2, the threshold is reduced so that the delay of a foreign arrival buffered at the DC along with the migration overhead will still fulfill the defined SLA. Figure 6-8 shows loss, migration and waiting probabilities for any of the DCs versus system load, where the results for both DCs are identical as they have identical system parameters. Results are shown for the two cases with/without migration overhead and for a DC without the load balancing algorithm.

As shown in Figure 6-8 the waiting probability of an arrival is not affected for a DC with/without LSSF algorithm, neither affected by the migration overhead, as the waiting probability is related to the system capacity which is not affected by the algorithm. However, the algorithm reduces the loss probability B due to the fact that an arrival that doesn’t find a place at its local DC can migrate to another DC if load balancing is implemented, whereas without load balancing it would have been lost. Loss probability is lower for the case of negligible migration overhead as the threshold is higher and more arrivals can be accepted. For the case of non-negligible migration overhead, the algorithm will still perform better for the cases of high load (A/n=0.9), but as the load increases to overload situations and buffers get filled up, threshold is easily reached thus loss probability increases to be similar to a DC without load balancing algorithm. Finally, Figure 6-8 shows that the migration probability for both test cases has the same value at average load situations as long as migration threshold is not reached. Once the DC experiences an overload situation and threshold is reached, case 2 with lower threshold will have higher number of dropped arrivals and thus migration probability drops.

Figure 6-9 shows the average delay calculated for delayed arrivals only. Average delays are kept bounded below the defined SLA by choice of system parameters. The figure shows that the application of LSSF load balancing algorithm does not affect the delay of those arrivals who wait. Slight variations only occurs at overload situations because for test case 2 of the algorithm, less migrations occur and thus more arrivals are lost, which accordingly reduces the average response time. Compared to a DC without load balancing the mean delay value depends on migration overhead; where for zero overhead the delay is slightly larger at overload situations than without load balancing as more arrivals are migrated and delayed at foreign DCs. As migration overhead increases mean delay value is reduced to match that of a system without load balancing.

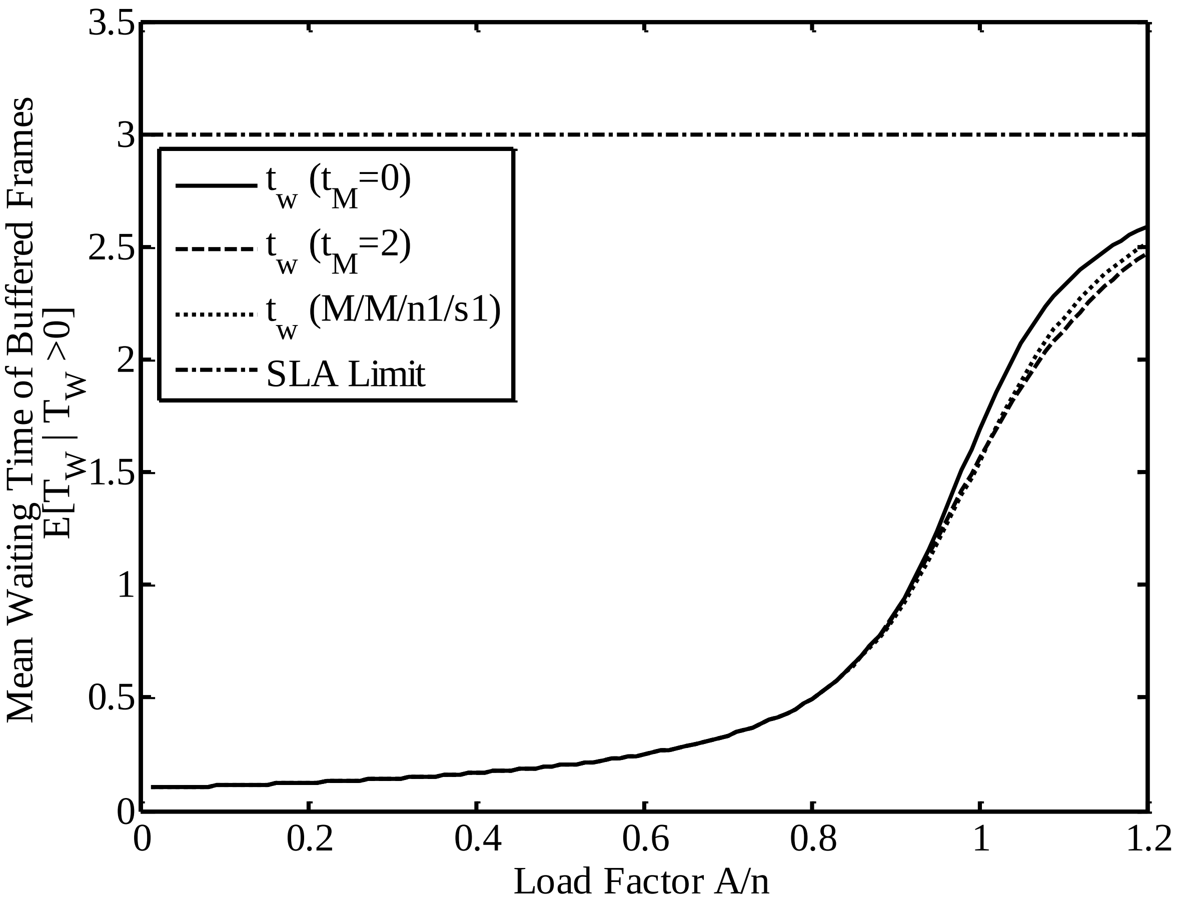


Figure 6-9 Mean Waiting Time of Delayed Requests versus Load (A/n) for LSSF Algorithm

Besides performance metrics obtained from analytical solution provided for the LSSF algorithm, the model was also implemented using an OMNeT++ simulation model between two DCs and on a the real DC test-bed for proving its effectiveness. Figures 6-10 and 6-11 show results for significant performance metrics obtained from the simulation and test-bed compared to those from analytical results. Figure 6-10 shows loss, delay and migration probabilities for two DCs operating by LSSF algorithm with the following parameters at all platforms: $n = 10, s = 30$ and $t_m = 0$ with 95% confidence intervals for simulation and test-bed results. In figure 6-10 results from different platforms have only minor differences as shown in the figure, where analytical solution always provides an upper bound for probabilities. Minor differences in calculated metrics result from factors in simulation and test-bed that cannot be accounted-for in the analytical solution; such as when an arrival is migrated to a foreign data center where the status of the foreign data center is very likely to change during the migration time. For example, a migrated arrival that is supposed to migrate to a foreign DC with a small queue size could find the queue empty during the elapsed migration time and thus gets immediately served. Another difference between analytical solution and other platforms is its accuracy in calculating very small numbers at all load values. This effect is shown in Figure 6-11 where mean delay of delayed frames has a zero value at low delays values despite having a non-zero value reported by the analytical solution. This effect occurs due to the fact that the simulation and test-bed experiments run for a defined number of users, and for obtaining minor effects such as delays a very low loads the number of observed arrivals need to be extremely high for the results to match those reported analytically. These factors beside the different random number generators used for generating inter-arrival and service times tend to cause a variation between results obtained from different platforms.

Another test-case obtained through test-bed experimentation is by testing the LSSF algorithm on four DCs instead of only two to prove its scalability across any number of DCs. Parameters for the test case included 4 servers each can handle 10 VMs simultaneously to match a DC with 10 servers, 30 buffer places and migration threshold of 29 queued arrivals without overhead. Figures 6-12 and 6-13 show the calculated performance probabilities and average delays where results show minor differences between the case of only two DCs. These minor differences in probabilities show lower loss probability and higher migration probability for the case with 4 DCs. This is explained by the economy of scale effect; as the number of DCs increase the probability of an arrival that would have been lost at its local DC to be accepted at another foreign DC increases, thus decreasing number of lost arriving requests and increasing number of migrated ones. All previously reported results shows that LSSF algorithm is effective at reducing loss probabilities of arrivals while maintaining SLAs by cooperation between DCs to server arrivals that cannot be served at their own DCs, and is scalable to any number of DCs

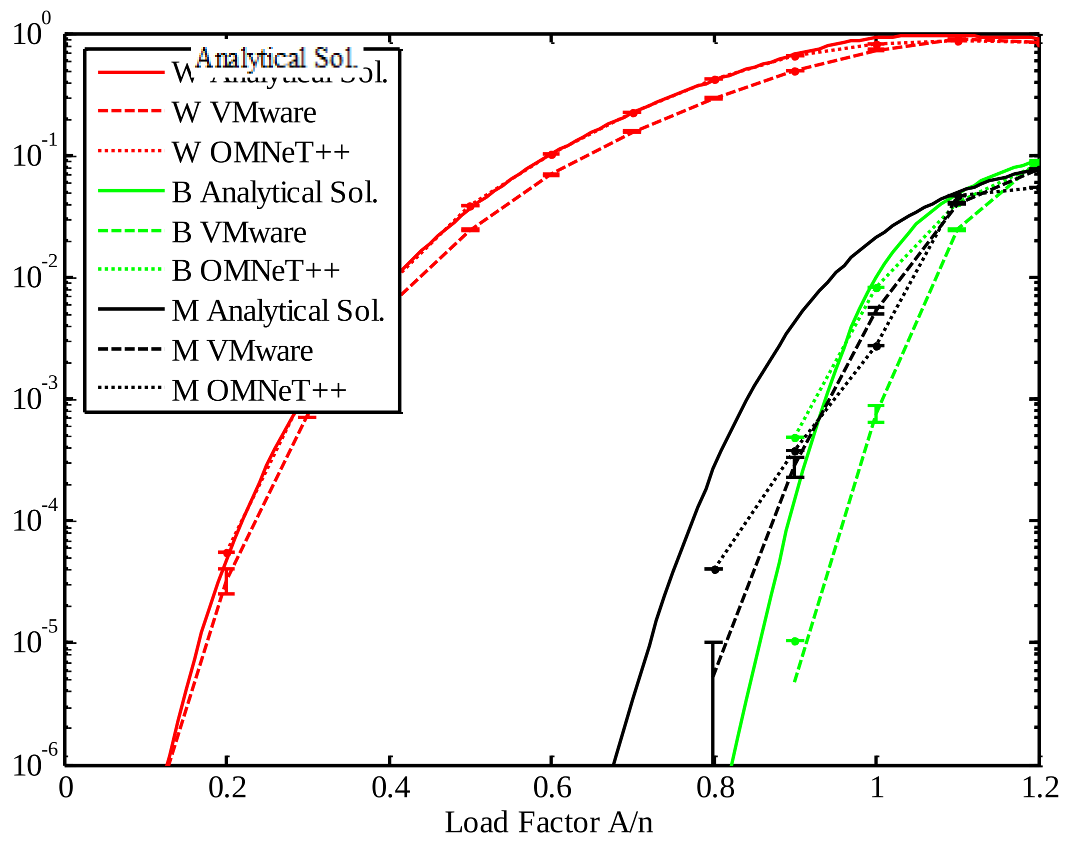


Figure 6-10 Loss (B), Migration (M) and Delay (W) Probabilities vs. Load (A/n) for LSSF Algorithm without overhead implemented on various Platforms

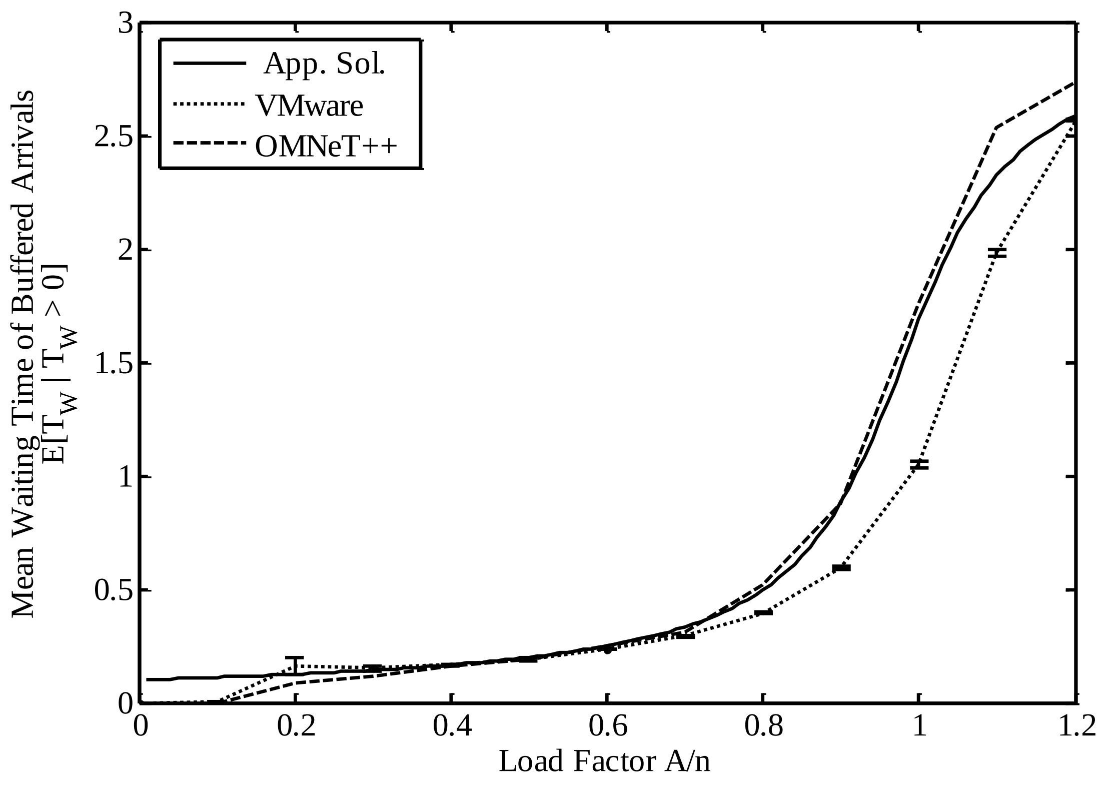


Figure 6-11 Mean Waiting Time of Delayed Requests versus Load (A/n) for LSSF Algorithm without overhead implemented on various Platforms

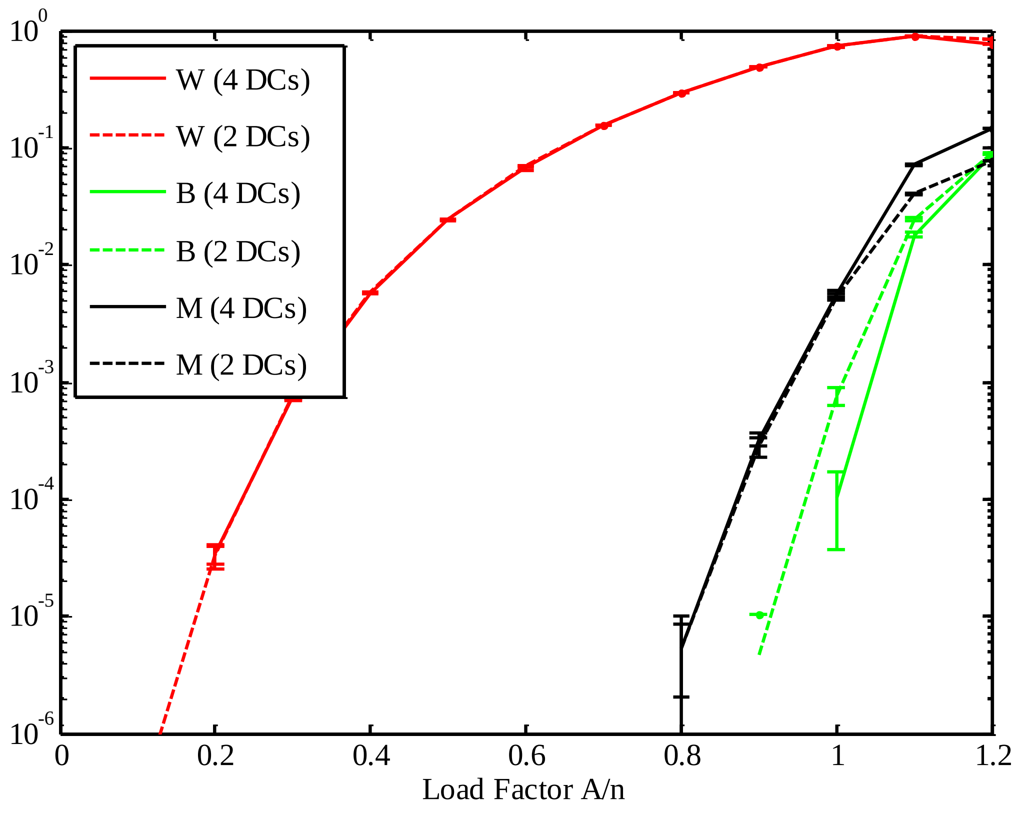


Figure 6-12 Loss (B), Migration (M) and Delay (W) Probabilities vs. Load (A/n) for LSSF Algorithm without overhead implemented on DC Test-bed

