Pós-Graduação em Ciência da Computação

Erico Augusto Cavalcanti Guedes

PERFORMABILITY ANALYSIS OF WEB CACHE SERVER CLUSTERS APPLIED TO SERVER VIRTUALIZATION

M.Sc. Dissertation

Federal University of Pernambuco
posgraduacao@cin.ufpe.br
www.cin.ufpe.br/~posgraduacao

RECIFE
2015

Federal University of Pernambuco
Center for Informatics
Graduate in Computer Science

Erico Augusto Cavalcanti Guedes

PERFORMABILITY ANALYSIS OF WEB CACHE SERVER CLUSTERS APPLIED TO SERVER VIRTUALIZATION

A M.Sc. Dissertation presented to the Center for Informatics of Federal University of Pernambuco in partial fulfillment of the requirements for the degree of Master of Science in Computer Science.

Advisor: *Paulo Romero Martins Maciel*

RECIFE
2015

Catalogação na fonte
Bibliotecária Joana D'Arc Leão Salvador CRB4-532

G924p Guedes, Erico Augusto Cavalcanti.
Performability analysis of web cache server clusters applied to server virtualization / Erico Augusto Cavalcanti Guedes. – Recife: O Autor, 2015.
126 f.: fig., tab.

Orientador: Paulo Romero Martins Maciel.
Dissertação (Mestrado) – Universidade Federal de Pernambuco. CIN, Ciência da Computação, 2015.
Inclui referências e apêndices.

1. Ciência da computação. 2. Avaliação de desempenho. 3. Virtualização. I. Maciel, Paulo Romero Martins (Orientador). II. Titulo.

004 CDD (22. ed.) UFPE-MEI 2015-073

Dissertação de Mestrado apresentada por **Erico Augusto Cavalcanti Guedes** à Pós Graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco, sob o título **"Performability Analysis of Web Cache Server Clusters Applied to Server Virtualization**", orientada pelo **Prof. Paulo Romero Martins Maciel** e aprovada pela Banca Examinadora formada pelos professores:

_____

Prof. Eduardo Antonio Guimarães Tavares
Centro de Informática/UFPE

_____

Prof. Almir Pereira Guimarães
Departamento de Computação / UFAL

_____

Prof. Paulo Romero Martins Maciel
Centro de Informática / UFPE

Visto e permitida a impressão.
Recife, 27 de fevereiro de 2015.

_____

Profa. Edna Natividade da Silva Barros
Coordenadora da Pós-Graduação em Ciência da Computação do
Centro de Informática da Universidade Federal de Pernambuco.

To my family: Adelair, my mother, Bruno, my brother, and Ariane, one half of me and who really taught me the meaning of 100% of availability.

Acknowledgements

I would like to thank my family, friends, professors, and my advisor Paulo Maciel, for all the support over these years.

I am very grateful to my mother, for her unconditional support and presence throughput all my academical life.

A special thanks to my friend Patricia Endo for her constant support, to professor, and my friend, Marco Domingues, who helped me at the initial stages of my graduate and research studies, and to Luís Eduardo Tenório Silva, for his insights regarding OS Virtualization.

Thanks to the professors and coworkers of the Federal Institute of Education, Science, and Technology of Alagoas - Campus Palmeira dos Índios, among them: Paulo Wagner, Rodrigo Lustosa, Carlos Antonio, Ronaldo Dionísio, and Rafael Antonello.

A thanks to my cousins Jefferson, Kleber, and Clériston, and to my aunt Maria do Socorro.

To all my dear friends of Miraflôres Building, thank you so much.

Thanks for collaboration to several colleagues I made at CIn/UFPE and MODCS Research Group: Jean Teixeira, Kádna Camboim, Carlos Julian, Kalil Bispo, Airton Pereira, Danilo Mendonça, Júlio Mendonça, and Demétrio Junior.

Finally, I would like to thank IFAL/PIn, CIn/UFPE, and CNPq for the support provided in this dissertation.

Resumo

O crescimento do poder de processamento dos computadores tem renovado o interesse na área de virtualização. Através da técnica de virtualização, um único servidor tem seus recursos particionados em diversos ambientes de execução separados, aumentando a eficiência no uso dos recursos computacionais. Para aplicar virtualização de servidores, é preciso selecionar uma abordagem de virtualização, que pode variar desde a completa emulação do hardware, como a técnica de virtualização total, até alternativas mais leves, como a virtualização baseada em contêineres, que particiona os recursos físicos de um servidor em múltiplas instâncias isoladas de processamento. Independentemente da abordagem de virtualização, a aplicação de virtualização de servidores leva a um aumento da sobrecarga do sistema. Desta forma, a melhoria da eficiência supracitada tem gerado um novo desafio: mesmo com a sobrecarga adicional proveniente da virtualização, é possível fornecer um serviço em ambiente virtualizado com desempenho similar, ou até mesmo superior, ao de infraestruturas não virtualizadas? Esta dissertação propõe uma avaliação de desempenho baseada em medições para investigar quantitativamente o comportamento de um agrupamento de servidores web cache em ambientes virtualizados e não virtualizados. Os primeiros resultados analisados não somente endorsaram o impacto sobre o desempenho, mas também revelaram uma degradação da disponibilidade no fornecimento do serviço web cache em agrupamentos virtualizados. Para combater esse problema, a metodologia proposta aplica redução de sobrecarga com o objetivo de alcançar 5 noves de disponibilidade para o fornecimento do serviço web cache em ambientes virtualizados. Além disso, a aplicação da técnica de experimentos fatoriais fracionados resultou em métricas de desempenho favoráveis em ambientes virtualizados. Os resultados dos estudos de caso que foram conduzidos permitiram identificar cenários nos quais os agrupamentos de servidores web cache virtualizados se comportaram de maneira favorável em comparação com os não virtualizados, produzindo recomendações sobre o uso otimizado dos agrupamentos virtuais investigados.