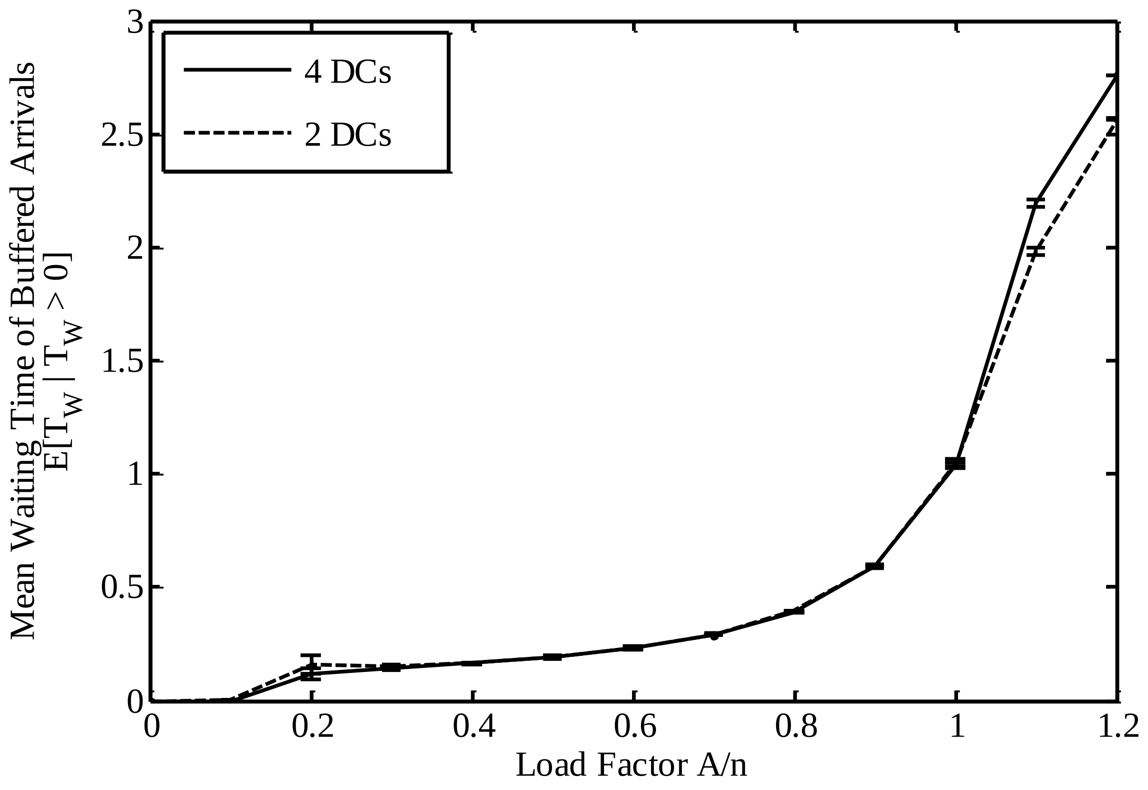


Figure 6-13 Mean Waiting Time of Delayed Requests versus Load (A/n) for LSSF Algorithm without overhead implemented on DC Test-bed

6.2 Algorithm 2: Shortest Response Time First (SRTF)

Shortest Response Time First algorithm ‘SRTF’ is a generalized upgrade of one of the already existing algorithms in literature explained in Chapter 3, the Join Shortest Queue algorithm. Upon a request’s arrival, Join Shortest Queue ‘JSQ’ algorithm decides which DC the arrival will be routed based on the queue size of each DC. Arrivals are routed to the shortest queue assuming that this selection criterion will guarantee a shorter waiting time for the arrival request than if it was served at its local DC. However, this criterion does not account for the service rates of servers at each DC, for example; joining a longer queue at a DC with high service rates might guarantee shorter waiting time than joining a shorter queue at a DC with low service rates. Thus, JSQ algorithm will only achieve minimal delays for requests only in case of homogeneous DCs with servers having homogeneous service rates. Shortest Response Time algorithm ‘SRTF’ generalizes this special case and allows for maintaining SLAs of users in terms of delay by proposing a criterion for managing the load between heterogeneous DCs with heterogeneous service rates. SRTF estimates the average delay time that an arriving request will experience at each of the available DCs, then routes it to the DC offering the least average waiting time. This approach allows for more accurate load balancing decisions and guarantees lowest average waiting time for users as it accounts for users’ SLAs while considering the current state of each DC. As the number of DCs and their servers increase along with their buffering capacity, economy-of-scale effect results in better performance for SRTF algorithm in terms of shorter average delays and lower loss probabilities.

6.2.1 Model Definition

This section explains the SRTF algorithm by applying it on two DCs modeled as two queuing systems with mutual overflow of requests between them. Similar to the previously explained LSSF algorithm in Section 6.1.1, the basic model consists of two identical DCs, each having n_i homogeneous servers and equal buffer space of length s_i. Each server is assumed to have a service rate of μ_i, and arrivals to each DC occur with an average rate of λ_i. The basic model is illustrated in Figure 6-10 using the same notations explained previously in Section 6.1.1 and will not be explained again to avoid repetition. As the figure shows, arrivals to any of the DC arrive to a gateway node first before being routed to the DC where they will be served as chosen by the algorithm. Assignment strategy at this gateway node is done according to the following rules:

- If an arrival finds an empty server at its local DC, it will be routed to it for being served immediately
- If no idle servers are available at the local DC, the arrival will be routed to the foreign DC if it has any idle servers where it will also be immediately served.
- If all servers at both DCs are busy, then the average waiting time that the arrival would experience if it was routed to each of the DCs is calculated using the following equation, for $i = 1,2$, given that the queue at any of the DCs is not full :

$$t_{wi} = \frac{x_i - n_i + 1}{n_i \mu_i}$$

(6-30)

where the expected waiting time of an arrival is the summation of service times of all arrivals ahead of it in the FIFO queue, plus one arrival being served at the queue at which it will be served at in the future.

1. After calculating the expected waiting times at both DCs, the arrival will be routed to the DC offering minimal expected average waiting time according to the following condition in equation 6-31:

$$\min\left\{t_{w1} = \frac{x_1 - n_1 + 1}{n_1 \mu_1}, t_{w2} = \frac{x_2 - n_2 + 1}{n_2 \mu_2}\right\}$$

(6-31)

2. If queues at both DCs are full, then the arrival will be lost.

According to the strategy explained above, SRTF algorithm provides the minimal delay an arrival could experience as it explores the current status of all available DCs to which the arrival can be migrated. However, this comes with an overhead due to the decision that has to be taken at the instant of arrival of each request before it is routed to the most suitable DC.

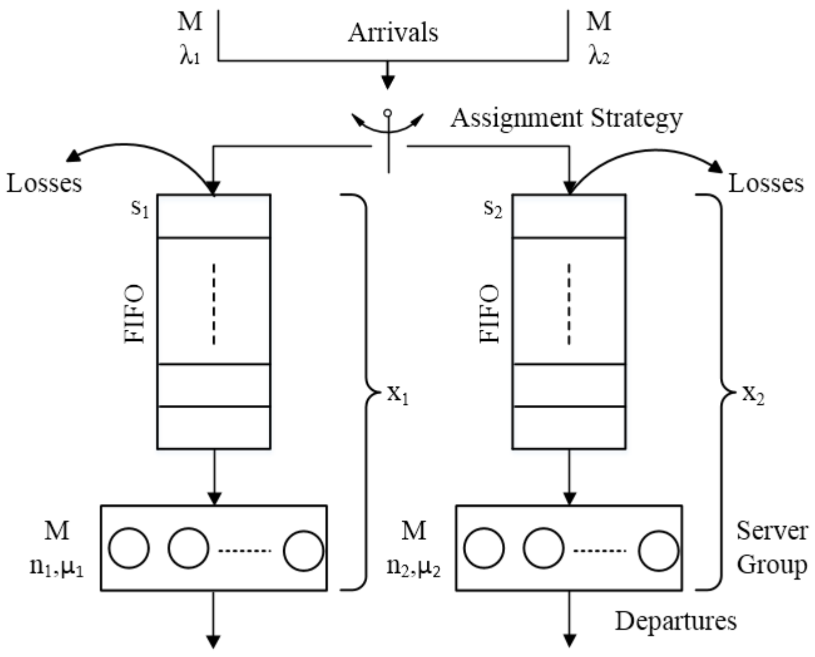


Figure 6-14 System Model for two Data Centers under SRTF Algorithm

6.2.2 Model Analysis

This section provides analysis for the proposed SRTF algorithm using different approaches. First the state transition diagram of the Markov Chain for two DCs operating under the algorithm is introduced and explained followed by an approach for solving steady state probabilities of the system numerically using Gauss-Seidel Method for the linear equations in equilibrium states. Mathematical analysis of the algorithm under Markovian assumption is verified in following sub-sections by OMNeT++ simulations as well as test-bed experimentation that test the model under more general traffic conditions and extended system parameters.

6.2.2.1 Mathematical Analysis

For evaluation of model's performance this section shows the model of two DCs operating under SRTF strategy represented by a two-dimensional Markov Chain. Figure 6-15 shows the state transition diagram indicating the states of both DCs and transitions between states at the instances of a request's arrival or end of a request's service time. Figure below shows the special case of two homogeneous DCs with identical number of servers $n_1 = n_2$, equal buffer places $s_1 = s_2$ and identical service rates for all servers μ. This selection of parameters makes the system similar to a single DC with $2n$ servers and $2s$ buffer places in a FIFO queue for an approximate solution. This economy of scale advantage provided by the SRTF algorithm allows an arrival to experience the lowest possible average waiting time, as it has the advantage of twice the number of available servers and twice the buffer space.

In this two-dimensional Markov chain shown below events are either arrival events to either one of the two DCs or a departure event from one of the servers at the two DCs. Changes to the current general state (x_1, x_2) occur according to the following conditions:

1. In case of an arrival event occurs:
   1.1. If $0 \le x_1 < n_1$ and $0 \le x_2 < n_2$ then an arrival at a rate of λ_1 will be served at DC1 changing system state into $(x_1 + 1, x_2)$ and an arrival at a rate of λ_2 will be served at DC2 changing system state into $(x_1, x_2 + 1)$. Both arrivals will be immediately served by idle servers at their local DC.
   1.2. If $0 \le x_1 < n_1$ and $x_2 \ge n_2$ then an arrival at a rate of λ_1 or $\lambda_2$will be routed to DC1 where it will be served at one of the idle servers in DC1, thus system state changes to $(x_1 + 1, x_2)$
   1.3. If $x_1 \ge n_1$ and $0 \le x_2 < n_2$ then an arrival at a rate of λ_1 or $\lambda_2$will be routed to DC2 where it will be served at one of the idle servers in DC2, thus system state changes to $(x_1, x_2 + 1)$
   1.4. If $x_1 \ge n_1$ and $x_2 \ge n_2$ then the assignment strategy for an arrival will depend on the current system state in terms of queue size and service rate. Figure 6-15 indicates transition rates to states where the arrival will be served at DC_1 or DC_2 by functions $\lambda(Q_1)$ and $\lambda(Q_2)$ respectively. These functions are determined as follows:

$$\lambda(Q_1) = \begin{cases} \lambda_1 + \lambda_2 & if\ (x_1 - n_1 + 1)/n_1\mu_1 < (x_2 - n_2 + 1)/n_2\mu_2 \\ \lambda_1 & if\ (x_1 - n_1 + 1)/n_1\mu_1 = (x_2 - n_2 + 1)/n_2\mu_2 \\ 0 & if\ (x_1 - n_1 + 1)/n_1\mu_1 > (x_2 - n_2 + 1)/n_2\mu_2 \end{cases}$$

(6-32)

$$\lambda(Q_2) = \begin{cases} \lambda_1 + \lambda_2 & if\ (x_1 - n_1 + 1)/n_1\mu_1 > (x_2 - n_2 + 1)/n_2\mu_2 \\ \lambda_2 & if\ (x_1 - n_1 + 1)/n_1\mu_1 = (x_2 - n_2 + 1)/n_2\mu_2 \\ 0 & if\ (x_1 - n_1 + 1)/n_1\mu_1 < (x_2 - n_2 + 1)/n_2\mu_2 \end{cases}$$

(6-33)

where decisions calculated by equations 6-32 and 6-33 guarantee routing of arriving requests to the DC whose queue offers shortest average waiting time. If both queues offer equal waiting times, then an arrival would be served at its local DC to avoid further delays resulting from migration overhead.

1.5. Finally, if $x_1 = n_1 + s_1$ and $x_2 = n_2 + s_2$, then all arrivals will be lost without a change to system state

2. In case of a departure event; current state (x_1, x_2) will change as follows:
   2.1. If departure occurs at a DC where $x_i \leq n_i$, the server will be switched-off as there are no more arrivals to serve. System state changes by reducing x_i element by 1.
   2.2. If departure occurs at a DC where $x_i > n_i$, then system state changes also by reducing x_i element by 1, where a request from the head of the queue will replace the departed arrival at the server.

For solving the state probabilities of the Markovian model explained above no closed-form solution exists nor could a recursive one be deduced similar to LSSF algorithm. To overcome this problem, Forward Kolmogorov balance equations were obtained for each state in the system, then the formed system of linear equations is solved numerically using Gauss-Seidel method with successive over-relaxation. Gauss-Seidel method starts by assuming an initial solution for the value of state probabilities, and through successive iterations the error between the assumed value and the correct value 'defect' is corrected stepwise with each iteration cycle. The algorithm is applied through the following steps illustrated by Figure 6-16:

1. All states are assumed to have an initial state probability, e.g., probabilities equal to the reciprocal of the number of states.
2. The following steps are performed at each state:
   2.1. Knowing the balance equation, a defect value resulting from the assumption of equal probabilities for all states is calculated.
   2.2. Probability of the state is corrected by subtracting the term $\frac{d_i}{a_i} * REL$, where d_i is the defect value, a_i is the coefficient of the probability of state i defined as the aggregated output rates from state i and REL is the relaxation factor.
   2.3. Defect values calculated from all states are added into term SD
3. Having calculated one round (iterative cycle) of all stages of step 2, cumulative defect value SD is checked against a defined threshold ε:
   3.1. If SD ≤ ε
   Algorithm is stopped and calculated steady state probabilities are normalized by dividing the value of each state by the summation of all state probabilities.
   3.2. If SD > ε
   Step 2 is repeated for as many rounds until the defect value decreases below the defined threshold, where the stopping condition is re-checked at the end of each round.

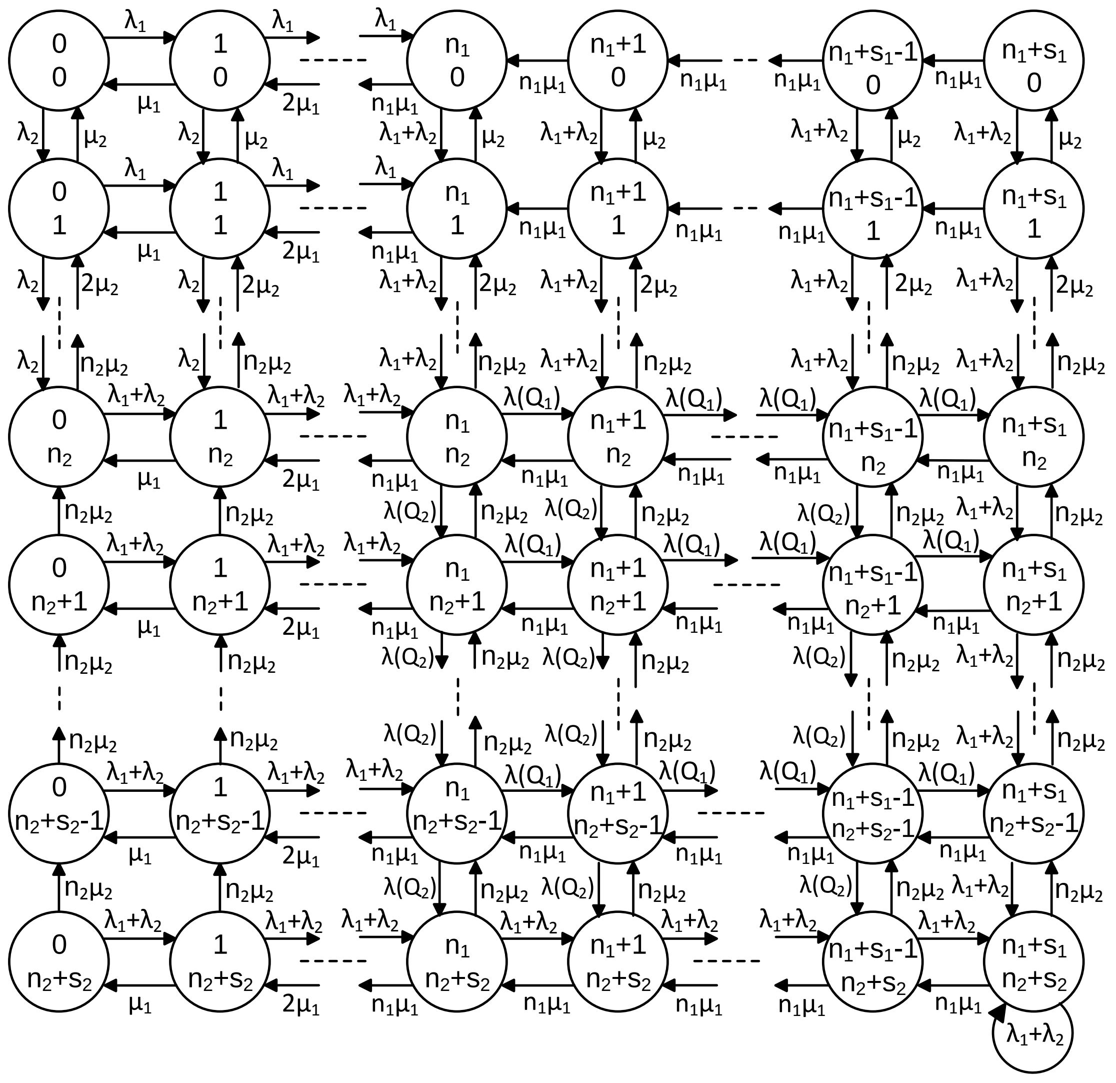


Figure 6-15 State Transition Diagram for two DCs operating under SRTF Algorithm

Defect value is calculated at each state using balance equations obtained according to the following equations:

- For first row states indicated by sub-space 1 in Figure 6-17:
  - For state (0,0)