Palavras-chave: Avaliação de Desempenho. Virtualização de Servidores. Virtualização baseada em Contêineres. Análise de Performabilidade

Abstract

The growth of computers processing power have been renewing the interest in virtualization area. Through virtualization technique, a single server has its resources partitioned into multiple separated execution environments, increasing computer resources usage efficiency. To apply server virtualization, it is necessary to select a virtualization approach, which may differ from complete emulation of entire underlying hardware, as full virtualization, to lightweight alternatives, such as container-based virtualization, that partitions the physical machine resources into multiple isolated user-space instances. Regardless of virtualization approach, application of server virtualization drives to increase of system overhead. So, aforementioned efficiency improvement has led to a new challenge: even with additional overhead of virtualization, is it possible to provide a service on virtualized environment with similar, or even better, performance of non-virtualized infrastructures? This dissertation proposes a measurement performance evaluation to quantitatively investigate the behavior of web cache server clusters on virtualized and non-virtualized environments. First analyzed results not only endorsed impact over performance, but also revealed availability degradation on providing virtualized web cache server clusters. To tackle this issue, proposed methodology focused on overhead reduction to reach five 9's of availability when providing web cache server cluster on virtualized infrastructure. Additionally, applying screening fractional factorial experiment technique resulted in favorable performance metrics on virtualized environments. Results of executed use cases enable to identify scenarios in which virtualized web cache server clusters perform better in comparison with non-virtualized ones, producing recommendations about optimized usage of investigated virtualized clusters.

Keywords: Performance Evaluation. Server Virtualization. Container-based Virtualization. Performability Analysis

List of Figures

2.1 Web Cache Cluster Architectures 29
2.2 OpenVZ Architecture 39
2.3 Containers networking: modes of operation 40

3.1 Supported web cache servers: physical machines(Nodes 01, 02, and 03), containers(CT), virtual machines(VM), and multiple-instances(I) 44
3.2 TISVEP startup flow for CTS experiment 45
3.3 TISVEP halt flow for CTS experiment 46

4.1 Methodology for Performability Analysis 54
4.2 Summarized sequence of performed experiments 56
4.3 Testbed for initial non-virtualized warm up experiments 57
4.4 Full Virtualization Initial Analysis 61
4.5 Virtualization Testbed Configuration: 3 VMs instantiated on 1 Physical Machine 62
4.6 Full Virtualization Final Analysis 62
4.7 Container-based Operating System Virtualization Initial Analysis 63
4.8 Container-based Operating System Virtualization Intermediate Analysis 64
4.9 COS Virtualization Final Results: one-to-one CTs and Hard Disks relationship 64
4.10 Container-based Operating System Virtualization Final Analysis 65
4.11 Performance metric: non-virtualized baseline storage presented better hit ratio than bind-mounted COS virtualization 66
4.12 Performance metrics. (a) . Similar behavior was observed to (a) response time and (b) throughput. 67
4.13 Hit ratio means 95% confidence intervals for factor's combination with mesh architecture 69
4.14 Hit Ratios comparative: 3GB and 30GB scenarios 70
4.15 Response Times comparative: 3GB and 30GB scenarios 71
4.16 Throughput comparative: 3GB and 30GB scenarios 72
4.17 Maximum number of CTs per HD for High Availability 73
4.18 Testbed configuration with maximum number of CTs for high availability . . . 74
4.19 Performance metrics for 180GB of total storage cache capacity with 18 containers 75
4.20 OpenVZ modes of operation analysis through web cache performance metrics behaviour 78
4.21 Bandwidth savings for OpenVZ network modes of operation 79
4.22 Performance metrics for 900GB of total storage cache capacity 80
4.23 Cache storage capacity over time - Containers (CTs) configuration 80

4.24 Relative Changes of performance metrics for 900GB of total storage cache capacity 81
4.25 CPU_CLK_UNHALTED: 3PMs x 9CTs on 3PMs - 900GB of total cache space 83
4.26 L2_CACHE_MISSES: 3PMs x 9CTs on 3PMs - 900GB of total cache space . . 84
4.27 L2 cache miss hardware events behaviour for 900GB of total cache capacity . . 85
4.28 Full Spool performance metrics - 1.8TB of total cache capacity 87
4.29 CPU_CLK_UNHALTED sum for 9CTs x 3PMs with 1.8TB of cache capacity . 89
4.30 L2_CACHE_MISSES sum for 9CTs x 3PMs with 1.8TB of cache capacity . . 91
4.31 L2 cache miss hardware events behaviour for 1.8TB of total cache capacity . . 92

B.1 NMS Monitoring Terminal: gathering real time commands results from remote servers. Specifically, the result of execution of the partition formatting command `fdisk` is depicted. It is used on the remote servers to configure the web cache server spools. 126

List of Tables

2.1 Service availability and downtime ratings 23
2.2 Number of failures X Resulting Availability 24

4.1 Warm up availability experiment parameters and result 57
4.2 95% Confidence Intervals for Hit Ratio(%) Means 60
4.3 95% Confidence Intervals for Response Time (ms) Means 60
4.4 95% Confidence Intervals for Throughput (resp/s) Means 60
4.5 Comparison of performance metrics 66
4.6 Initial configuration of experiments during tuning performance 67
4.7 Web Cache observed factors and levels 68
4.8 Availability for each evaluated scenario 68
4.9 Configuration for virtualized scenario: aiming at maximum number of CTs per HD for High Availability 73
4.10 Bandwidth savings for 18 containers and 180GB of cache 76
4.11 Byte Hit Ratio for 18 containers and 180GB of cache on cluster 76
4.12 OpenVZ modes of operation experiment design 77
4.13 900GB experiment design 79
4.14 Saved bandwidth(TB) 82
4.15 Maximum RSS for non-virtualized full spool experiments 86
4.16 Bandwidth savings for 9CTs on 3PMs with 1.8TB of total cache storage 88
4.17 Bandwidth savings for 3PMs with 1.8TB of total cache storage 88
4.18 CPU_CLK_UNHALTED comparison: 900GB x 1.8TB of spool storage capacity. Values on columns 900GB and 1.8TB are multiplied by 10^{12} 89
4.19 L2_CACHE_MISSES summed values comparison: 900GB x 1.8TB of spool storage capacity. Values on columns 900GB and 1.8TB are multiplied by 10^{12} . 91

5.1 Related Works 99

List of Acronyms

API - *Application Program Interface*.