    Balance equation is

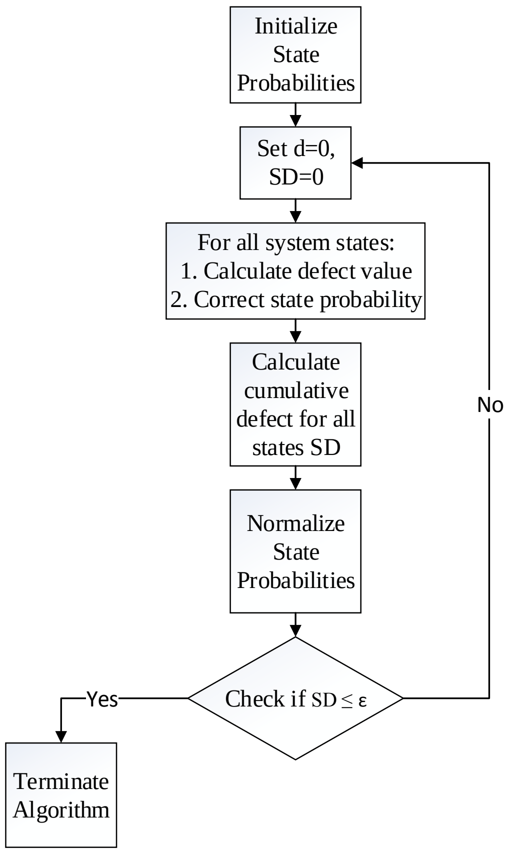
$$(\lambda_1 + \lambda_2)p(0,0) = \mu_1 p(1,0) + \mu_2 p(0,1) \tag{6-34}$$

    so defect value can be calculated as

$$d = (\lambda_1 + \lambda_2)p(0,0) - \mu_1 p(1,0) - \mu_2 p(0,1) \tag{6-35}$$

    and the probability of state can be corrected as described in step 2.2 using equation 6-36:

$$p(0,0) = p(0,0) - d * REL/(\lambda_1 + \lambda_2) \tag{6-36}$$



Initialize State Probabilities
Set d=0, SD=0
For all system states:
1. Calculate defect value
2. Correct state probability
Calculate cumulative defect for all states SD
Normalize State Probabilities
Check if SD ≤ ε
Yes
No
Terminate Algorithm

Figure 6-16 Flow Chart for Gauss-Seidel Algorithm

- For states $(1,0)$ till $(n_1 - 1,0)$

$$d = (\lambda_1 + \lambda_2 + x_1\mu_1)p(x_1, 0) - \lambda_1 p(x_1 - 1,0) - (x_1 + 1) * \mu_1 p(x_1 + 1,0) - \mu_2 p(x_1, 1) \tag{6-37}$$

- For state $(n_1, 0)$

$$d = (\lambda_1 + \lambda_2 + n_1\mu_1)p(n_1, 0) - \lambda_1 p(n_1 - 1,0) - n_1\mu_1 p(n_1 + 1,0) - \mu_2 p(n_1, 1) \tag{6-38}$$

- For states $(n_1 + 1,0)$ till $(n_1 + s_1 - 1,0)$

$$d = (\lambda_1 + \lambda_2 + n_1\mu_1)p(x_1, 0) - n_1 * \mu_1 p(x_1 + 1,0) - \mu_2 p(x_1, 1) \tag{6-39}$$

- For state $(n_1 + s_1, 0)$

$$d = (\lambda_1 + \lambda_2 + n_1\mu_1)p(n_1 + s_1, 0) - \mu_2 p(n_1 + s_1, 1) \tag{6-40}$$

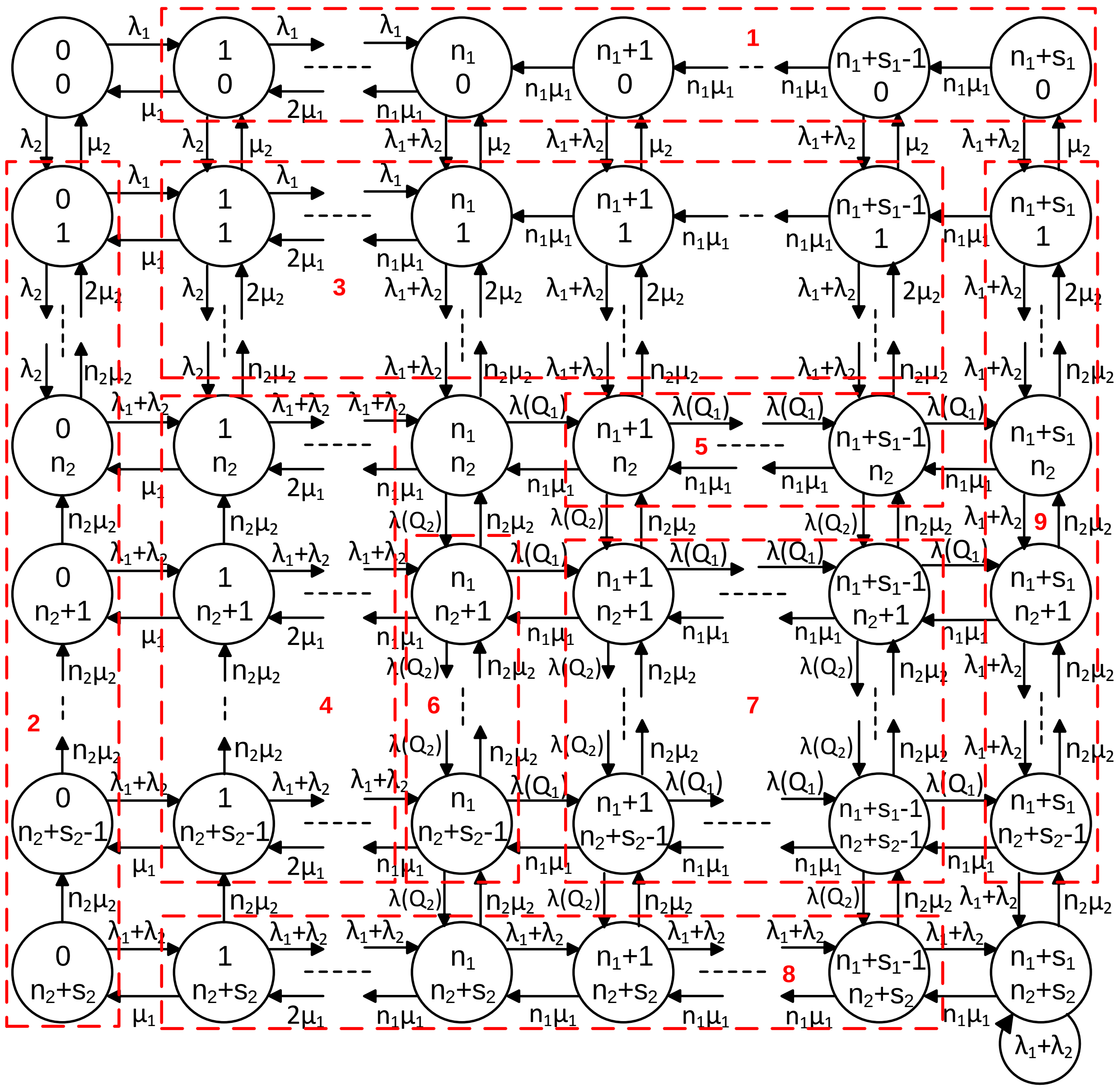


Figure 6-17 Solution Sub Spaces for SRTF Algorithm

- For first column states indicated by sub-space 2 in Figure 6-17:
  - For states $(0,1)$ till $(0, n_2 - 1)$

$$d = (\lambda_1 + \lambda_2 + x_2\mu_2)p(0,x_2) - \lambda_2 p(0,x_2 - 1) - \mu_1 p(1,x_2) - (x_2 + 1) * \mu_2 p(0,x_2 + 1)$$

(6-41)

  - For state $(0, n_2)$

$$d = (\lambda_1 + \lambda_2 + n_2\mu_2)p(0,n_2) - \lambda_2 p(0,n_2 - 1) - \mu_1 p(1,n_2) - n_2\mu_2 p(0,n_2 + 1)$$

(6-42)

- For states $(0, n_2+1)$ till $(0, n_2+s_2-1)$

$$d = (\lambda_1+\lambda_2+n_2\mu_2)p(0,x_2) - \mu_1 p(1,x_2) - n_2\mu_2 p(0,x_2+1)$$

(6-43)

- For state $(0, n_2+s_2)$

$$d = (\lambda_1+\lambda_2+n_2\mu_2)p(0,n_2+s_2) - \mu_1 p(1,n_2+s_2)$$

(6-44)

- For states included in rows from 1 to n_2-1 and columns from 1 to n_1+s_1-1 indicated by sub-space 3 in Figure 6-17:
  - For states where $x_1 = 1{:}n_1-1$ and $x_2 = 1{:}n_2-1$

$$d = (\lambda_1+\lambda_2+x_1\mu_1+x_2\mu_2)p(x_1,x_2) - \lambda_1 p(x_1-1,x_2) - \lambda_2 p(x_1,x_2-1) - (x_1+1)*\mu_1 p(x_1+1,x_2) - (x_2+1)*\mu_2 p(x_1,x_2+1)$$

(6-45)

  - For states $(n_1,1)$ till (n_1,n_2-1)

$$d = (\lambda_1+\lambda_2+n_1\mu_1+x_2\mu_2)p(n_1,x_2) - \lambda_1 p(n_1-1,x_2) - (\lambda_1+\lambda_2)p(n_1,x_2-1) - n_1*\mu_1 p(n_1+1,x_2) - (x_2+1)*\mu_2 p(n_1,x_2+1)$$

(6-46)

  - For states where $x_1 = n_1+1{:}n_1+s_1-1$ and $x_2 = 1{:}n_2-1$

$$d = (\lambda_1+\lambda_2+n_1\mu_1+n_2\mu_2)p(x_1,x_2) - (\lambda_1+\lambda_2)p(x_1,x_2-1) - n_1\mu_1 p(x_1+1,x_2) - (x_2+1)*\mu_2 p(x_1,x_2+1)$$

(6-47)

- For states included in rows from n_2 to n_2+s_2-1 and columns from 1 to n_1-1 indicated by sub-space 4 in Figure 6-17:
  - For states $(1,n_2)$ till (n_1-1,n_2)

$$d = (\lambda_1+\lambda_2+x_1\mu_1+n_2\mu_2)p(x_1,n_2) - (\lambda_1+\lambda_2)p(x_1-1,n_2) - \lambda_2 p(x_1,n_2-1) - (x_1+1)*\mu_1 p(x_1+1,n_2) - n_2*\mu_2 p(x_1,n_2+1)$$

(6-48)

  - For states where $x_1 = 1{:}n_1-1$ and $x_2 = n_2+1{:}n_2+s_2-1$

$$d = (\lambda_1+\lambda_2+x_1\mu_1+n_2\mu_2)p(x_1,x_2) - (\lambda_1+\lambda_2)p(x_1-1,x_2) - (x_1+1)*\mu_1 p(x_1+1,x_2) - n_2\mu_2 p(x_1,x_2+1)$$

(6-49)

- For state (n_1, n_2)

$$d = [\lambda(Q_1)(n_1,n_2) + \lambda(Q_2)(n_1,n_2) + n_1\mu_1 + n_2\mu_2]p(n_1,n_2) - (\lambda_1 + \lambda_2) * p(n_1,n_2-1) - (\lambda_1+\lambda_2)p(n_1-1,n_2) - n_1\mu_1 p(n_1+1,n_2) - n_2\mu_2 p(n_1,n_2+1)$$

(6-50)

- For states (n_1+1, n_2) till (n_1+s_1-1, n_2) indicated by sub-space 5 in Figure 6-17:

$$d = [\lambda(Q_1)(x_1,n_2) + \lambda(Q_2)(x_1,n_2) + n_1\mu_1 + n_2\mu_2]p(x_1,n_2) - (\lambda_1+\lambda_2) * p(x_1,n_2-1) - \lambda(Q_1)(x_1-1,n_2)p(x_1-1,n_2) - n_1\mu_1 p(x_1+1,n_2) - n_2\mu_2 p(x_1,n_2+1)$$

(6-51)

- For states (n_1, n_2+1) till (n_1, n_2+s_2-1) indicated by sub-space 6 in Figure 6-17:

$$d = [\lambda(Q_1)(n_1,x_2) + \lambda(Q_2)(n_1,x_2) + n_1\mu_1 + n_2\mu_2]p(n_1,x_2) - (\lambda_1+\lambda_2) * p(n_1-1,x_2) - \lambda(Q_2)(n_1,x_2-1)p(n_1,x_2-1) - n_1\mu_1 p(n_1+1,x_2) - n_2\mu_2 p(n_1,x_2+1)$$

(6-52)

- For states included in rows from (n_1+1) till (n_1+s_1-1) and columns from (n_2+1) till (n_2+s_2-1) indicated by sub-space 7 in Figure 6-17:

$$d = [\lambda(Q_1)(x_1,x_2) + \lambda(Q_2)(x_1,x_2) + n_1\mu_1 + n_2\mu_2]p(x_1,x_2) - \lambda(Q_1)(x_1-1,x_2) * p(x_1-1,x_2) - \lambda(Q_2)(x_1,x_2-1)p(x_1,x_2-1) - n_1\mu_1 p(x_1+1,x_2) - n_2\mu_2 p(x_1,x_2+1)$$

(6-53)

- For last row states indicated by sub-space 8 in Figure 6-17:
  - For states $(1, n_2+s_2)$ till (n_1-1, n_2+s_2)

$$d = (\lambda_1+\lambda_2+x_1\mu_1+n_2\mu_2)p(x_1,n_2+s_2) - (\lambda_1+\lambda_2)p(x_1-1,n_2+s_2) - (x_1+1) * \mu_1 p(x_1+1,n_2+s_2)$$

(6-54)

  - For states (n_1, n_2+s_2) till (n_1+s_1-1, n_2+s_2)

$$d = (\lambda_1+\lambda_2+n_1\mu_1+n_2\mu_2)p(x_1,n_2+s_2) - (\lambda_1+\lambda_2)p(x_1-1,n_2+s_2) - \lambda(Q_2)(x_1,n_2+s_2-1)p(x_1,x_2-1) - n_1\mu_1 p(x_1+1,n_2+s_2)$$

(6-55)

- For last column states indicated by sub-space 9 in Figure 6-17:
  - For states $(n_1 + s_1, 1)$ till $(n_1 + s_1, n_2 - 1)$

$$d = (\lambda_1 + \lambda_2 + n_1\mu_1 + x_2\mu_2)p(n_1 + s_1, x_2) - (\lambda_1 + \lambda_2)p(n_1 + s_1, x_2 - 1) - (x_2 + 1) * \mu_2 p(n_1 + s_1, x_2 + 1)$$

(6-56)

  - For states $(n_1 + s_1, n_2)$ till $(n_1 + s_1, n_2 + s_2 - 1)$

$$d = (\lambda_1 + \lambda_2 + n_1\mu_1 + n_2\mu_2)p(n_1 + s_1, x_2) - (\lambda_1 + \lambda_2)p(n_1 + s_1, x_2 - 1) - \lambda(Q_1)(n_1 + s_1 - 1, x_2) * p(n_1 + s_1 - 1, x_2) - n_2 * \mu_2 p(n_1 + s_1, x_2 + 1)$$

(6-57)

- Finally for state $(n_1 + s_1, n_2 + s_2)$

$$d = (n_1\mu_1 + n_2\mu_2)p(n_1 + s_1, n_2 + s_2) - (\lambda_1 + \lambda_2)p(n_1 + s_1 - 1, n_2 + s_2) - (\lambda_1 + \lambda_2)p(n_1 + s_1, n_2 + s_2 - 1)$$

(6-58)

Having obtained all state probabilities of a system of two DCs using Iterations of Gauss- Seidel method with defined threshold of $\epsilon = 10^{-6}$, the most significant performance metrics for evaluating algorithm's performance are calculated for the DC_i, $i = 1,2$, using the following equations below and illustrated by sup-spaces shown in Figure 6-18:

- Carried traffic by each DC_i Y_i indicating the average server occupancy at each DC_i

$$Y_1 = \sum_{x_1=0}^{n_1-1} \sum_{x_2=0}^{n_2+s_2} x_1 p(x_1, x_2) + \sum_{x_1=n_1}^{n_1+s_1} \sum_{x_2=0}^{n_2+s_2} n_1 p(x_1, x_2)$$

(6-59)

$$Y_2 = \sum_{x_1=0}^{n_1+s_1} \sum_{x_2=0}^{n_2-1} x_2 p(x_1, x_2) + \sum_{x_1=0}^{n_1+s_1} \sum_{x_2=n_2}^{n_2+s_2} n_2 p(x_1, x_2)$$

(6-60)

- Probability of a request being lost when both data centers are fully occupied B_i

$$B_1 = B_2 = p(n_1 + s_1, n_2 + s_2)$$

(6-61)

- Probability of a request being migrated from DC_i to a less loaded DC M_i

$$M_1 = \sum_{x_1=n_1}^{n_1+s_1} \sum_{x_2=0}^{n_2-1} p\,(x_1, x_2) + \sum_{x_2=n_2}^{n_2+s_2-1} p\,(n_1 + s_1, x_2) + \sum_{x_1=n_1}^{n_1+s_1-1} \sum_{x_2=n_2}^{n_2+s_2-1} p\,(x_1, x_2) * \frac{\lambda(Q_2)}{\lambda(Q_1) + \lambda(Q_2)}$$

(6-62)

$$M_2 = \sum_{x_1=0}^{n_1-1} \sum_{x_2=n_2}^{n_2+s_2} p\,(x_1, x_2) + \sum_{x_1=n_1}^{n_1+s_1-1} p\,(x_1, n_2 + s_2) + \sum_{x_1=n_1}^{n_1+s_1-1} \sum_{x_2=n_2}^{n_2+s_2-1} p\,(x_1, x_2) * \frac{\lambda(Q_1)}{\lambda(Q_1) + \lambda(Q_2)}$$

(6-63)

where an arrival is migrated from DC_i if there exists an empty server in DC_j and none at DC_i, or if all servers and buffer places in DC_i are occupied while DC_j has an empty buffer space, or finally if there are empty buffer spaces in both DC_i and DC_j but DC_j offers a shorter response time.