ARP - *Address Resolution Protocol*.

BHR - *Byte Hit Ratio*.

CARP - *Cache Array Routing Protocol*.

CDF - *Cumulative Distribution Function*.

CI - *Confidence Interval*.

COS - *Container-based Operating System*.

CPU - *Central Process Unit*.

CT - *Container*.

DAS - *Directed-Attached Storage*.

GDFS - *Greedy-Dual Size Frequency*.

HD - *Hard Disk*.

HPC - *High Performance Computing*.

HR - *Hit Ratio*.

HTCP - *Hypertext Cache Protocol*.

HTTP - *Hypertext Transfer Protocol*.

ICP - *Internet Cache Protocol*.

IP - *Internet Protocol*.

IRQ - *Interrupt Request Queue*.

KVM - *Kernel-based Virtual Machine*.

LB - *Lower Bound*.

LFU - *Least Frequently Used*.

LFUDA - *Least Frequently Used with Dynamic Aging*.

LRU - *Least Recently Used*.

LXC - *Linux Containers*.

MAC - *Media Access Control*.

NFS - *Network File System*.

NIC - *Network Interface Card*.

NMS - *Network Management System*.

OOM - *Out Of Memory*.

OS - *Operating System*.

PC - *Program Counter*.

PM - *Physical Machine*.

RAID - *Redundant Array of Inexpensive Disks*.

RAM - *Random Access Memory*.

RPM - *Revolutions Per Minute*.

RSS - *Resident Set Size*.

SMP - *Simultaneous Multiprocessing*.

SSH - *Secure Shell*.

TCP - *Transmission Control Protocol*.

TISVEP - *Tuning Infrastructure for Server Virtualization Experiment Protocol*.

URL - *Uniform Resource Locator*.

UB - *Upper Bound*.

VCPU - *Virtual Central Process Unit*.

VCS - *Virtual Computer Systems*.

VE - *Virtual Environment*.

VM - *Virtual Machine*.

VPS - *Virtual Private Server*.

VMM - *Virtual Machine Monitor*.

WWW - *World Wide Web*.

Contents

1 Introduction **16**

1.1 Motivation 18
1.2 Objectives 19
1.3 An overview 19
1.4 Dissertation Structure 21

2 Background **22**

2.1 Applied Performability Concepts 22
2.1.1 Performance 24
2.1.2 Software Hang 25
2.1.3 Profiling 26
2.2 Web Cache Service 27
2.2.1 Replacement Policies 27
2.2.2 Web Cache Clusters 29
2.2.3 Intercache Protocols 30
2.3 Workload for Web Cache Service 32
2.3.1 Probability Distributions 33
2.4 Server Virtualization 34
2.4.1 Applied Virtualization Solutions 37
2.4.1.1 KVM 37
2.4.1.2 OpenVZ 37

3 TISVEP - Tuning Infrastructure for Server Virtualization Experiment Protocol **41**

3.1 TISVEP Features 42
3.1.1 Implementation Technology 43
3.1.2 Start up and halt flows 44
3.2 TISVEP Static Entities 47
3.2.1 TISVEP config.properties file 47
3.3 TISVEP Messages 51

4 Performability Analysis of Virtual Web Cache Services **53**

4.1 Performability Analysis Methodology 53
4.2 Application of Proposed Methodology: High Availability 56
4.2.1 Non-virtualized initial experiments 56
4.2.2 Full Virtualization High Availability Results 60
4.2.3 Container-based Virtualization High Availability Results 63

4.3 Tuning Performance Phase . . . 65
4.3.1 Initial Metrics Analysis . . . 65
4.3.2 Phase 1: screening fractional factorial experiment . . . 68
4.3.3 Phase 2: applying most favorable performability parameters . . . 69
4.4 Case study 1: CTs per HD . . . 73
4.4.1 Two CTs per hard disk: discussion of the results . . . 73
4.5 Case study 2: Analysis of OpenVZ network modes of operation . . . 76
4.6 Case study 3: Scaling and profiling web cache server . . . 78
4.6.1 Performability analysis with 900GB of storage . . . 79
4.6.2 Profiling web cache server clusters . . . 82
4.7 Case study 4: Full spool performability analysis . . . 85
4.7.1 Profiling web cache server clusters with storage capacity of 1.8TB . . . 88
4.8 Summary of recommendations . . . 92

5 Related Works **94**
5.1 Performance Evaluation of Server Virtualization through Measurements . . . 94
5.2 Experiments Automation . . . 97
5.3 Discussion . . . 98

6 Conclusions **100**
6.1 Future Works . . . 101

References **104**

Appendix **112**

A Codes **113**
A.1 Web-polygraph Pareto Distribution Implementation . . . 113
A.2 TISVEP config.properties file instance . . . 114

B Specification of TISVEP Messages **117**
B.1 Infrastructure Configuration Messages . . . 117
B.2 Service Configuration Messages . . . 119
B.3 Benchmarking Configuration Messages . . . 120
B.4 Experiment Monitoring and Management Messages . . . 121
B.5 TISVEP Operation . . . 124

1 Introduction

Interest in virtualization has resurged since computers have enough processing power to use virtualization as a technique of partitioning the resources of a single server into multiple separated execution environments (MEDINA; GARCÍA, 2014). Computing resources of providers are pooled to serve multiple clients through a multi-tenant model, with dynamically assigned and reassigned resources, whether physical or virtual, following to consumer demand. Such characteristic is called resource pooling, one of essential features of cloud computing. (MELL; GRANCE, 2011). With virtualized infrastructures, resources can be readily allocated and released in an elastic way, even automatically, to scale rapidly outward and inward proportional with necessary demand.