- Mean number of buffered arrivals L_i

$$L_1 = \sum_{x_1=n_1+1}^{n_1+s_1} \sum_{x_2=0}^{n_2+s_2} (x_1 - n_1) p(x_1, x_2)$$

(6-64)

$$L_2 = \sum_{x_1=0}^{n_1+s_1} \sum_{x_2=n_2+1}^{n_2+s_2} (x_2 - n_2) p(x_1, x_2)$$

(6-65)

- Probability that a request is immediately served upon arrival to its own DC I_i

$$I_1 = \sum_{x_1=0}^{n_1-1} \sum_{x_2=0}^{n_2+s_2} p(x_1, x_2)$$

(6-66)

$$I_2 = \sum_{x_1=0}^{n_1+s_1} \sum_{x_2=0}^{n_2-1} p(x_1, x_2)$$

(6-67)

where a request is immediately served at its DC_1 if a server was idle at the instance of its arrival.

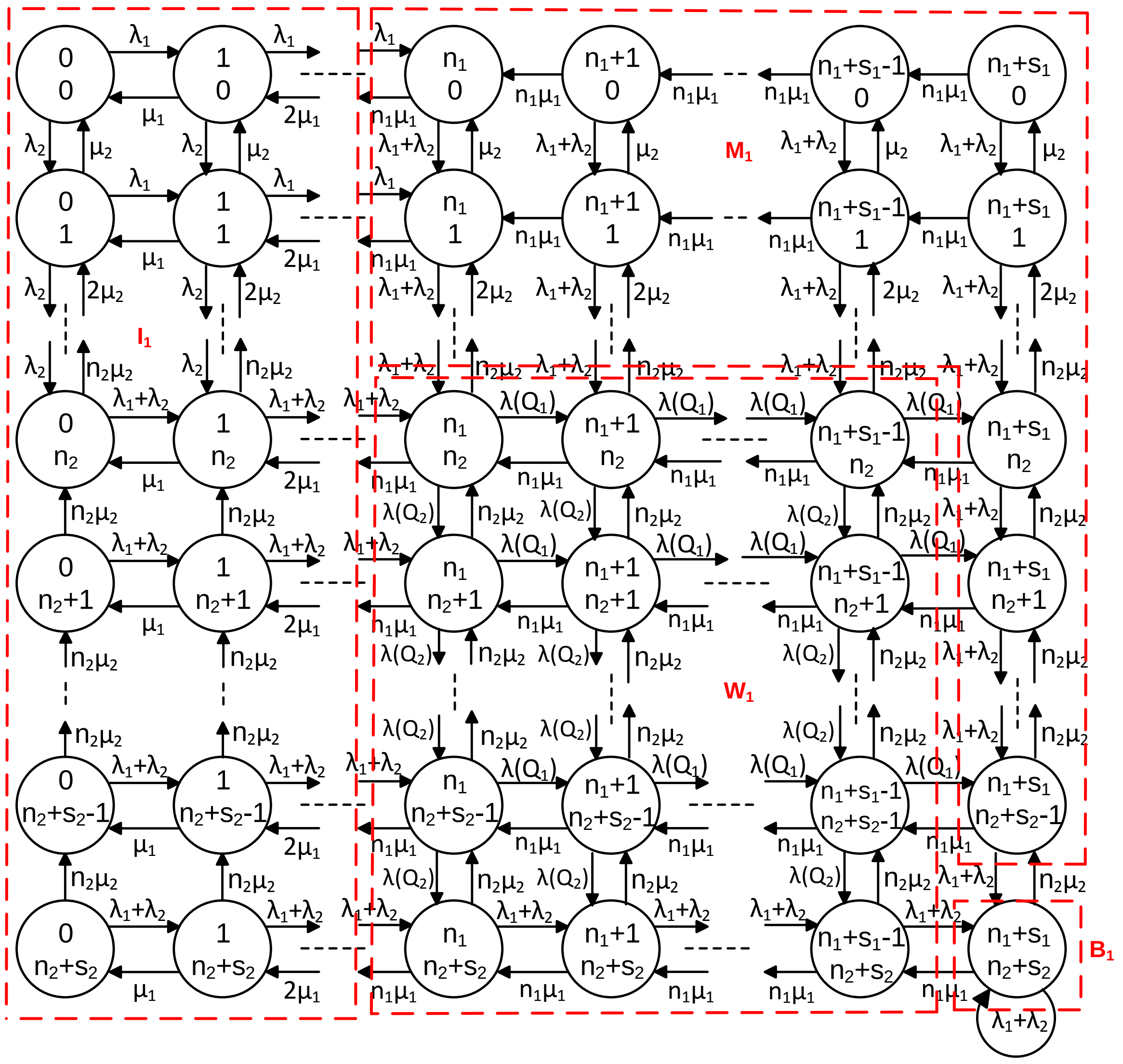


Figure 6-18 Sub Spaces for Performance Metrics of SRTF Algorithm

- Probability of a request being delayed upon arrival to its own DC W_i

$$W_1 = \sum_{x_1=n_1}^{n_1+s_1-1} \sum_{x_2=n_2}^{n_2+s_2-1} p(x_1, x_2) * \frac{\lambda(Q_1)}{\lambda(Q_1)+\lambda(Q_2)} + \sum_{x_1=n_1}^{n_1+s_1-1} p(x_1, n_2+s_2)$$

(6-68)

$$W_2 = \sum_{x_1=n_1}^{n_1+s_1-1} \sum_{x_2=n_2}^{n_2+s_2-1} p(x_1, x_2) * \frac{\lambda(Q_2)}{\lambda(Q_1)+\lambda(Q_2)} + \sum_{x_2=n_2}^{n_2+s_2-1} p(n_1+s_1, x_2)$$

(6-69)

Equations 6-68 and 6-69 describe the cases at which an arriving request will be buffered at its own DC. Buffering occurs if all local servers are occupied and the remote DC is fully occupied so the arrival can only be buffered at its local buffer, or if the waiting time that will be experienced in the local buffer is shorter than the waiting time at the remote buffer.

- Mean waiting time of a request w_i

$$w_1 = \sum_{x_1=n_1}^{n_1+s_1-1} \sum_{x_2=n_2}^{n_2+s_2-1} p(x_1,x_2) * (x_1 - n_1 + 1) * \frac{h_1}{n_1} * \frac{\lambda(Q_1)}{\lambda(Q_1)+\lambda(Q_2)} + \sum_{x_1=n_1}^{n_1+s_1-1} p(x_1, n_2+s_2) * (x_1 - n_1 + 1) * \frac{h_1}{n_1}$$

(6-70)

$$w_2 = \sum_{x_1=n_1}^{n_1+s_1-1} \sum_{x_2=n_2}^{n_2+s_2-1} p(x_1,x_2) * (x_2 - n_2 + 1) * \frac{h_2}{n_2} * \frac{\lambda(Q_2)}{\lambda(Q_1)+\lambda(Q_2)} + \sum_{x_2=n_2}^{n_2+s_2-1} p(n_1+s_1, x_2) * (x_2 - n_2 + 1) * \frac{h_2}{n_2}$$

(6-71)

The mean waiting time of arrivals at the local DC is calculated using Little's Theorem and delay probabilities explained by Equations 6-68 and 6-69. Each term in the two equations is multiplied by the average delay that would be experienced by the arrival at its own DC, which is equal to the number of customers ahead of the arrival in the queue upon its arrival instance divided by the service rate of the DC.

- Mean waiting time of a delayed request t_{wi}

$$t_{wi} = w_i / W_i$$

(6-72)

where the mean waiting time of delayed arrivals is calculated by dividing the mean waiting time of all arrivals by the waiting probability.

Results for all above explained performance metrics will be shown in the following Section 6.2.3 and verified against results from simulations as well as test bed experimentation.

6.2.2.2 Simulation Model

Similar to the simulation model for LSSF algorithm introduced in sub-Section 6.1.2.2 the simulation model of SRTF algorithm has the same components and functions; except for decisions on placement of arriving requests which will only be explained here to avoid repetition. When an arriving request is generated at the source module of the DC, it checks the status of its own DC as well as all other DCs in order to forward the request to the DC at which it can receive the lowest response time. Arrival requests exit the source module via the output gate connected to the local DC only when there is an idle local server available or if the local passive queue length is lower than length of all other foreign queues. When the arriving request has to be migrated, the source module forwards it to another DC with an idle server or if none exist then to the DC offering shortest response time. Calculations for loss, migration and delay probabilities as well as mean delay of delayed requests are calculated using equations 6-30, 6-31, 6-32 and 6-33 respectively.

6.2.2.3 Experimental Setup

The test-bed configuration used for implementing SRTF algorithm and evaluating its performance on real server equipment is the same as that explained in Section 6.1.2.3 and will not be explained again to avoid repetition. The only difference is in the PowerCLI scripts deployed on vSphere for implementing the algorithm and its allocation strategies for VM requests.

6.2.3 Results

This section evaluates the performance of SRTF algorithm and its effectiveness in balancing the load between DCs while maintaining the lowest possible mean delay for delayed arrivals. Besides solving the model analytically using Gauss-Seidel method, it is simulated on two DCs using OMNeT++ and implemented on a DC test-bed. Results for the three approaches are shown in Figure 6-19 for the waiting, loss and migration probabilities of arriving requests at the two DCs compared to waiting and loss probabilities of an equivalent DC without load balancing. Results from the simulation and test-bed are presented with 95% confidence intervals using a sample size of 20. The figure shows that when compared to a DC without load balancing both waiting and loss probabilities are reduced by the SRTF algorithm as load balancing allows arriving requests that would have been delayed or lost at their local DC to be migrated to another DC where it will experience lower response time. Considering different implementation approaches for the algorithm, all the three methods show almost the same results for probabilities with only minor differences due to the difference in how each platform operates. Most importantly is that the analytical solution always provides upper bound for the results, which allows for accurate prediction for the behavior of DCs. The same effect can be observed in Figure 6-20 showing mean delay for delayed arrivals obtained from different platforms compared to that of a DC without load balancing. As illustrated in the results, SRTF is able to maintain the low delay values until high load values where delay slightly increases due to the migrated arrivals that wait at foreign DCs.

Zero delay values at low-loads resulting from simulation and test-bed platforms occurs due to the minor waiting probability approaching 10^{-6} which implies one in a million delayed arrivals. Such rare effects can hardly be captured in the simulation unless it runs for several millions of users and for the test-bed unless millions of VMs were created. However both scenarios would consume extremely long time durations (in order of days) that could not be conducted through this work and is of minimal importance as the main concern for delay values is to always be kept under defined SLA levels during high load regions

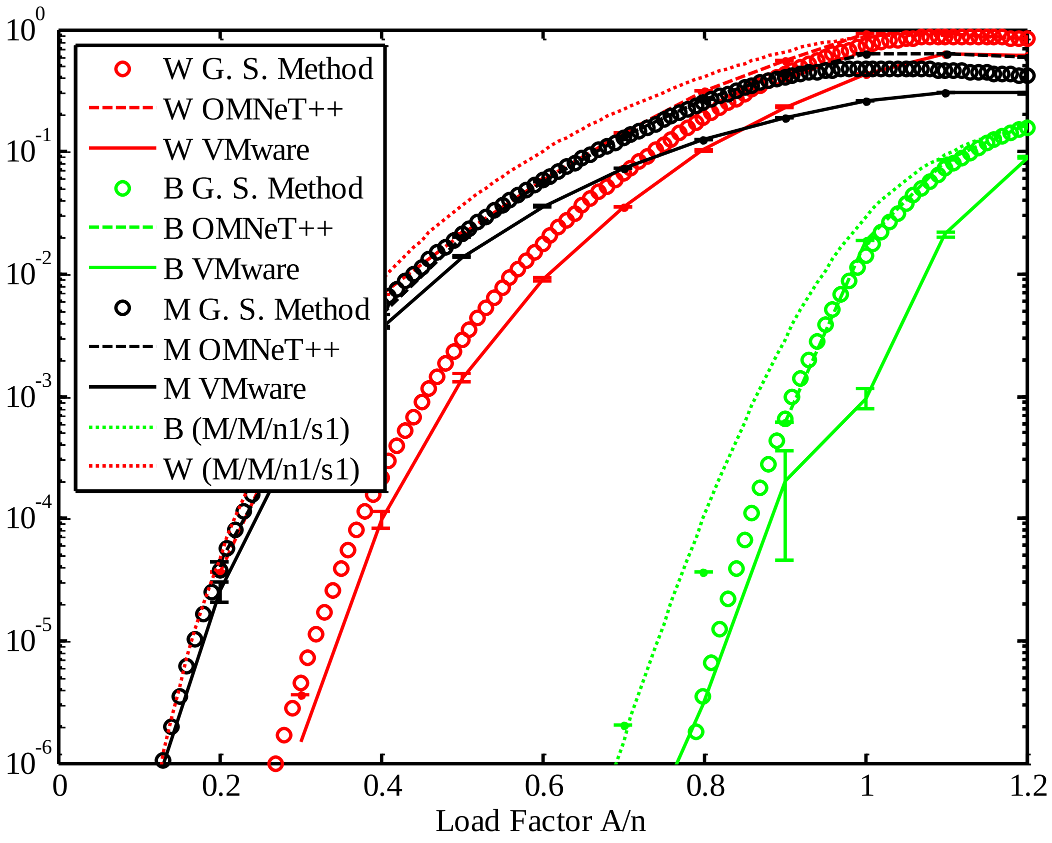


Figure 6-19 Loss (B), Migration (M) and Delay (W) Probabilities vs. Load (A/n) for SRTF Algorithm implemented on various Platforms

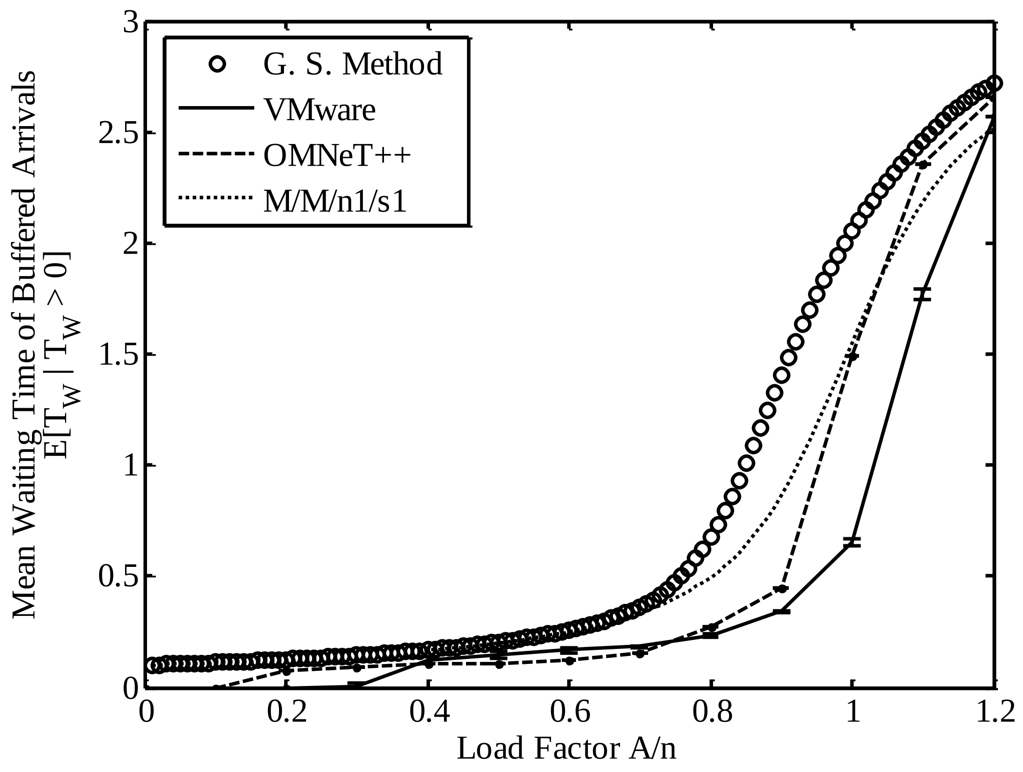


Figure 6-20 Mean Waiting Time of Delayed Requests versus Load (A/n) for SRTF Algorithm implemented on various Platforms

For further validation of the SRTF model and the assumptions made for solving it analytically using Markov Chains, the simulation model has been altered to test other inter-arrival and service time distributions than the negative-exponential distribution assumed by Markovian models. Figures 6-21 and 6-22 show the mean delay of delayed requests while using various distributions with the same mean value for generating requests' inter-arrival and service times, respectively. In Figure 6-21 the following distributions have been used for generating inter-arrival durations:

- Exponential distribution with mean $t_{inter-arrival}$
- Constant interarrival times = $t_{inter-arrival}$
- Uniform distribution with limits $[0,2 * t_{inter-arrival}]$

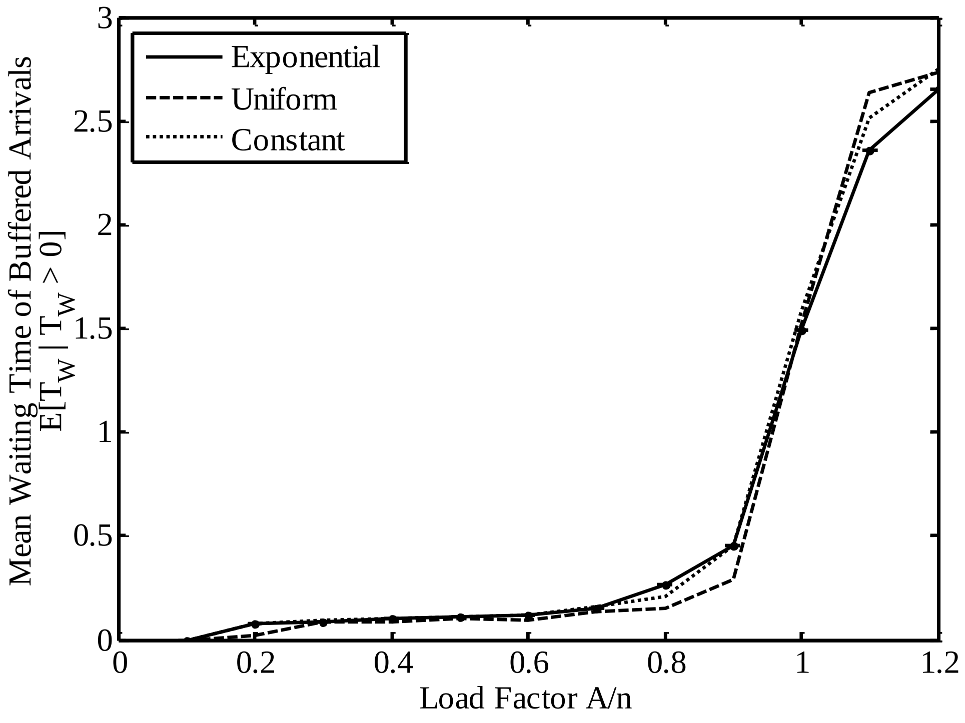


Figure 6-21 Mean Delay of Delayed Requests E [Tw |Tw>0] for SRTF Algorithm for different Inter-Arrival Time Distributions

Mean delays in Figure 6-22 were obtained for service durations generated by the following distributions:

- Exponential distribution with mean 1
- Normal distribution with mean 1 and variance 1[2]
- Uniform distribution with limits [0,2]
- Constant interarrival times = 1

[2] The mean and variance used for this case in the simulation tool refer to the range $-\infty<t<\infty$ of the stochastic variable t. The error for the mean and variance due to the truncation decreases rapidly with increasing mean

Both figures show that the difference in delay value resulting from inter-arrival and service times obtained from different distributions are minimal, which proves the applicability of the proposed models to any type of DCs and the validity of Markovian assumptions required for modeling and solving the algorithms using Markov Chains.

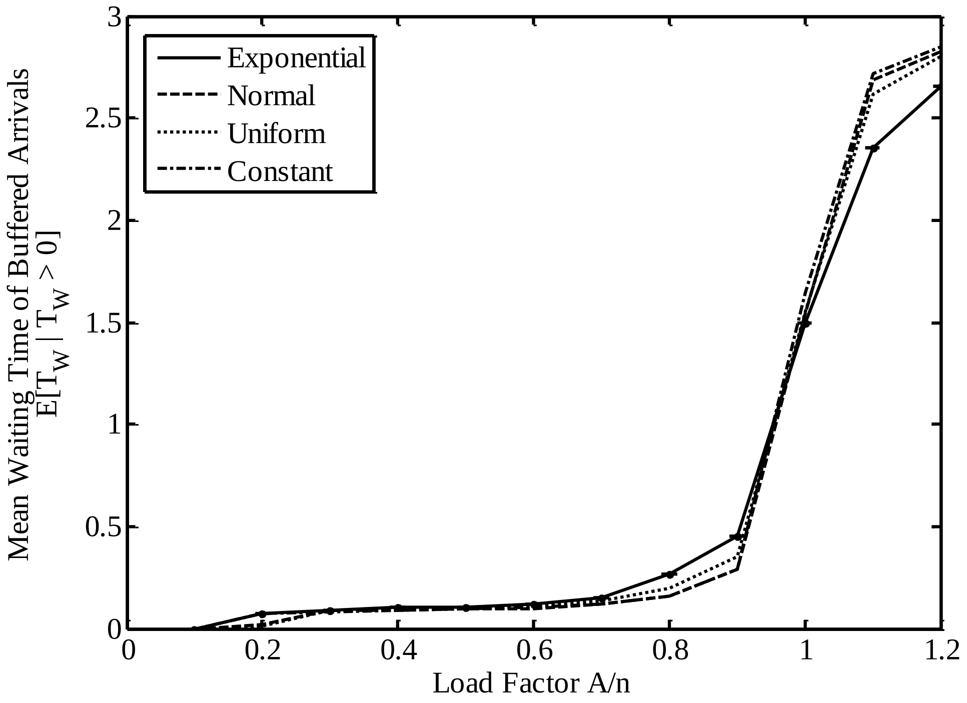


Figure 6-22 Mean Delay of Delayed Requests E [Tw |Tw>0] for SRTF Algorithm for different Service Time Distributions

6.3 Comparison and Evaluation

Having provided results for the performance of LSSF and SRTF algorithms within the previous two sections of this chapter, this section provides a comparison between the two algorithms with an evaluation for each algorithm and use-cases when each one can be of better use. Results in this section was based on DCs with 10 servers, a queue of size 30 and for LSSF algorithm migration thresholds are set to 19 and 39 for cases with/without migration overhead, respectively. Another test case is included for SRTF algorithm with migration overhead ($t_m = 2$) obtained from test-bed experimentation in order to have a fair comparison between the two algorithms in cases of negligible ($t_m = 0$) or long ($t_m = 2$) migration overheads.

Considering delay as a main factor affecting users' SLA and their Quality of Experience (QoE), Figure 6-23 compares mean delays experienced by those arriving requests who wait for a DC implementing SRTF algorithm and LSSF algorithm with/without migration overhead and an equivalent DC without load balancing. Results show that SRTF algorithm can achieve the lowest average delay for delayed arrivals over all load values since delay is its main criteria for load balancing. Increase in migration overhead only results in a minimal increase in the mean delay values at low and intermediate load regions since the algorithm will always assign the arriving requests to the DC with minimal response time and migration overhead. As load value increases all DCs tend to be overloaded, thus migrations decrease and losses increases resulting in a decrease in the mean delay value. As for LSSF algorithm it maintains the same delay experience of users in DCs without load balancing for low and intermediate load values, whereas for high and overload regions when migrations occur the delay value differs according to the value of migration overhead. For negligible values of migration overhead the average delay tends to be higher as less arrivals are lost where they are migrated and delayed at foreign DCs. As migration overhead increases these migrations are reduced and accordingly the mean delay value. Disregarding migration overheads, SRTF algorithm is able to provide lower mean response times for delayed requests.

These effects are also visible in Figure 6-24 which compares loss, delay and migration probabilities for the previously mentioned systems. Shortest mean delays of SRTF algorithm shown in Figure 6-23 is caused by lower loss and delay probabilities compared to a DC without load balancing, where the decrease in these probabilities is compensated by a high migration probability since arriving requests are always routed to the DC offering least response time. DCs operating under LSSF algorithm also achieve lower loss probabilities than those without load balancing as arrivals can be migrated to foreign DCs instead of being lost.

Comparing the two algorithms together; both algorithms are able to reduce loss probabilities with slightly lower losses achieved by SRTF algorithm due to absence of migration thresholds that could prevent a foreign DC from accepting a migrated request. SRTF also shows lower waiting probabilities than LSSF for both values of migration overhead due to the fact that SRTF migrates an arrival to a foreign DC if it could receive instant service there other than waiting at its local queue. These effects are compensated by much higher migration probabilities for SRTF as LSSF algorithm prevents an arriving request to be migrated unless it finds its local DC at its full capacity which only occurs at high/overload situations; whereas SRTF allows migrations regardless of system load. Migration probabilities for both algorithms tend to drop at overload situations when the migration overhead is high, where at these situations the DCs tend to be overloaded with long expected delay durations for any arriving request where these long delays added to it long migration overheads would not meet the migration criteria of a request to have its delay limit specified by SLAs met at the foreign DC upon migration.

Generally, each of the proposed load balancing algorithms presented in this chapter is superior under certain conditions. For cases when users' requests are highly sensitive to delay and service level agreements SRTF would be the best choice for load balancing among DCs as it provides the lowest mean delay an arrival could experience. However for cases with high migration overheads and long transmission delays between geographically distant DCs, LSSF would perform better as it tends to suppress migrations to overload situations only. Determining migration thresholds of LSSF algorithm could be tuned according to migration delay values, where long delays require low thresholds and short migration overheads require higher thresholds.

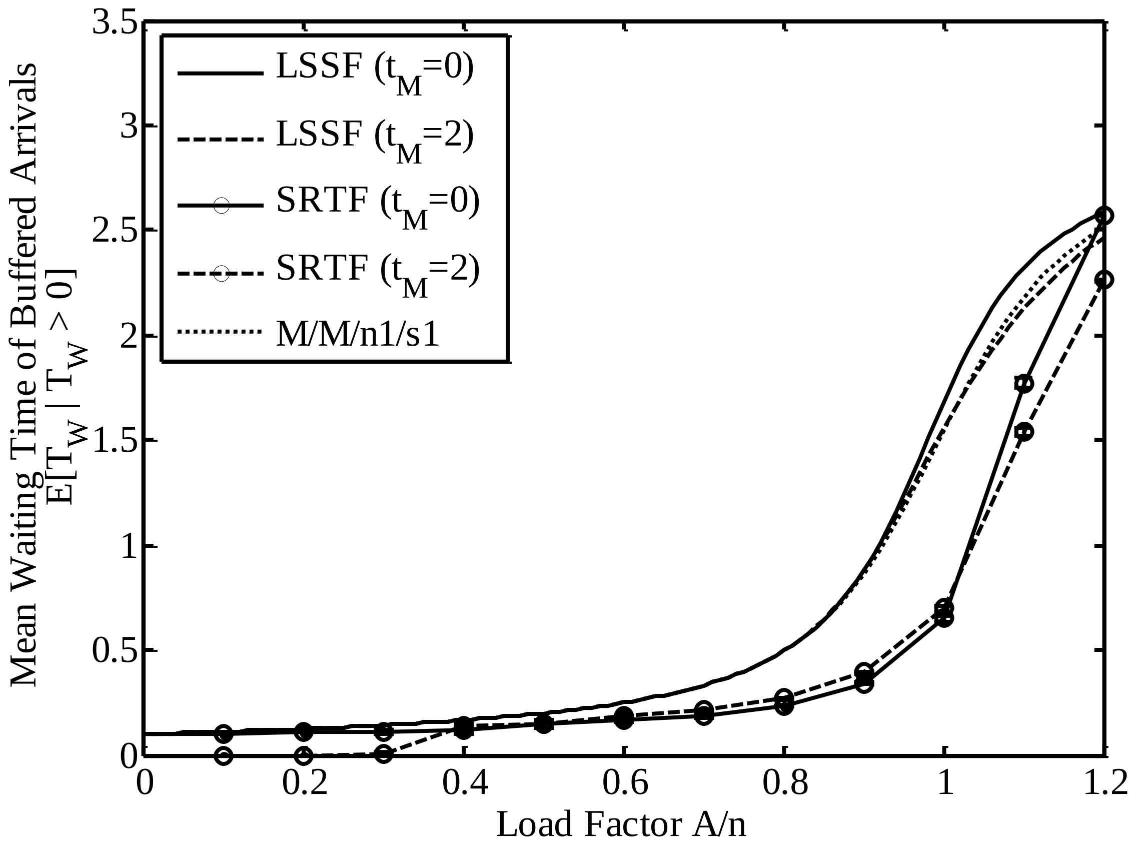


Figure 6-23 Mean Delay of Delayed Frames for LSSF and SRTF Algorithms compared to $\boldsymbol{M/M/n/n+s}$ Queue

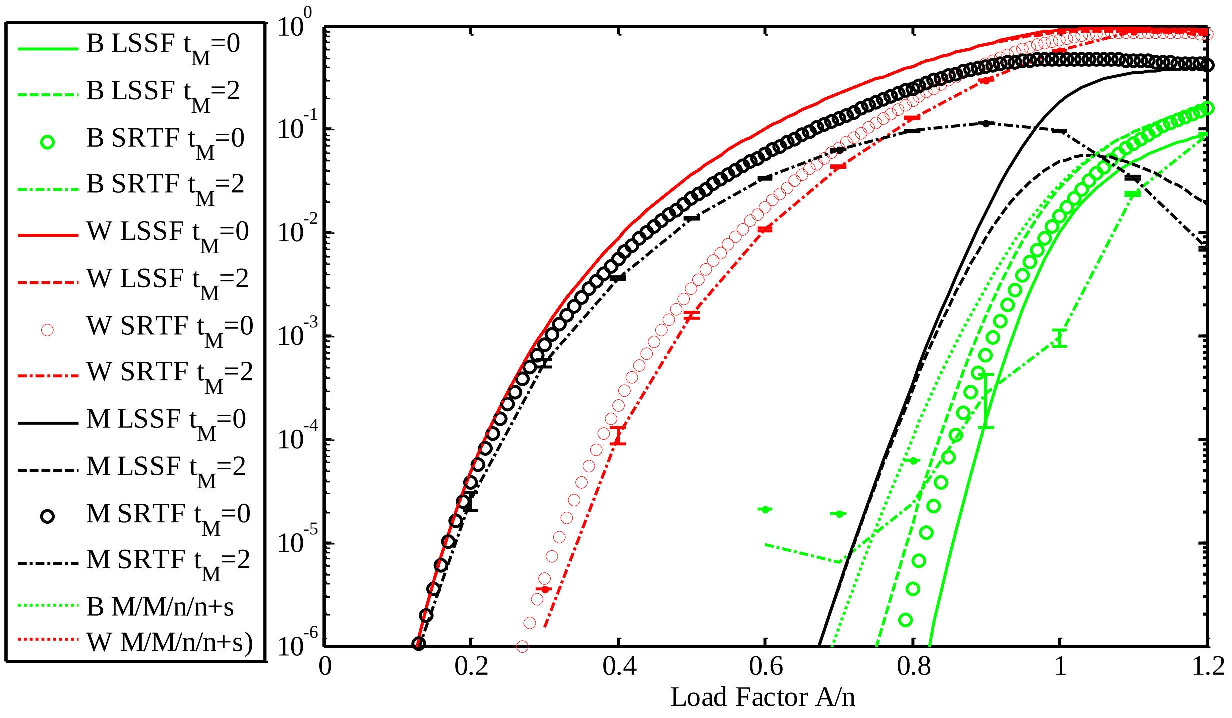


Figure 6-24 Loss (B), Delay (W) and Migration (M) Probabilities for LSSF and SRTF Algorithms compared to $\boldsymbol{M/M/n/n+s}$ Queue

Chapter 7 Conclusion and Outlook

The rise of Cloud Computing technology and its huge development over the past years resulted in a rapid growth in the size of its data centers and accordingly its energy bill. Power consumed by data centers for operating and cooling of its equipment contributes to almost 1.5% of the world-wide energy consumption, and is forecasted to increase over the coming years. In attempt to reduce this increased consumption numerous approaches have been proposed such as server consolidation, sleep modes and reduced operational frequency by Dynamic Voltage and Frequency Scaling, all of which tend to achieve an energy efficient operation by cloud data centers. Besides the energy challenge, designing load balancing algorithms to balance the load among several data centers of cloud providers while meeting the requirements of cloud computing services is another challenge. Despite the existence of several load balancing approaches in literature most of them could not offer efficient load balancing solutions for cloud computing for being static, having single points of failure, requiring long transmission delays or long processing delays due to complexity. Cloud Computing however require algorithms that are dynamic with automatic resource provisioning for fast adaptation to any updates in the data center's state, proactive to over-load situations by preventing them from occurring, distributed without centralized points of failure to avoid relying the data center's operation on a single node, and finally and most importantly perform load balancing decisions without any compromise to users' specified Service Level Agreements (SLA).

This thesis contributes to solving the energy efficiency and load balancing dilemmas in Cloud Computing environments by proposing an algorithm for energy efficiency based on the idea of server consolidation for an efficient operation of DCs; as well as two load balancing algorithms to distribute the load among several operator's DCs with the goal of enhancing DCs' performance while meeting users' SLAs. For studying these algorithms and accurately predicting their effects on DCs' performance a modeling approach for DCs as queuing systems has been introduced in Chapter 4. In the context of chapter 4 all modeling details and assumptions that accurately describe the DC's operation using queuing system operations are illustrated by describing different system states and transition among these different states using Markov Chains. Besides explaining theoretical methodology followed for analyzing the proposed algorithms, Chapter 4 also introduced other platforms used for testing the algorithms. OMNeT++ simulation models are explained with their compound and simple modules required to implement a queuing system with different operational strategies in accordance to the implemented algorithm. A test-bed for emulating cloud DCs has also been set-up for better understanding of the algorithms' behavior in realistic environments.

Within the context of Chapter 5 in this thesis an algorithm for energy efficiency operation of Data centers has been proposed. The algorithm is based on the idea of server consolidation by consolidating the DC's load on fewer number of servers so that other under-utilized servers are turned idle and can be switched-off to save their idle power consumption. This is performed in the algorithm using hystereses behavior where defined queuing thresholds are set to determine a load surge that requires new server activation. The algorithm is explained using Markov Chains and analyzed mathematically using a novel recursive algorithm which solves for the steady-state probabilities of the system under Markovian assumptions. Most significant performance metrics required for evaluating the algorithm are explained such as probabilities of system states, mean delays of delayed arriving requests, servers' activation rates and power saving efficiency. The algorithm has shown its ability to dynamically adapt the number of active servers in the DC to its current load, so that the load of the DC is consolidated on the least number of active servers. This is proven by showing the probability of state peaks at each corresponding load value and the vast reduction in activation rates of servers. Besides its effective consolidation strategy the algorithm also shows a favorable effect of stabilizing the mean delay of delayed requests over a vast range of loads, where delays remain almost of a constant value while system is loaded between 5 to 95%. The proposed hystereses algorithm has also proven its efficiency at meeting its original goal of reducing power consumption of DCs with an achieved power saving efficiency of 20% at low load values due to the reduced number of active servers.