Since resumption of virtualization usage, efforts have been directing to improve the performance of infrastructures that employ virtualization. Such efforts aim to achieve a similar performance of native non-virtualized environments with the remarkable challenge of trying to overcome it, even with one additional abstraction layer: the hypevisor.

Also called Virtual Machine Monitor (VMM), the hypervisor places on top of physical resources, enabling their collective use by multiple systems, aiming to increase efficiency of such physical resource. Fortunately, the software and hardware technologies have significantly developed to bring the overheads of virtualization down, so that virtualization is considered acceptable for almost every workload (MCDOUGALL; ANDERSON, 2010). Even with such improvements, hypervisor virtualization solutions, such as full virtualization and paravirtualization, when applied to reach similar performance of non-virtualized environments, have presented overhead levels that prevent this goal. A third option, called container-based operating system virtualization, claims to have lower performance overheads of virtualization solutions. It creates multiple isolated user-space instances, partitioning physical machines resources and directly using OS system calls, while concurrent hypervisor solutions are required to provide a complete emulation of entire underlying hardware. On web environment, virtualization refers to operating system abstraction, yielding virtualized servers. An outstanding advantage of using virtual servers is the possibility of a more granular control of how the hardware resources are used. The technology, thus, reduces the number of needed servers, thereby saving money and using

resources more efficiently.

Both hypervisor and container-based virtualization solutions provide a great opportunity to build elastically scalable systems, which are able to provide additional capability with minimum costs. A solid approach, called server virtualization, efficiently permits software to be separated from the physical hardware, presenting logical servers of computing resources, in which they can be accessed with more advantages over the original configuration (TOPE et al., 2011). Server virtualization enables an improving technique, called server consolidation. Originally deployed on different servers, it aggregates multiple services and applications on one physical server, allowing to reduce the power consumption of underlying facilities and to resolve hardware underutilization.

At same time that server consolidation enables cost reduction of equipments, it also promotes a further challenge: reduce virtualization overheads to keep agreed performance levels of provided virtualized services. Additionally, although a substantial amount of research works on virtualization copes with performance analysis, no results about impacts over availability in conjunction with performance analysis through measurements evaluations were identified: even if any obstacles occur, service should be provided smoothly so users can use cloud computing services. Availability is intrinsically defined as the probability that a system is operating satisfactorily at any point in time when used under stated conditions (STAPELBERG, 2009). Reach high availability having influence on profit losses avoidance and negative consequences from service down times.

As described by Zavarsky et al. (TOPE et al., 2011), server virtualization has created some growing problems and disorder on virtualized components, such as unresponsive virtualized systems, crashed virtualized server, and misconfigured virtual hosting platforms. From these variations for the normal operational conditions, one key factor that can impact on server virtualization availability is a phenomenon of unresponsiveness, called software hang, that is a major threat to software dependability (SONG; CHEN; ZANG, 2010). It causes time intervals of system's inability to produce appropriate reacting, resulting in ungiven responses.

To struggle against such undesirable behavior, iterative infrastructure refining methodology is used. It aims to reduce overheads and improve availability by tackling unresponsiveness phenomenon, drives to an innovative approach on server virtualization performability analysis. The methodology applied in this dissertation uses measurements to evaluate performance of an I/O bound network service when failures are considered. On such scenarios, the impacts of server virtualization overhead on availability, through measurements, were not established by previous known related researches.

To drive this research, a web cache cluster will be used as aforementioned I/O bound network service. Web cache clusters will enable to evaluate the behavior of both intensive disk I/O and network I/O operations. Web cache systems are applied on network environments to improve the quality of users access when requests are performed. Web caches have been deploying to reduce network traffic and provide better response times. A web cache cluster is a

group of separated caching proxies configured to act like a single server (WESSELS, 2001). It is a powerful solution to enhance capability and availability of web cache system. Many networks present cache clusters to serve more objects and provide redundant services. There are many advantages on using clustering machines as a single server, such as: scalability, maximization of resource efficiency (while caching more objects), avoiding single point of failure when using some specific caching configurations (like transparent proxying (AGUILAR; LEISS, 2006)), reliability, and reduce response times (DUAN; GU, 2010).

The adopted methodology was based on replication of experiments, aiming at measure performance metrics of cluster environments. It requires an impactful amount of configuration. The tuning volume is even wider on virtualized infrastructures. To overcome this issue, employed methodology applies automation of experiments replications. It is also desired to produce a statistically significant number of samples, aiming to output relevant conclusions about analyzed performability factors. Such approach is resembling to proposed experiments automation performed by Huber et al. (HUBER et al., 2010), where each set of experiments is executed multiple times to characterize the impact of considered factors.

Based on previous stated arguments, this research aims to use measurement performance evaluation technique to verify the possibility of executing a virtualized web cache cluster with similar availability and performance levels, or even higher, when compared with native non-virtualized infrastructure. A methodology that applies intensive effort in overheading reduction of virtualized environments was used. It employs system capacity optimization using replication of automated experiments.

1.1 Motivation

Internet and infrastructure service providers conduct researches to minimize computational resources costs on their data centers. With the rise in energy costs and the advancement of computer processing power and memory capacity, running separate, under-utilized server hardware with specific roles is no longer a luxury (AHMED, 2014), and thus technologies such as server virtualization and server consolidation have arisen.

Even within the narrower context of computer I/O, focused on this research, virtualization has a long, diverse history, exemplified by logical devices that are deliberately separate from their physical instantiations. For example (WALDSPURGER; ROSENBLUM, 2012), in computer storage, a logical unit number (LUN) represents a logical disk that may be backed by anything from a partition on a local physical drive to a multidisk RAID volume exported by a networked storage array.