In addition to its efficiency at reducing DC's power consumption, the proposed hystereses algorithm accommodates several realistic properties of servers such as activation overheads, sleep modes and DVFS specified by C-states and P-states, respectively. Two case studies have been presented in this thesis to show case the effectiveness of sleep modes and reduced frequency states at increasing the energy efficiency of cloud data centers. The first case study was implemented by the model of the proposed hystereses algorithm by varying activation overheads as well as power consumption at different system states to model servers being in sleep states, where long activation overheads implied deeper sleep states and less energy consumption whereas short activation overheads implied light sleep states and relatively higher power consumption. Sleep states indicated by HSB modes were also compared against CSB mode where idle servers are completely switched-off. Analysis have shown that sleep modes are able to reduce the delay experienced while booting a server from an off state significantly at low load values, while switching-off servers is slightly more efficient at reducing energy consumption. Decisions on how to compromise between saving energy with minimal degradation in system performance should be carefully considered by DCs' administrators for selecting appropriate operation modes for their systems. The second case study provided insights into the application of DVFS states on servers for reducing their frequency and accordingly power consumption. Model results have shown that as operational frequency decreases the service durations of arriving requests is increases as well as number of buffered arrivals thus triggering more servers to be activated. Thus the attempt to reduce energy consumption is accompanied by increased delay of requests causing performance degradation as well as more activated servers that compensate the power-saving effect. Thus DVFS should only be applied in cases when a DC is hosting requests that are not time-sensitive in order to save energy consumption considerably.

This thesis also introduced two algorithms for load balancing between cloud DCs within the context of Chapter 6, LSSF and SRTF. Both algorithms have been modeled using two-dimensional Markov Chains and solved analytically using an iterative algorithm for LSSF and Gauss-Seidel method for SRTF to obtain steady state probabilities of the DC operating by the model. The two algorithms are proposed for different use-cases; LSSF algorithm is proposed for scenarios when migration overhead between DCs is large compared to service and waiting times of requests as it suppresses migrations to cases when an arrival could not be accepted by its local DC. This effect has been proven by results of the analytical solution where LSSF algorithm shows reduced loss probabilities for arrivals as it migrates arrivals instead of being lost as well as slightly higher mean delays but bounded to be lower than specified SLAs. Different cases of LSSF algorithm have been tested with various migration overheads to show their effect where higher thresholds increase migration probability and average delays. These configuration parameters of the algorithm should be set according to SLA requirements agreed upon between cloud providers and their users.

The second proposed algorithm is an upgrade to the known JSQ algorithm which performs load balancing decisions in consideration to response times of servers in the DC rather than queue lengths. SRTF routes an incoming arrival to the DC where it would receive either an instant service or the shortest average delay time. Analysis of the algorithm showed that it is able to significantly reduce the average delays of requests compared to LSSF and DCs without load balancing, but at the expense of high migration probabilities. SRTF is foreseen to be most efficient at balancing the load among DC handling time-sensitive applications where migration overhead is not significant to the average delay values.

In addition to analytical solutions provided for the proposed algorithms in this thesis that easily enable system administrators to tune their system parameters and accurately predict the performance of DCs operating by any of the proposed algorithms; models have also been tested by simulations as well as test-bed implementations for exploring different application cases that cannot be tested by theoretical analysis as they are based upon Markovian assumptions. OMNeT++ simulations carried out for the three proposed algorithms with various distributions for inter-arrival times, service times and activation times showed that Markovian assumptions are a valid assumptions as model's performance was not affected by the change of any of the distributions. Test-bed experiments for load balancing algorithms on 4 DCs rather than the case of only two DCs implemented by the analytical solutions also showed its validity and applicability to balance the load among any number of DCs.

In conclusion, proposed algorithms in this thesis have shown their effectiveness at achieving an energy efficient operation for cloud DCs as well as balancing the load among several DCs effectively. Analysis methods used for studying these algorithms have also shown to be of high accuracy at predicting behavior of DCs operating by these algorithms. Close observation of the presented results show that energy efficiency algorithm is most effective at low/average load situations where the gain by reducing or eliminating power consumption of idle servers is high, while at high loads when the system is fully operative the power saving efficiency tends to zero. Whereas for load balancing the proposed algorithms have shown to be of most importance at regions of high load / overload to prevent losses of requests by migrating them to foreign DCs. Different load balancing criterion were shown to have different effects on QoE of arriving requests in terms of average delay, thus selecting a strategy for load balancing among cloud DCs should always be done in reference to users SLAs to make sure they are not compromised.

An outlook for the work done in this thesis is to compare the results obtained from testing proposed algorithms against other existing approaches in literature. For example; recursive algorithm proposed for solving the multiple hystereses model for energy-efficiency could be compared to other analysis methods for the same queuing model such as Green's Method or Stochastic Complementation which provide exact numerical solutions for multi-server queuing systems with hystereses. However this was not performed due to the high complexity of the previously mentioned methods and their limitations to solving test-cases of small sizes. As for the load balancing algorithms, further studies are required for approaching an exact numerical solution for solving steady-state probabilities of a model composed of any number of DCs with mutual overflows. Algorithms for mathematical analysis proposed in this thesis are limited to cases of only two DCs with mutual overflow represented using two-dimensional Markov Chains, where the addition of further DCs to the model corresponds to an increase in the dimension of the Markov Chain resulting in high dimensional models that cannot be solved exactly to the best of our knowledge but can be simulated by extending the proposed simulation models.

Through the context of this thesis algorithms have been proposed and tested using various methods to predict their effect when deployed to real DCs. Although a test-bed was configured to emulate a DC using minimal hardware equipment and the same software used by cloud operators, experimentation on large-scale DCs would be more insightful for testing the algorithms on larger set of equipment serving real online user requests instead of traffic generated using simulators. This step was not feasible during the time frame of this thesis as it required granted access for carrying out experimental work at one of the cloud providers' premises, which unfortunately was not achievable. Such experimentation could allow for testing the algorithms under more complicated scenarios beyond the capability of analytical solution such as balancing the load between more than two DCs or handling bulk arrivals; which are all cases that could be predicted using our simulation models

References

[1] "Big Data's Impact on the data Supply Chain", Cognizant 20-20 Insights, May 2012

[2] "Data Center Efficiency with Intel Power Management Technologies", Intel Information technology, February 2010

[3] "Data Center Users Group Special Report: Inside the Data Center 2008 and Beyond", Technical Report by Emerson Network Power, 2008.

[4] "Energy Logic: Reducing Data Center Energy Consumption by Creating Savings that Cascade Across Systems", White Paper by Emerson Network Power, 2009.

[5] "Enhanced Intel SpeedStep Technology for the Intel Pentium M Porcessor", White Paper by Intel, March 2004 http://download.intel.com/design/network/papers/30117401.pdf

[6] A. B. Nagarajan, F. Mueller, C. Engelmann, S. L. Scott, "Proactive fault tolerance for HPC with Xen virtualization", Proceedings of the 21st Annual International Conference on Supercomputing, Seattle, Washington, USA, June 17-21, 2007, pp. 23-32.

[7] A. Beloglazov, R. Buyya, "Adaptive threshold-based approach for energy-efficient consolidation of virtual machines in cloud data centers", Proceedings of the 8th International Workshop on Middleware for Grids, Clouds and e-Science, article no. 4, Bangalore, India, 2010

[8] A. Beloglazov, R. Buyya, "Energy Efficient Allocation of Virtual Machines in Cloud Data Centers", 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing, Melbourne, Australia, 2010, pp. 577-578.

[9] A. Beloglazov, R. Buyya, "Energy Efficient Resource Management in Virtualized Cloud Data Centers" Proceedings of the 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing, 2010, pp. 826-831.

[10] A. Beloglazov, R. Buyya, "Managing Overloaded Hosts for Dynamic Consolidation of Virtual Machines in Cloud Data Centers Under Quality of Service Constraints", IEEE Transactions on Parallel and Distributed Systems, volume 24, No.7, pp. 1366-1379, July 2013.

[11] A. Beloglazov, R. Buyya, Y. C. Lee, A. Zomaya, "A Taxonomy and Survey of Energy-Efficient Data Centers and Cloud Computing Systems", Journal of Advances in Computers, vol. 82, pp. 47-99, 2011.

[12] A. Checko, H. L. Christiansen, Y. Yan, L. Scolari, G. Kardaras, M. S. Berger, L. Dittmann, "Cloud RAN for Mobile Networks: A Technology Overview", IEEE Communications Surveys and Tutorials, vol. 17, no. 1, pp. 405-426, 2015.

[13] A. Gandhi, M. H. Balter, I. Adan, "Server Farms with Setup Costs", Journal of Performance Evaluation, vol. 67, issue 11, pp. 1123-1138, November 2010.

[14] A. Gandhi, M. H. Balter, M. A. Kozuch, "The Case for Sleep States in Servers", Proceedings of the 4th Workshop on Power-Aware Computing and Systems, Article no. 2, Portugal, October 2011.

[15] A. Gandhi, M. H. Balter, R. Das, C, Lefurgy, "Optimal Power Allocation in Server

Farms", Proceedings of the 11th International Joint Conference on Measurement and Modeling of Computer Systems, Seattle ,USA, June 2009, pp. 157-168

[16] A. Gandhi, S. Doroudi, M. H. Balter, A. S. Wolf, "Exact Analysis of the M/M/k/Setup Class of Markov Chains via recursive Renewal Reward", proceedings of the ACM SIGMETRICS International Conference on Management and Modeling of Computer Systems, pp. 153-166, USA, June 2013.

[17] A. Gandhi, V. Gupta, M. H. Balter, M. A. Kozuch, "Optimality Analysis of Energy-Performance Trade-off for Server Farm Management", Journal of Performance Evaluation, vol. 67, issue 11, pp. 1155-1171, November 2010.

[18] A. Greenberg, J. Hamilton, D. A. Maltz, P. Patel, "The cost of a Cloud: research problems in data center networks", ACM SIGCOMM Computer Communication Preview Newsletter, vol. 39, issue 1, pp. 68-73, January 2009.

[19] A. Gulati, A. Holler, M. Ji, G. Shanmuganathan, C. Waldspurger, X. Zhu, "VMware Distributed Resource Management: Design, Implementation, and Lessons Learned", VMware technical Journal, Spring 2012.

[20] A. Iosup, S. Ostermann, N. Yigitbasi, R. Prodan, T. Fahringer, D. Epema, "Performance Analysis of Cloud Computing Services for Many-Tasks Scientific Computing", IEEE Transactions on Parallel and Distributed Systems, Volume 22 Issue 6, pp. 931-945, June 2011

[21] A. Khiyaita, M. Zbakh, H. El Bakkali, D. El Kettani, "Load Balancing Cloud Computing: State of art", Network Security and Systems (JNS2), pp. 106-109, 20-21 April 2012.

[22] A. Koepke et al., "Simulating Wireless and mobile Networks in OMNeT++ the MiXiM vision", Proceedings of the 1st International Conference on Simulation Tools and techniques for Communications, Networks and Systems & Workshops, article no. 60, Marseille, France, March 2008

[23] A. Krioukov, P. Mohan, S. Alspaugh, L. Keys, D. Culler, R. H. Katz, "NapSAC: Design and Implementation of a Power-Proportional Web Cluster", Proceedings of the 1st ACM SIGCOMM workshop on Green Networking, pp. 15-22, New Delhi, India, August 2010.

[24] A. Shribman, B. Hudzia, "Pre-copy and post-copy VM live migration for memory intensive applications", Proceedings of the 18th International Conference on Parallel processing workshops Euro-Par'12, Greece, August 27-31, 2012

[25] A. Varga, R. Hornig, "An Overview of the OMNeT++ Simulation Environment", Proceedings of the 1st International Conference on Simulation Tools and techniques for Communications, Networks and Systems & Workshops, article no. 60, Marseille, France, March 2008

[26] A. W. Malik et al., "CloudNetSim++: A Toolkit for Data Center Simulations in OMNeT++", 11th Annual High Capacity Optical Networks and Emerging/Enabling Technologies, Charlotte, NC, 2014, pp. 104-108.

[27] A. W. Malik, S. U. Khan, "Data Center Modeling and Simulation Using OMNeT++", Chapter from Handbook on Data Centers, pp. 839-855, March 2015.

[28] A.K. Sidhu, S. Kinger, " Analysis of Load Balancing Techniques in Cloud

Computing", International Journal of Computers and Technology, Volume 4 No.2, pp. 737-741, March-April, 2013

[29] Ad-Hoc Advisory group, "ICT for Energy Efficiency", Brussels, 24.10.2008 http://ec.europa.eu/information_society/activities/sustainable_growth/docs/consultations/advisory_group_reports/ad-hoc_advisory_group_report.pdf

[30] Advanced Configuration and Power Interface Specification, November 2013 http://www.acpi.info/spec.htm

[31] Average electricity consumption per electrified household. https://www.wec-indicators.enerdata.eu/household-electricity-use.html

[32] B. Heller, S. Seetharaman, P. Mahadevan, Y. Yiakoumis, P. Sharma, S. Banerjee, N. McKeown, "ElasticTree: saving energy in data center networks", Proceedings of the 7th USENIX Conference on Networked Systems Design and Implementation, San Jose, California, April 2010.

[33] B. Kepes, "Understanding the Cloud Computing Stack: SaaS, PaaS, IaaS", White paper, July 2016 https://support.rackspace.com/white-paper/understanding-the-cloud-computing-stack-saas-paas-iaas/

[34] B. Li, J. Li, J. Huai, T. Wo, Q. Li, L. Zhong, "EnaCloud: An Energy-Saving Application Live Placement Approach for Cloud Computing Environments", IEEE International Conference on Cloud Computing, Bangalore, 2009, pp. 17-24

[35] B. Yang, X. Xu, F. Tan, D. Park, "An Utility-Based Job Scheduling Algorithm for Cloud Computing Considering Reliability Factor", International Conference on Cloud and Service Computing (CSC), pp.95-102, 12-14 December 2011.

[36] C. A. Waldspurger, "Memory Resource Management using VMware ESX Server", Proceedings of the 5th symposium on Operating Systems design and implementation, vol. 36, issue SI, pp. 181-194, winter 2002.

[37] C. Chou, D. Wong, L. N. Bhuyan, "DynSleep: Fine-Grained Power Management for a Latency-Critical Data Center Application", proceedings of the International Symposium on Low Power Electronics and Design ISLPED '16, pp. 212-217, San Francisco, USA, August 2016.

[38] C. Chou, L. Bhuyan, "A Multicore Vacation Scheme for Thermal-Aware Packet Processing", 33rd International Conference on Computer Design ICCD, pp. 565-572, New York, 2015.

[39] C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, A. Warfield, "Live Migration of Virtual Machines", Proceedings of the 2nd conference on Symposium on Networked System Design & Implementation NSDI 2005, Volume 2, pp. 273-286.

[40] C. D. Patel, C. E. Bash, R. Sharma, M. Beitelmal, R. Friedrich, "Smart Cooling of Data Centers", Proceedings of the International Electronic Packaging Technical Conference and Exhibition ASME 2003, pp. 129-137, Hawaii, USA, July 2013

[41] C. F. Chou, L. Golubchik, J. C. S. Lui, "Multiclass Multiserver Threshold-Based Systems: A Study of Non- instantaneous Server Activations", IEEE Transactions on Parallel and Distributed Systems, vol. 18, no.1, pp. 96-110, January 2007.

[42] C. Huankai, F. Wang, N. Helian, G. Akanmu, " User-priority guided Min-Min

scheduling algorithm for load balancing in cloud computing", National Conference on Parallel Computing Technologies, pp. 1-8, 21-23 February 2013

[43] C. Minkenberg, G. Rodriguez, "Trace-Driven co-simulation of High-performance Computing Systems using OMNeT++", International Conference on Simulation Tools and Techniques, 2009.

[44] C. Peoples, G. Parr, S. McClean, P. Morrow, B. Scotney, "Energy Aware Scheduling across 'Green' Cloud Data Centers", IFIP/IEEE International Symposium on Integrated Network Management (IM 2013), pp. 876-879, 27-31 May 2013.

[45] C. Wu, R. Chang, H. Chan, "A Green Energy-Efficient Scheduling Algorithm using the DVFS Technique for Cloud Datacenters", Journal of Future Generation Computer Systems, vol. 37, pp. 141-147, July 2014.

[46] C. Y. Li, C. Peng, P. Cheng, S. Lu, X. Wang, F. Ren, T. Wang, "An Energy Efficient Perspective on Rate Adaptation for 802.11n NIC", IEEE Transactions on Mobile Computing, vol. 15, no. 6, pp. 1333-1347, June 2016.

[47] D. Borgetto, H. Casanova, G. Da Costa, J. Pierson, "Energy-aware Service Allocation", Journal of Future Generation Computer Systems, vol. 28, issue 5, pp. 769-779, May 2012.

[48] D. Cavdar, F. Alagoz, "A Survey of Research on Greening Data Centers", 2012 IEEE Global Communications Conference GLOBECOM, Anaheim, CA, 2012, pp. 3237-3242.

[49] D. Chaudhary, B. Kumar, "Analytical Study of Load Scheduling Algorithms in Cloud Computing", International Conference on Parallel, Distributed and Grid Computing (PDGC), pp. 7-12, 11-13 December 2014.

[50] D. Chaudhary, R.S. Chhillar, "A New Load Balancing Technique for Virtual Machine Cloud Computing Environment", International Journal of Computer Applications, Volume 69, No. 23, pp. 37-40, May 2013

[51] D. Dharwar, S. S. Bhat, V. Srinivasan, D. Sarma, P. K. Banerjee, "Approaches Towards Energy-Efficiency in the Cloud for Emerging Markets" IEEE International Conference on Cloud Computing in Emerging Markets CCEM, pp. 1-6, Bangalore, 2012

[52] D. Kakadia, N. Kopri, V. Varma, "Network-aware Virtual Machine Consolidation for Large Data Centers", Proceedings of the 3rd International Workshop on Network-Aware Data Management, article no. 6, 2013.

[53] D. Meisner, B. T. Gold, T. F. Wenisch, "PowerNap: eliminating server idle power", Proceedings of the 14th International Conference on Architectural Support for programming languages and operating systems, pp. 205-216, USA, March 7-11, 2009.

[54] D. Meisner, T. F. Wenisch, "DreamWeaver: architecture support for deep sleep", Proceedings of the 17th International Conference on Architectural support for programming Languages and Operating Systems, pp. 313-324, London, England, March 3-7, 2012

[55] D. Niyato, S. Chaisiri, L. B. Sung, "Optimal Power management for Server Farm to Support Green Computing", Proceedings of the 9th IEEE/ACM International Symposium on Cluster Computing and the Grid, pp. 87-91, May 2099.

[56] E. Bugnion, S. Devine, M. Rosenblum, J. Sugerman, E. Y. Wang, “Bringing Virtualization to the x86 Architecture with the Original VMware Workstation”, ACM Transaction on Computer Systems (TOCS), Volume 30, Issue 4, Article No. 12, November 2012.

[57] E. Le Sueur, G. Heiser, “Dynamic Voltage and Frequency Scaling: The Laws of Diminishing Returns”, Proceedings of the International Conference on Power Aware Computing and Systems, pp. 1-8, Vancouver, BC, Canada, 2010.

[58] E. Le Sueur, G. Heiser, “Slow Down or Sleep, That is the Question”, Proceedings of the USENIX Annual Technical Conference, Portland, OR, June 2011.

[59] E. N. Elnozahy, M. Kistler, R. Rajamony, “Energy-efficient Server Clusters”, Proceedings of the 2nd International Conference on Power0aware Computer Systems, pp. 179-197, 2002

[60] ESXi Hypervisor http://www.vmware.com/products/esxi-and-esx.html

[61] G. Buttazzo, “Scalable Applications for Energy-Aware Processors”, in Embedded Software, 2002, pp. 153-165.

[62] G. Dhiman, T. S. Rosing, “System-Level Power Management using Online Learning”, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 28, no. 5, pp. 676-689, May 2009.

[63] G. Sarmila, N. Gnanambigai, P. Dinadayalan, “ Survey on Fault Tolerant Load Balancing Algorithms in Cloud Computing”, 2nd International Conference on Electronics and Communication Systems (ICECS), pp. 1715-1720, 26-27 February 2015.

[64] G. von Laszewski, L. Wang, A. J. Younge, X. He, “Power-aware Scheduling of Virtual Machines in DVFS-enabled Clusters”, IEEE International Conference on Cluster Computing and Workshops, New Orleans, LA, 2009, pp. 1-10.

[65] H. H. Kramer, V. Petrucci, A. Subramanian, E. Uchoa, “A Column Generation Approach for Power-Aware Optimization of Virtualized Heterogeneous Server Clusters”, Journal of Computers and Industrial Engineering, vol. 63, issue 3, pp. 652-662, November 2012.

[66] H. Mehta, P. Kanungo, M. Chandwani, “Decentralized Content Aware Load Balancing Algorithm for Distributed Computing Environments”, Proceedings of the International Conference and Workshop on Emerging Trends in Technology ICWET, pp. 370-375, 2011.

[67] H. Mi, H. Wang, G. Yin, Y. Zho, D. Shi, L. Yuan, “Online Self-Reconfiguration with performance Guarantee for Energy-Efficient Large-Scale Cloud Computing Data Centers”, IEEE International Conference on Services Computing, pp. 514-521, Miami, Florida, 2010.

[68] H. Nakada, T. Hirofuchi, H. Ogawa, S. Itoh, “Toward Virtual machine Packing Optimization Based on genetic Algorithm”, Chapter at Distributed Computing, Artificial Intelligence, Bioinformatics, Self-Computing and Ambient Assisted Living, vol. 5518, pp. 651-654

[69] https://azure.microsoft.com

[70] https://cloud.google.com/appengine/

[71] I. De Falco, R. Del Balio, E. Tarantino, R. Vaccaro, “Improving search by incorporating evolution principles in parallel Tabu search”, IEEE World Congress on Computational Intelligence, Proceedings of the First IEEE Conference on Evolutionary Computation, volume 2, pp. 823-828, 27-29 June 1994.

[72] I. Habib, “Virtualization with KVM”, Linux Journal, Article no. 8, vol. 2008, Issue 166, February 2008.

[73] I. Mitrani, “Service Center trade-offs between Customer Impatience and Power Consumption”, Journal of Performance Evaluation, vol. 68, issue 11, pp. 1222-1231, November 2011.

[74] I. Takouna, W. Dawoud, C. Meinel, “Accurate Multicore Processor Power Models for Power-aware Resource Management”, Proceedings of the IEEE 9th International Conference on Dependable, Autonomic and Secure Computing (DASC), pp. 419-426, Sydney, 2011.

[75] I.A. Mohialdeen, “Comparative Study of Scheduling Algorithms in Cloud Computing Environments”, Journal of Computer Science, pp. 252-263, 2013.

[76] I.J.B.F. Adan, J. Wessels, and W.H.M. Zijm. Analysis of the symmetric shortest queue problem. Stochastic Models, 6:691–713, 1990

[77] IBM Corporation, “IBM’s Strategy for Dynamic Infrastructure”, 2008

[78] J. Adhikari, S. Patil, “Double Threshold Energy Aware Load Balancing in Cloud Computing”, Fourth International Conference on Computing, Communications and Networking Technologies, pp. 1-6, 4-6 July 2013.

[79] J. C. S. Lui, L. Golubchik, “Stochastic Complement Analysis of Multi-Server Threshold Queues with Hysteresis”, Journal of Performance Evaluation, vol. 35, issues 1-2, March 1999, pp. 19-48.

[80] J. D. C. Little, “A Proof of the Queuing Formula: $L = \lambda W$”, Operations Research, 9(3), pp. 383-387, May 1961.

[81] J. G. Koomey, “Growth in Data Center Electricity Use 2005 To 2010”, August 1, 2011
http://www.analyticspress.com/datacenters.html

[82] J. Galloway, K. Smith, J. Carver, “An Empirical Study of Power Aware Load Balancing in Local Cloud Architectures”, 9th International Conference on Information Technology: New Generations, pp. 232-236, 16-18 April 2012.

[83] J. Guo, Z. Y. Lei, “A Kind of Wormhole Attack Defense Strategy of WSN based on Neighbor Nodes Verification”, IEEE 3rd International Conference on Communication Software and Networks, pp. 564-568, 2011.

[84] J. Hamilton, “Cooperative Expandable Micro-Slice Servers (CEMS): Low Cost, Low Power Servers for Internet-Scale Services”, proceedings of Conference on Innovative Data Systems Research, 2009.

[85] J. Jeong, S. Kim, H. Kim, J. Lee, E. Seo, “Analysis of Virtual Machine live-migration as a method for Power-Capping”, The Journal of Supercomputing, vol. 66, issue 3, pp. 1629-1655, December 2013.

[86] J. Keilson, “Green’s Function Methods in Probability Theory”, Journal of the London Mathematical Society, Volume s1-42, Issue 1, pp. 368-369, 1967.

[87] J. Koomey, “Growth in data center electricity use 2005 to 2010”, report by Analytical Press, completed at the request of The New York Times.

[88] J. Kremer, “Cloud Computing and Virtualization”, White paper

[89] J. Onisick, “Data Center 101: Server Virtualization”, September 2010, http://www.definethecloud.net/tag/data-center-virtualization/

[90] K. Al Nuaimi, N. Mohamed, M. Al Nuaimi, J. Al-Ajroodi, “A Survey of Load Balancing in Cloud Computing: Challenges and Algorithms”, Second Symposium on Network Cloud Computing and Applications (NCCA), pp. 137-142, 3-4 December 2012.

[91] K. Bilal, S. U. Khan, J. Kolodziej, L. Zhang, K. Hayat, S. A. Madani, N. Min-Allah, L. Wang, D. Chen, “A Comparative Study of Data Center Network Architectures”, Proceedings of the 26th European Conference on Modeling and Simulation ECMS, 2012.

[92] K. Chow, B. Liu, “On Mapping Signal Processing Algorithms to a Heterogeneous Multiprocessor System”, International Conference on Acoustics, Speech, and Signal Processing, Volume 3, pp. 1585-1588, 14-17 April 1991.

[93] K. Church, A. Greenberg, J. Hamilton, “On delivering Embarrassingly Distributed Cloud Services”, Proceedings of 7th ACM workshop on Hot Topics in Networks ‘Hotnets’, Calgary, Canada, October 6-7, 2008

[94] K. H. Kim, A. Beloglazov, R. Buyya, “Power-aware Provisioning of Virtual Machines for Real-Time Cloud Services”, Journal of Concurrency and Computation: Practice and Experience, vol. 23, issue 13, pp. 1491-1505, September 2011.

[95] K. Kant, “Data Center Evolution, A Tutorial on State of the art, issues, and challenges”, Journal of Computer Networks: The International Journal of Computer and Telecommunications Networking Volume 53 Issue 17, December, 2009, Pages 2939-2965.

[96] K. S. Kim, “Integration of OMNeT++ Hybrid TDM/WDM-PON Models into INET Frameworks”, International Workshop on OMNeT++, 2011.

[97] K. S. Rao, P. S. Thilagam, “Heuristics based server consolidation with residual resource defragmentation in cloud data centers”, Journal of Future generation Computer Systems, vol. 50, issue C, pp. 87-98, September 2015.

[98] K. Z. Ibrahim, S. Hofmeyr, C. Iancu, E. Roman, “Optimized pre-copy live migration for memory intensive applications”, Proceedings of the International Conference for High Perfomance Computing, Networking, Storage and Analysis, Article No. 40, Seattle, Washington, November 12-18, 2011

[99] L. A. Barroso, U. Hoelzle, “The Case for Energy-Proportional Computing”, Journal of Computer, vol. 40, issue 12, pp. 33-37, December 2017.

[100] L. A. Barroso, U. Hoelzle, “The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines”, Text Book.

[101] L. Bertini, J. C. B. Leite, D. Mosse, “Power Optimization for Dynamic Configuration in Heterogeneous Web Server Clusters”, Journal of Systems and Software, vol. 83, issue 4, pp. 585-598, April 2010.

[102] L. C. Zhong, J. M. Rabaey, A. Wolsitz, “An Integrated Data-Link Energy Model for Wireless Sensor Networks”, IEEE International Conference on Communications, vol.

7, pp. 3777-3783, 2004.

[103] L. Gkatzikis, I. Koutsopoulos, “Migrate or Not? Exploiting Dynamic Task Migration in Mobile Cloud Computing Systems”, IEEE Wireless Communications 2013, pp. 24-32

[104] L. He, D. Zou, Z. Zhang, C. Chen, H. Jin, S. A. Jarvis, “Developing resource consolidation frameworks for moldable virtual machines in Clouds”, Journal of Future Generation Computer Systems, vol. 32, pp. 69-81, March 2014

[105] L. Kleinrock, “A Vision for the Internet”, ST Journal for Research , Volume 2, pp 4-5, November 2005

[106] L. Kolb, A. Thor, E. Rahm, “Load Balancing for MapReduce-based Entity Resolution”, proceedings of 28th International Conference on Data Engineering (ICDE), pp. 618-629, 2012.

[107] L. M. Silva, J. Alonso, P. Silva, J. Torres, A. Andrzejak, “Using Virtualization to improve Software Rejuvenation”, Sixth IEEE International Symposium on Network Computing and Applications NCA 2007, Cambridge, MA, 2007, pp. 33-44.

[108] L. Wang, H.J. Siegel, V.P. Roychowdhury, A. Maciejewski, “Task Matching and Scheduling in Heterogeneous Computing Environments Using a Genetic-Algorithm-Based Approach”, Journal of Parallel and Distributed Computing, Volume 47, Issue 1, pp. 8-22, November 1997

[109] L. Wang, J. Tao, M. Kunze, A. C. Castellanos, D. Kramer and W. Karl, "Scientific Cloud Computing: Early Definition and Experience," 10th IEEE International Conference on High Performance Computing and Communications HPCC 2008, Dalian, pp. 825-830

[110] L. Zhu, J. Chen, Q. He, D. Huang, S. Wu, “ITC-LM: A smart Iteration-Termination Criterion Based Live Virtual Machine Migration”, Proceedings of the 10th IFIP International Conference on Network and Parallel Computing NPC 2013, vol. 8174, pp. 118-129

[111] M. Armbrust, A. Fox, R. Griffith, A.D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, M. Zaharia, “A view of Cloud Computing”, Communications of the ACM, Volume 53 Issue 4, April 2010. Pages 50-58.

[112] M. Aruna, D. Bhanu, R. Punithagowri, “A Survey on Load Balancing Algorithms in Cloud Environment”, International Journal of Computer Applications, Volume 82, No. 16, pp. 39-43, November 2013.

[113] M. E. Gebrehiwot, S. A. Aalto, P. Lassila, “Optimal Sleep-state Control of Energy-Aware M/G/1 Queues”, Proceedings of the 8th International Conference on Performance Evaluation Methodologies and Tools, pp. 82-89, Bratislava, Slovakia, December 2014.

[114] M. Gusat, R. Birke, C. Minkenberg, “Delay-Based Cloud Congestion Control”, IEEE Global Telecommunications Conference GLOBECOM 2009, Honolulu, 2009, pp. 1-8.

[115] M. Herlich, N. Bredenbals, H. Karl, “Delayed (de-)activation in Servers with a Sleep Mode”, Journal of Sustainable Computing: Informatics and Systems, vol. 10, pp. 48-55, June 2016.

[116] M. K. Bhatti, C. Belleudy, M. Auguin, "Hybrid Power Management in Real Time Embedded Systems: an Interplay of DVFS and DPM Techniques", Journal of Real-Time Systems, vol. 47, issue 2, pp. 143-162, March 2011.

[117] M. Kozuch, M. Satyanarayanan, "Internet Suspend/Resume", proceedings of the Fourth IEEE Workshop on Mobile Computing Systems and Applications WMCSA '02, page 40, June 20-21, 2002

[118] M. Lin, A. Wierman, L. L. H. Andrew, E. Thereska, "Dynamic Right-sizing for Power-proportional Data Centers", IEEE/ACM Transactions on Networking (TON), vol. 21, issue 5, pp. 1378-1391, October 2013.

[119] M. Mashaly, "Modeling and Evaluation of Power Management Strategies in ICT Systems", M.Sc. Thesis, 2011.

[120] M. Mashaly, P.J. Kühn, "Modeling and Analysis of Virtualized Multi-Service Cloud Data Centers with Automatic Server Consolidation and Prescribed Service Level Agreements", 41st IEEE conference on Local Computer Networks (LCN), November 7-10, 2016, Dubai, UAE.

[121] M. R. Hines, U. Deshpande, K. Gopalan, "Post-copy live migration of Virtual Machines", ACM SIGOPS Operating Systems Review, vol. 43, Issue 3, pp. 14-26, July 2009.

[122] M. Shiraz, A. Gani, R. H. Khokhar, R. Buyya, "A Review on Distributed Application Processing Frameworks in Smart Mobile Devices for Mobile Cloud Computing", IEEE Communications Surveys and Tutorials, vol. 15, no. 3, pp. 1294-1313, 2013.

[123] M. Sindelar, R. K. Sitaraman, P. Shenoy, "Sharing-aware algorithms for virtual machine colocation", Proceedings of the 23rd annual ACM symposium on Parallelism in Algorithms and Architectures, pp. 367-378, San Jose, California, USA, 4-6 June 2011

[124] M. Stansberry, "2014 Data Center Industry Survey", survey by Uptime Institute, 2014

[125] N. Cordeschi, M. Shojafar, D. Amendola, E. Baccarelli, "Energy-Efficient adaptive networked datacenters for the QoS support of real-time applications", The Journal of Supercomputing, vol. 71, issue 2, pp. 448-478, February 2015.

[126] N. Kansal, I. Chana, "Cloud Load Balancing Techniques: A Step Towards Green Computing", International Journal of Computer Science (IJCSI), Volume 9, Issue 1, pp. 238-246, January 2012.

[127] N. Mostafa, M. Mashaly, M. Ashour, "Modeling and Simulation of Energy-Efficient Cloud data Centers", IEEE Conf. 2nd International Conference on Engineering and Technology ICET, Cairo, Egypt, April 19-20, 2014

[128] N. Sarkar, R. Membarth, "Modeling and Simulation of IEEE 802.11g using OMNeT++", Text Book, January 2010.

[129] O. C. Ibe, J. Keilson, "Multi-Server Threshold Queues with Hysteresis", Journal of Performance Evaluation, Vol. 21, Issue 3, January 1995, pp. 185-213.

[130] O. Mostafa, "Implementation of a Test Bed for Testing Load balancing Algorithms in Cloud DCs", Bachelor Thesis at the Faculty of Information Engineering and Technology, German University in Cairo, 2016.

[131] OMNeT++ Discrete Event Simulator Documentation https://omnetpp.org/documentation

[132] P. Berham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, A. Warfield, “Xen and the Art of Virtualization”, Proceedings of the 19th ACM symposium on Operating Systems Principles SOSP 2003, pp. 164-177.