Server and network virtualization enable to truly create a virtual infrastructure. Virtual data centers, virtual storage systems, and virtual desktop PCs are just a few real-world applications where virtualization is being heavily used. As one of the core components, virtual machine monitor affects the performance of virtualization systems to a great extent, so it's important to

measure and analyze the performance of systems when provided over virtual machine monitors.

The initial goal of this research would be to accomplish a performance evaluation between virtualized and non-virtulalized environments, motived by the possibility of execute services over former infrastructure with similar, or even higher, performance of latter one. Nevertheless, a deeper issue took place: initial experiments results revealed a usual incapacity of analyzed virtualized system to response on expected time. Such incapacity had been attributing to virtualization infrastructure: in non-virtualized scenarios, such problem did not arise. Thus, the initial performance evaluation evolved to a performability analysis, due to system's unavailability.

1.2 Objectives

The main objective of this research is to analyze quantitatively the behavior of web cache server cluster on virtualized and non-virtualized environments, proposing approaches to provide such service on virtualized environment with similar, or even higher, performance than on non-virtualized one.

The following collection of specific objectives were identified:

- to distinguish workload characteristics of web cache traffics based on previous academic researches;
- to identify workload generation tools that best fit to web cache environment stimulus, based on their workload characteristics;
- to identify behavior differences between web services over virtualized and non-virtualized environments through conducted experiments;
- set up and manage of private cloud computing test environment to execution of measurement experiments;
- to implement an automated mechanism for experiments configuration in order to minimize the work due to the high number of required tuning steps;
- to compare and to explain why some metrics produce different behaviors under virtualized and non-virtualized scenarios;
- to investigate which factors produce behavior differences between virtualized and non-virtualized web cache server clusters.

1.3 An overview

This section presents an overview of the applied activities and methods aimed at reaching the proposed goals of the dissertation.

To perform a quantitative analysis of the behavior of web cache server clusters, the first activity carried out was to conduct a state of the art study about web cache server behavior, intending to identify which probability distributions best fit to web objects' popularity pattern and size. In this context, the Arlitt and Williamson research (ARLITT M.F.; WILLIAMSON, 1997) presented a workload characterization study for Internet Web servers. Through collected data from six different web server access logs, three from academic environments, two from scientific research institutions, and one from a commercial Internet provider, and subsequent study about file sizes, it could be observed that file size distributions are heavy-tailed, i.e., distribution matches well with the Pareto distribution. Regarding web objects' popularity pattern, Breslau et. al (BRESLAU et al., 1999), through the analysis of six traces from proxies at academic institutions, corporations and Internet Service Providers, found that the distribution of page requests generally follows a Zipf-like distribution.

Aware of the distributions that should be used, a workload generation tool was selected and configured to excite the web cache server cluster with an I/O bound workload. Two web cache server clusters were configured: a non-virtualized one, that was used to yield the baseline results, and a virtualized one, whose behavior was investigated.

Two different virtualization infrastructures were analyzed: a full virtualization one, based on a KVM hypervisor managed by OpenNebula (OPENNEBULA, 2014), and a container one, based on OpenVZ managed by Proxmox (PROXMOX, 2014). The first series of virtualized experiments were conducted in the KVM based infrastructure, aiming at investigating the performance metrics of the web cache cluster. Their results presented a high level of unavailability. It was detected that a KVM-based virtualized web cache server cluster was unable to answer due to the overload of the full virtualization infrastructure, resulting in the phenomenon of unresponsiveness.

The observed issue of unresponsiveness, tackled in this research, was explored by Song, Chen, and Zang (SONG; CHEN; ZANG, 2010). Software hang, a phenomenon of unresponsiveness, is still a major threat to software dependability. Their work presented a survey of hang-related bugs on three server-side and one client-side applications. 5.58% of studied software hangs were related to carelessly configuring software, and 16.74% were due to the environment, caused by unexpected environments that applications were depending on. So, precise software configuration and environment set up were deeply explored recommendations in this research, as approaches for the avoidance of the unresponsiveness phenomenon.

From these first results, the availability of the web cache server cluster began to be monitored. For non-virtualized cluster, no unresponsiveness failures were observed. Based on this observation, a target availability of five 9's was established as a goal for virtualized environments. So, a fair comparison between both non-virtualized and virtualized cluster might be conduced, and the five 9's of availability was stated as a requirement for the performance analysis. Several techniques of overhead reduction were investigated and applied to reach this goal. The scenarios that were evaluated through conducted experiments, and that presented the

required five 9's of availability, had their performance metrics studied and discussed through several case studies.

The procedure of overhead reduction enables to reach five 9's of availability, however a large number of configuration activities must be performed, so that executing them without automation will increase the runtime in a prohibitive way. Also the monitoring process, as well as storage of the results, must be automated to reduce the time spent in the execution of experiments. The chosen alternative to automate the experiments was the development of a lightweight solution. The TISVEP (Tuning Infrastructure for Server Virtualization Experiment Protocol) protocol was design, implemented and subsequently applied to automatically manage the execution of the experiments.

1.4 Dissertation Structure

The remaining parts of the dissertation are organized as follows:

Chapter 2 presents fundamental concepts about performability, all necessary details about explored web cache server functionalities, including its related workload characterization, different clustering architectures, replacement policies and inter-cache communication protocols. Concepts about virtualization technology are also explored. Together, this triad constitutes the foundations of current research.

Chapter 3 presents TISVEP - Tuning Infrastructure for Server Virtualization Experiment Protocol. It was developed as a solution to automate experiments replications, whose execution time costs, due to amount of required configuration tunings, reach a prohibitive level due to the amount of replications that would be performed.

Chapter 4 presents adopted methodology for performability analysis of both studied virtualized and non-virtualized web cache cluster infrastructures, the complete made actions to reach high availability for web cache server cluster on virtualized environments, as well as performance results on these high available virtualized scenarios, compared with baseline non-virtualized configuration. Furthermore, case studies and results obtained from automated experiments are also discussed. Covered scenarios enable to: analyze scalability of containers for provisioning of high available web cache server cluster; profile hardware events to investigate causes of some better performance metric levels on virtualized environments; and solutions of high available full storage capability of web cache server clusters on testbed infrastructure.

Chapter 5 presents the related works, as well as a comparative discussion between them and this work.

Chapter 6 draws some conclusions, shows reached contributions and possible future works.

2

Background

This chapter presents the theoretical basis over which the present Master's dissertation was developed. First, performability concepts are held, highlighting the concepts of availability interval, User-Perceived Service Availability. Afterwards, web cache proxy service, and its features, are presented. Lastly, server virtualization and cloud computing are discussed, completing the necessary knowledge to understand the fundamentals that compound presented work.

2.1 Applied Performability Concepts

The term performability was originally used by Meyer (MEYER, 1980), mainly to reflect attributes like reliability and other associated performance attributes like availability, and maintainability. Meyer states that if the performance of a computing system is "degradable", performance and reliability issues must be dealt with simultaneously in the process of evaluating system effectiveness. For this purpose, a unified measure, called performability was introduced and the foundations of performability evaluation and modeling are established. For performability analysis, both availability as reliability metrics can be combined with performance metrics aiming at assess the system capacity of deliver its service. As described in (VARGAS; BIANCO; DEETHS, 2001), depending on a customer's situation, system requirements may favor reliability or availability.

Reliability is the continuity of correct service. It measures the ability of a system to function continuously without interruptions. The academic definition of reliability, `R(t)`, is the probability of a system performing its intended function over a given time period, `t` (AVIZIENIS; LAPRIE; RANDELL, 2001).

On the other hand, availability is the readiness for correct service. It is intrinsically defined as the probability that a system is operating satisfactorily at any point in time when used under stated conditions, where the time considered includes the operating time and the active repair time.

One example of a customer's situation: reliability must have a higher precedence in cases where systems are placed in remote areas and maintenance is expensive. Nevertheless, the

availability is always required (AVIZIENIS; LAPRIE; RANDELL, 2001).

Availability ratings are commonly quantified as the number of nine's. Five 9's, also called by **high availability**, is a common expectation availability for critical systems. Table 2.1 presents the maximum service downtime for traditional availability ratings (BAUER; ADAMS, 2012).

Table 2.1: Service availability and downtime ratings

Number of 9's	Service Availability (%)	System Type	Annualized Down Minutes	Practical Meaning
1	90	Unmanaged	52596.00	Down 5 weeks per year
2	99	Managed	5259.60	Down 4 days per year
3	99.9	Well managed	525.60	Down 9 hours per year
4	99.99	Fault tolerant	52.60	Down 1 hour per year
5	99.999	High availability	5.26	Down 5 minutes per year
6	99.9999	Very high availability	0.53	Down 30 seconds per year
7	99.99999	Ultra availability	0.05	Down 3 seconds per year

The **interval availability** (RUBINO; SERICOLA, 1993) concept is applied for estimating availability during the adopted experiments. It is a dependability measure defined as the fraction of time during which a system is operational over a finite observation period. For this research, the interval availability is measured at client-side. Conceptually, such approach is called **user-perceived service availability** (SHAO et al., 2009). The user-perceived service availability is the system availability observed by the user, with the system operational or not. The system can be working, however may be unresponsive due to, for example, overheading, yielding some time out events. Considering such scenario, the system is not working properly for the viewpoint of the users.

The following relationship model is used to calculate availability(A) during finite observation periods of adopted experiments:

$$A = \frac{up\ time}{operational\ cycle} = \frac{up\ time}{up\ time + down\ time} \tag{2.1}$$

The *operational cycle* is the overall time period of operation being investigated, whereas *up time* is the total time in which the system was working properly, to user viewpoint, during the operational cycle.

The unresponsiveness events are collected, in preset finite observation periods, summed, and used as the *down time* factor of Equation 2.1. The events in which the service answered

correctly are also collected and summed. This summed value is used as *up time* factor in the aforementioned equation. For example, consider a web application that performs requests to a web server. It is used a finite observation period of 1 hour. If 4 failures are observed, the interval availability, with the service unresponsiveness being checked every second, will be equal to:

$$A = \frac{up\ time}{up\ time + down\ time} = \frac{3596}{3596+4} = 99.888\% \quad (2.2)$$

The result of expression 2.2, A=99.888%, shows that two nines of availability were achieved. Similarly, if 3 failures are observed, A = 99.916%, i.e., three 9's of availability. An availability of 100% is reached if the service is able to answer all 3600 requests, one per second, without response failures.

Table 2.2 summarizes above results. It shows the number of failures and the resulting availability.

Table 2.2: Number of failures X Resulting Availability

Finite observation period (h)	Number of failures	Resulting availability
1	4	99.888%
1	3	99.916%
1	0	100%

The target percentage of availability on analyzed system is five nines: 99.999%, equivalent to a down time of 6.05 seconds per week.

2.1.1 Performance

Users, administrators, and designers of computer systems are all interested in performance evaluation. Their goal is to obtain or provide the highest performance at the lowest cost. This goal has resulted in continuing evolution of higher performance and lower cost systems (JAIN, 1991).

When one is confronted with a performance-analysis problem, there are three fundamental techniques that can be used to find the desired solution. These are measurements of existing systems, simulation, and analytical modeling (LILJA, 2005). The key consideration in deciding the evaluation technique is the life-cycle stage in which the system is (JAIN, 1991).

Measurements are possible only if the evaluated system already exists, or at least something similar to it. They generally provide the best results since, given the necessary measurement tools, no simplifying assumptions need to be made. This characteristic also makes results based on measurements of an actual system the most believable in comparison with the other techniques.