[133] P. J. Kühn, “Systematic Classification of Self-Adapting Algorithms for Power-Saving Operation Modes of ICT Systems”, 2nd International Conference of Energy-Efficient Computing and Networking (e-Energy 2011), New York, USA, May 30- June 1, 2011

[134] P. J. Kühn, M. Mashaly, “Dynamic Load Balancing in Cloud Networks – Resource Management, Modeling and Performance”, Proc. of the second Conf. on Energy-Efficient Data Centers (E 2DC 2013), Berkeley, CA., May 21, 2013.

[135] P. Mahadevan, P. Sharma, S. Banerjee, P. Ranganathan, “Energy Aware Network Operations”, Proceedings of the 28th IEEE International Conference on Computer Communications Workshops INFOCOM ‘09, pp. 25-30, Rio de Janeiro, Brazil, April 2009

[136] P. Mell, T. Grance, “The NIST definition of Cloud Computing”, NIST, 2011

[137] P. Riteau, C. Morin, T. Priol, “Shrinker: Improving live migration of virtual clusters over WANs with distributed data duplication and content-based addressing”, Proceedings of the 17th International Conference on Parallel Processing (Euro-Par’11), Volume Part 1, pp. 431-442, Springer-Verlag Berlin, 2011

[138] P.J. Kuehn, M. Mashaly, “Multi-Server, Finite Capacity Queuing System with Mutual Overflow”, Proc. 2nd European Teletraffic Seminar (ETS), Karlskrona, Sweden, Sept. 30.0ct.2013

[139] P.J. Kühn, M. Mashaly “Performance of Self-Adapting Power-Saving Algorithms for ICT Systems”, IFIP/IEEE Symposium on Integrated Network and Service Management (IM 2013), Ghent, May 27-30, 2013.

[140] P.J. Kühn, M. Mashaly, “Automatic Energy Efficiency Management of Data Center Resources by load-dependent server activation and sleep modes”, Elsevier Journal of Ad Hoc Networks, Volume 25, Part B, Pages 497-504, 2014.

[141] P.J. Kühn, M. Mashaly, “Load Balancing in Distributed Cloud data Center Configurations – performance and Energy-Efficiency”, submitted to the 6th International Workshop on Energy Efficient Data Centers E2DC 2017, Hong Kong

[142] P.J. Kühn, M. Mashaly, “Modeling and Performance Evaluation of Self-Adapting Algorithms for the Optimization of Power-Saving Operation Modes”, Proc. 1st European. Teletraffic Seminar (ETS), Poznan, Poland, February. 14-16, 2011.

[143] Q. Deng, D. Meisner, A. Bhattacharjee, T. F. Wenisch, R. Bianchini, “CoScale: Coordinating CPU and Memory System DVFS in Server Systems”, Proceedings of the 45th Annual IEEE/ACM International Symposium on Microarchitecture MICRO-45, pp 143-154, Vancouver, Canada, December 2012.

[144] Q. Zhang, L. Cheng, R. Boutaba, “Cloud Computing: state-of-the-art and research challenges”, Journal of Internet Services and Applications, Volume 1, Issue 1, pp 7-18, May 2010

[145] R. Ahmad, A. Gani, S. Hamid, M. Shiraz, F. Xia, S. Madani, “Virtual machine migration in cloud data centers: a review, taxonomy, and open research issues”, Journal of Supercomputing, volume 71, Issue 7, pp 2473-2515, July 2015

[146] R. Basmadjian, F. Niedermeier, H. De Meer, “Modelling and Analyzing the Power Consumption of Idle Servers”, 2012 Sustainable Internet and ICT for Sustainability (SustainIT), Pisa, 2012, pp. 1-9

[147] R. Birke, G. Rodrigez, C. Minkenberg, “Towards Massively Parallel Simulations of Massively Parallel High-Performance Computing Systems”, International Conference on Simulation Tools and Techniques, 2012.

[148] R. Buyya, C. Vecchiola, S. T. Selvi, “Mastering Cloud Computing”, Text Book, 1st Edition, May 2013

[149] R. Kohavi, R. M. Henne, D. Sommerfield, “Practical guide to controlled experiments on the web: listen to your customers not to the hippo”, Proceedings of the 13th ACM SIGKDD international conference on knowledge discovery and data mining, pp. 959-967, San Jose, California, USA, August 12-15, 2007

[150] R. Lee, B. Jeng, “Load Balancing Tactics in the Cloud”, International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery, pp. 447-454, 10-12 October 2011

[151] R. W. Ahmad, A. Gani, S. H. Ab.Hamid, M. Shiraz, A. Yousafzai, F. Xia, “A survey on virtual machine migration and server consolidation frameworks for cloud data centers”, Journal of Network and Computer Applications, vol. 52, pp. 11-25, June 2015

[152] R. Zhou, F. Liu, C. Li, T. Li, “Optimizing Virtual Machine Live Storage Migration in Heterogeneous Storage Environment”, Proceedings of the 9th ACM SIGPLAN/SIGOPS International Conference on Virtual execution environments, Houston, Texas, USA, March 16-17, 2013.

[153] S. Fayek, “Building a GUC Mini Cloud Test Bed”, Bachelor Thesis at the Faculty of Information Engineering and Technology, German University in Cairo, 2015.

[154] S. Kumar, V. Talwar, V. Kumar, K. Schwan, “Vmanage: Loosely coupled platform and virtualization management in data centers”, Proceedings of the 6th International Conference on Autonomic Computing ICAC, June 15-19, Barcelona, Spain, 2009.

[155] S. Liu, S. Ren, G. Quan, M. Zhao, S. Ren, “Profit Aware Load Balancing for Distributed Cloud Data Centers” IEEE 27th International Symposium on Parallel and Distributed Processing, Boston, MA, 2013, pp. 611-622.

[156] S. Liu, S. Ren, G. Quan, S. Ren, “Profit Aware Load Balancing for Distributed Cloud data Centers”, IEEE 27th International Symposium on Parallel ^ Distributed Processing (IPDPS), May 2013

[157] S. Mulay, S. Jain, “Enhanced Equally Distributed Load Balancing Algorithm for Cloud Computing”, International Journal of Research in Engineering and Technology, Volume 2, Issue 6, pp. 976-980, June 2013.

[158] S. Nakarani, C. Tovey, “On Honey Bees and Dynamic Server Allocation in Internet Hosting Centers”, Journal of Adaptive Behavior, Volume 12, Issue 3-4, pp. 223-240, 2004.

[159] S. Nedevschi, L. Popa, G. Iannacconem S. Ratnasamy, D. Wetherall, “Reducing Network Energy Consumption via Sleeping and Rate-Adaptation”, Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation, pp.

323-336, San Francisco, California, April 2008.

[160] S. Niles, P. Donovan, "Virtualization and Cloud Computing: Optimized Power, Cooling and Management Maximizes Benefits", White Paper 118, Revision 3, Schneider Electric. http://www.schneider-electric.ch/documents/company/white-papers/whitepaper_virtualization_and_cloud_computing_en.pdf

[161] S. Osman, D. Subhraveti, G. Su, J. Nieh, "The design and implementation of Zap: a system for migrating computing environments", Proceedings of the 5th Operating Systems Design and Implementation (OSDI 2002), Boston, MA, December 2002.

[162] S. Osman, D. Subhraveti, G. Su, J. Nieh, "The design and implementation of Zap: a system for migrating computing environments", Proceedings of the 5th Symposium on Operating systems design and implementation OSDI 2002, vol. 36 issue SI, pp. 361-376, Winter 2002.

[163] S. Parsa, R. Entezari-Maleki, " RASA: A New Task Scheduling Algorithm in grid Environment", World Applied Sciences Journal (Special Issue of Computer and IT), Volume 7, pp. 152-160, 2009

[164] S. Patel, R. Patel, H. Patel, S. Vahora, " CloudAnalyst: A survey of Load Balancing Policies", International Journal of Computer Applications, volume 117, pp. 21-24, May 2015

[165] S. Srikantaiah, A. Kansal, F. Zhao, "Energy aware Consolidation for Cloud Computing", Proceedings of the 2008 Conference on Power Aware Computing and Systems

[166] S. Wang, K. Yan, W. Liao, S. Wang, "Towards a Load Balancing in a Three-level Cloud Computing Network", 3rd IEEE International Conference on Computer Science and Information Technology (ICCSIT), Volume 1, pp. 108-113, 9-11 July 2010.

[167] S.B. Shaw, A.K. Singh, "A Survey on Scheduling and Load Balancing Techniques in Cloud Computing Environment", 5th International conference on Computer and Communication Technology (ICCCT), pp. 87-95, 26-28 September 2014

[168] T. Braun, H. Siegel, N. Beck, L. Boloni, M. Maheswaran, A. Reuther, J. Robertson, M. Theys, B. Yao, D. Hensgen, R. Freund, "A Comparison of Eleven Static Heuristics for Mapping a Class of Independent Tasks onto Hetereogeneous Distributed Computing Systems", Journal of Parallel and Distributed Computing, Volume 61, Issue 6, pp. 810-837, June 2001.

[169] T. C. Ferreto, M. A. Netto, R. N. Calheiros, C. A. De Rose, "Server Consolidation with migration control for virtualized data centers", Journal of Future Generation Computer Systems, vol. 27, issue 8, pp. 1027-1034, October 2011.

[170] T. Gunarathne, T. Wu, J. Qiu, G. Fox, "MapReduce in the Clouds for Science", proceedings of the 2nd International Conference on Cloud Computing Technology and Science (CloudCom), pp. 565-572, November/December 2010.

[171] T. Hirofuchi, H. Nakada, S. Itoh, S. Sekiguchi, "Reactive Consolidation of Virtual Machines Enabled by Postcopy Live Migration", proceedings of the 5th International Workshop on Virtualization technologies in Distributed Computing, pp. 11-18, San Jose, California, USA, June 2011

[172] T. Mastelic, A. Oleksiak, H. Claussen, I. Brandic, J. Pierson, A. V. Vasilakos, "Cloud

Computing: Survey on Energy Efficiency", ACM Computing Surveys Journal (CSUR), vol. 47, issue 2, article no. 33, January 2015.

[173] T. Mudge, U. Holzle, "Challenges and Opportunities for Extremely Energy-Efficient Processes", IEEE Micro, vol. 30, no. 4, pp. 20-24, July-August 2010.

[174] T. Wood, G. T. Levin, P. Shenoy, P. Desnoyers, E. Cecchet, M. D. Corner, "Memory buddies: exploiting page sharing for smart colocation in virtualized data centers", proceedings of the ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments, 2009, pp.31-40.

[175] T.-J. Lu, W. Yue, T. Hasegawa, "A Mutual Overflow System with Simultaneous Occupation of Resources", Journal of the Operations Research Society of Japan, Vol. 41, No. 1, March 1998, pp. 81-90.

[176] U. Deshpande, K. Keahey, "Traffic-Sensitive Live Migration of Virtual Machines", 15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, Shenzhen, 2015, pp. 51-60.

[177] U. Deshpande, U. Kulkarni, K. Gopalan, "Inter-rack live Migration of Multiple Virtual Machines", Proceedings of the 6th International workshop on Virtualization Technologies in Distributed Computing Data, pp. 19-26, Delft, The Netherlands, June 18-22.

[178] V. Avelar, D. Azevedo, A. French, "PUE: A Comprehensive Examination of the Metric", The Green Grid, white paper no.49, 2014 http://www.thegreengrid.org/

[179] V. C. Emeakaroha, M. A. S. Netto, R. N. Calheiros, I. Brandic, R. Buyya, C. A. F. De Rose, "Towards autonomic detection of SLA violations in Cloud Infrastructures", Journal of Future Generation Computer Systems, vol. 28, issue 7, pp. 1017-1029, July 2012.

[180] V. J. Maccio, D. G. Down, "On Optimal Policies for Energy-Aware Servers", Journal of Performance Evaluation, vol. 90, pp. 36-52, August 2015.

[181] V. K. M. Raj, R. Shriram, "Power Management in Virtualized Datacenter – A Survey", Journal of Network and Computer applications, vol. 69, Issue C, pp. 117-133, July 2016.

[182] V. Medina, J. M. Garcia, "A survey of migration mechanisms of virtual machines", Journal of ACM Computing Surveys (CSUR), vol. 46, issue 3, Article no. 30, 2014.

[183] V. Shrivastana, P. Zerfos, K. Lee, H. Jamjoom, Y. Liu, S. Banerjee, "Application-aware Virtual Machine Migration in Data Centers", Proceedings of IEEE INFOCOM 2011, New York, pp 66-70.

[184] V. Valancius, N. Laoutaris, L. Massoulie, C. Diot, P. Rodriguez, "Greening the internet with nano data centers", Proceedings of the 5th International conference on Emerging networking experiments and technologies, Rome, Italy, December 1-4, 2009

[185] V. Venkatachalam, M. Franz, "Power Reduction Techniques for Microprocessor Systems", ACM Computer Surveys (CSUR), vol. 37, no. 3, pp. 195-237, 2005.

[186] Vcenter Server https://www.vmware.com/products/vcenter-server.html

[187] VMware PowerCLI Documentation https://www.vmware.com/support/developer/PowerCLI/

[188] vSphere http://www.vmware.com/products/vsphere.html

[189] W. Fisher, M. Suchara, J. Rexford, “Greening Backbone Networks: Reducing Energy Consumption by shutting off cables in bundled links”, Proceedings of the 1st ACM SIGCOMM workshop on Green Networking, pp 29-34, New Delhi, India, August 2010.

[190] www.vmware.com

[191] X. Evers, “ A Literature Study on Scheduling in Distributed Systems”, October 1992

[192] Y. Chen, F. Xia, D. Shang, A. Yakovlev, “Fine-grain Stochastic Modeling of Dynamic Power Management Policies and Analysis of their Power-Latency Tradeoffs”, IET Software, vol. 3, no. 6, pp. 458-469, December 2009.

[193] Y. Gao, H. Guan, Z. Qi, Y. Hou, L. Liu, “A multi-objective ant colony system algorithm for virtual machine placement in cloud computing”, Journal of Computer and System Sciences, vol. 79, issue 8, pp. 1230-1242, December 2013.

[194] Y. Lua, Q. Xiea, G. Kliotb, A. Geller, J. Larus, A. Greenberg, “ Join-Idle-Queue: A Novel Load Balancing Algorithm for Dynamically Scalable Web Services”, International Journal on Performance Evaluation, Volume 68, Issue 11, pp 1056-1071, November 2011.

[195] Y. Wang, X. Wang, “Virtual Batching: Request Batching for Server Energy Conservation in Virtualized Data Centers”, IEEE Transactions on Parallel and Distributed Systems, vol. 24, no. 8, August 2013.

[196] Y. Zhao, J. Zhang, Y. Ji, W. Gu, “Routing and Wavelength Assignment Problem in PCE-Based Wavelength-Switched Optical Networks”, Journal of Optical Communications and Networking, vol. 2, issue 4, pp. 196-205, 2010.

[197] Z. Cao, S. Dong, “An energy-aware heuristic framework for virtual machine consolidation in Cloud computing”, Journal of Supercomputing, vol. 69, issue 1, pp. 429-451, July 2014

[198] Z. Ren, B. H. Krogh, R. Marculescu, “Hierarchical Adaptive Dynamic Power Management”, IEEE Transactions on Computers, vol. 54, issue 4, pp. 409-420, April 2005.

[199] Z. Wang, C. Mccarthy, X. Zhu, P. Ranganathan, V. Talwar, “Feedback Control Algorithms for Power Management of Servers”, Proceedings of the 3rd International Workshop on Feedback Control Implementation and Design in Computing Systems and Networks (FeBid), Annapolis, MD 2008.

[200] Z. Zhang, Z. Lu, Q. Chen, X. Yan, L. R. Zheng, “COSMO: CO-Simulation with MATLAB and OMNeT++ for Indoor Wireless Networks”, IEEE Global Telecommunications Conference GLOBECOM 2010, Miami, Florida, 2010, pp. 1-6.

Acknowledgements

I would like to express my most sincere gratitude to my supervisor Prof. Paul J. Kühn for I would’ve never done this project without his invaluable knowledge, ideas, guidance and support through the past 7 years. I am grateful for giving me the chance to become an external PhD student at University of Stuttgart and for his patience to supervise me remotely. He has taught me everything I know about conducting research and teaching, and for that I am eternally grateful. I would also like to thank Prof Hermann de Meer and Prof. Andreas Kirstädter for reviewing this thesis within a short time frame.

I would also like to thank the German University of Cairo and DAAD for funding my research visits and allowing me to conduct parts of my research work at the University of Stuttgart. Also for the company EMC2 for the equipment used at the GUC mini-cloud lab where our experimental work was conducted. Throughout my study and employment years at the GUC I have met many people whom I am grateful for their support; my supportive colleagues: Hadil, Yasmine, Maria and Hana, my students whom I benefited a lot from supervising their bachelor theses: Andrew, Nada, Mary, Omar, Shady, Karim and Marwan. I am most grateful to my lecturers who have taught and guided me: Dr. Tallal, Dr. Ashour, Dr. Amr and especially Prof. Yasser Higazi who has been my greatest support since my first day of employment at the GUC.

Through the long journey of the PhD I have been blessed to have a supporting and understanding family. I wouldn’t have been able to finish this work without the support of my Mother, for she has always supported and motivated me, allowed me to have endless working hours and always encouraged me to be the best person I can. She has always been my role model in all life aspects: a great loving mother, a successful team leader and a reputable Professor. Also I am thankful for my Father who is the greatest father one could have, for being our strength during the tough times and the backbone of our family. I am forever grateful to them for the quality life and education they gave me and for their endless encouragement and support to me and my brother who has also been my greatest motivator through each step of my life. I have been lucky and thankful to have an understanding husband who encouraged me and put up with my long study hours and research trips. Finally my gratitude and prayers go out to my late grandmother for her endless love and encouragement.