If the evaluated system is a new concept, analytical modeling and simulation are the appropriated techniques to be used. Analytical modeling and simulation can be used for situations where measurement is not possible, but in general it would be more convincing to others if the analytical modeling or simulation is based on previous measurement (JAIN, 1991).

The analytical modeling uses a set of mathematical descriptions of the system, such as equations and functions, to describe its behavior. Despite these models consider specific parameters of a system, they can be easily adapted to other systems. During the construction of the models, one should take into account their complexity and practicality.

A simulation of a computer system is a program written to model important features of the system under analysis. Since simulators are nothing more than computer programs, they can be easily modified to study the impact of changes made to almost any of the simulated components (LILJA, 2005).

The choice of an appropriate evaluation technique depends on the problem being solved. Some key differences among presented solutions are flexibility, cost, believability and accuracy. The flexibility of a technique is an indication of how easy it is to change the system to study different configurations. Analytical modeling and simulation are high flexible, whereas measurement technique provide information about only the specific system being measured. Regarding to the cost, that corresponds to the time, effort, and money necessary to perform the appropriate experiments, analytical modeling and simulations are the preferred techniques, in comparison with measurements, due to the simplifications that they can offer to evaluate the system under analysis. On the other hand, believability and accuracy are higher for measurement, since no simplifying assumptions need to be made (LILJA, 2005).

2.1.2 Software Hang

Almost every user of modern computer software has had the annoying experience of the unresponsiveness problem known as soft hang (WANG et al., 2008). Also known as software hang, it forms a major threat to the dependability of many software systems.

Increasing complexity of large software systems introduces several sources of non determinism, which may lead to hang failures: the system appears to be running, but part of its services is perceived as unresponsive. As described by Zavarsky et al. (TOPE et al., 2011), server virtualization has created some growing problems and disorder, such as unresponsive virtualized system.

According to Song et al. (SONG; CHEN; ZANG, 2010), 5.58% of reported software hang bugs are related to misconfiguration due to carelessly configured software; 15.88% are related to design problems, caused by application design errors, maintenance mistakes or unexpected working scenarios; and 16.84% issues related to unforeseen environments applications depending on. Such troubles sum 38.80% of total reports collected from bug databases of three web server applications.

Aiming to avoid previously experienced aforementioned issues related to applications misconfiguration and unforeseen working scenarios, automating configuration and execution of experiments, as well as posterior storage of generated results, were employed. Experiments automation is a powerful tool for minimizing appearance of human errors due to manual replication of setup procedures.

Along this research, such unresponsiveness phenomenon on web cache server provided on virtualized environments was observed. An aggressively tuning approach was mandatory to overcome such problem when aiming to reach high available web cache server clusters.

2.1.3 Profiling

There are a set of tools that computer systems experts can use to investigate the performance of the such systems. They can be categorized as providing system-wide or per-process informations (GREGG, 2013). Most of the tools are based on either counters, regarding to several statistics maintained by the operating systems, or tracing, that collects per-event data for analysis. However, there are also performance tools that are based on profiling.

A profile provides an overall view of the execution behavior of a program (LILJA, 2005). It aims at revealing interaction between software and underlying machine. Profiling tools characterizes a target by collecting samples of its behavior.

Based on how profilers collects information, they are classified in event-based or statistical profilers. Event-based profiling is triggered by any performance monitor event that occurs on the processor. Statistical profilers checks the Program Counter (PC) at regular intervals to determine the state of the executing software. Profiling can also be used on untimed hardware events, such as CPU hardware cache misses (GREGG, 2013). One main advantage of hardware events is its lightweight approach to collect execution profiles in production environments, resulting in negligible overhead.

Manufacturers of modern processors include performance event counters registers. They are used to count specific processor events, such as data-cache misses, or the duration of events, such as the number of clock cycles that an application spent outside of halt state (DEVICES, 2010). The events provided are architecture-specific. The hardware manuals available for the processors are the core reference for the available events.

Profiling can be used as a technique of performance tuning, as well as to investigate possible bottlenecks that can result in system's overheads. Several research works have been using profiling of hardware events as a technique to compare performance between virtualized and non-virtualized infrastructures (YE et al., 2010; DU; SEHRAWAT; ZWAENEPOEL, 2011; APPARAO; MAKINENI; NEWELL, 2006; APPARAO et al., 2008).

2.2 Web Cache Service

The goal of cache systems utilization is store data and programs most recently accessed, in a faster memory. This principle is used on web proxy cache systems. Web caching is a crucial technology in Internet because it represents an effective means for reducing bandwidth demands, improving web server availability, and reducing networks latencies (DUAN; GU, 2010).

Web proxy cache systems are applied on network environments to improve the quality on access from users when requests are performed. They act as application-level intermediaries for Web requests which retain copies of previously requested resources in the hope of improving overall quality of service by serving the content locally (COOPER; DILLEY, 2001). They store requested objects nearest of end user, usually inside local area network, so that if same objects are requested again, by any host belonging to the network, it will not be necessary to yield another request to the origin server, since a copy of considered object is already stored on the web cache server (WESSELS, 2004).

Immediate advantages of employ web cache services is as much save external link bandwidth utilization as reduce latency on response times to requests from internal network users, since requested objects are served from the cache instead of the original server.

The term cache hit refers to the situations when a web client requests an object and it is stored on cache. Otherwise, it is called a cache miss. One way to measure the effectiveness of a web cache system is through hit ratio (G., 2001.)(WESSELS, 2001). Hit ratio, also called cache hit ratio (or document hit ratio) (TOTTY et al., 2002) is the percentage of requests that are satisfied as cache hits (WESSELS, 2001) , measured as the ratio of the total number of hits to the total number of all transactions. Usually, this includes both validated and invalidated hits. Validated hits can be tricky because these requests are forwarded to origin servers, incurring slight bandwidth and latency penalties. Note that the cache hit ratio tells only how many requests are hits - it doesn't tell how much bandwidth or latency was saved. Other metrics that can be analyzed to measure the benefits of web cache utilization are byte hit ratio, saved bandwidth, and response times.

Byte Hit Ratio (BHR) is a more appropriate metric to analyze bandwidth consumption, since different objects will vary in size. Byte hit ratio represents the fraction of all bytes transferred that were served from cache. It is measured by the ratio of total volume (a sum of response sizes) of hits to the total volume of all transactions. Quantitative values of BHR shows the bandwidth saved by the cache (GONZALEZ-CANETE; CASILARI; TRIVINO-CABRERA, 2006).

2.2.1 Replacement Policies

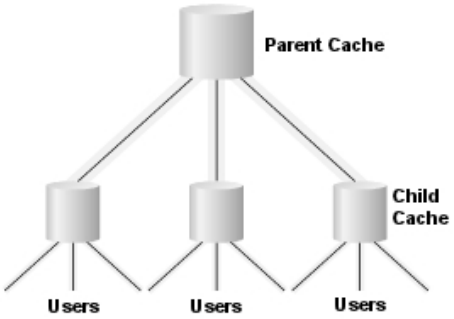
The cache replacement policy parameter determines which objects are evicted (replaced) when disk space is needed (CACHE.ORG, 2013). Replacement policies aims to maximize the hit

ratio. Usually a cache assigns some kind of value to each object and removes the least valuable ones (WESSELS, 2001). Caching researchers and developers have proposed and evaluated numerous replacement algorithms. Those discussed here are policies supported by Squid web proxy cache server (WESSELS, 2001):

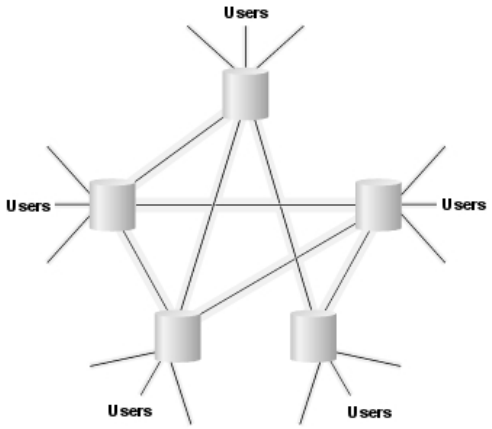
- Least Recently Used (LRU): most popular replacement algorithm used by web caches. LRU removes the objects that have not been accessed for the longest time. Every time that an object is accessed, his replacement priority is restarted, meaning that it will be the last object selected to be replaced by web proxy cache server. Squid provides two versions of LRU: one implemented of linked list (original list based LRU policy) and one implemented on heap. The latter, according to (DILLEY; ARLITT; PERRET, 1999), has a better performance, considering hit ratio and byte hit ratio, than the former.

- Least Frequently Used (LFU): similar to LRU, but instead of selecting based on time since access, the significant parameter is the number of accesses. LFU replace objects with small access counts and keeps objects with high access counts. One drawback of this algorithm is cache pollution: if a popular object becomes unpopular, it will remain in the cache a long time, preventing other newly popular objects from replacing it. One possibility to overcome this debit, Squid supports an enhanced version of Last Frequently Used algorithm, called heap LFU with Dynamic Aging (heap LFUDA) (DILLEY; ARLITT; PERRET, 1999). This variant of LFU uses a dynamic aging policy to accommodate shifts in the set of popular objects. In the dynamic aging policy, the cache age factor is added to the reference count when an object is added to the cache or an existing object is modified. This prevents previously popular documents from polluting the cache. Instead of adjusting all key values in the cache, the dynamic aging policy increments the cache age when evicting objects from the cache, setting it to the key value of the evicted object. This has the property that the cache age is less than or equal to the minimum key value in the cache.

- Greedy-Dual Size (GDS): it assigns value to cached objects based on the cost of a cache miss and the size of the object. GDS does not specify exactly what cost means, offering a great optimization flexibility. Some values that can be applied as cost are latency (time it takes to receive a response), number of packets transmitted over the network or the number of hops between the origin server and the cache. Squid uses a variant implementation of GDS, called heap GDS-Frequency (heap GDSF). It takes into account frequency of reference. Furthermore, Squid keeps popular objects with smaller size in the cache: if two cached objects are with same popularity, the object with larger size will be purged. It optimizes object hit ratio by keeping smaller popular objects. On the other hand, it achieves a lower byte hit ratio, since it evicts larger (possibly popular) objects.



(a) Hierarchical Architecture



(b) Mesh Architecture

Figure 2.1: Web Cache Cluster Architectures

2.2.2 Web Cache Clusters

A web cache cluster is a group of separated caching proxies configured to act like a single server (WESSELS, 2001). It is a potent solution to enhance capability of web cache system. Many networks presents cache clusters to serve more objects and provide redundant services. In truth, there are many advantages on using clustering machines as a single server, as: scalability, maximization of resource efficiency (while cache more objects), avoiding single point of failure when using some specific caching configurations, like transparent proxying, reliability and reduce response times (DUAN; GU, 2010).

Because of the endless growth of Internet, the Web proxy cache system using single computer could not meet the increasing client demand. Web proxy cache cluster, a typical cooperative proxy system, is a powerful solution to enhance Web proxy cache system's capacity (DUAN; GU, 2010). On local network cache, two typical architectures are applied: mesh and hierarchical. They differ on communication approach performed among nodes that composes cluster.

Caching hierarchy is the name generally given for a collection of web proxy cache servers that forward requests to one another (WESSELS, 2001). Members of a hierarchy have either parent or siblings relationship. Parent nodes can forward cache misses for their children, whereas sibling nodes are not allowed to forward them, as can be seen in Figure 2.1a. Groups of siblings nodes can not compound a hierarchy at all. In these cases, where there is not a hierarchical sense, the collection of servers is called a cache mesh, as shown in Figure 2.1b.

Although hierarchies usually yield high hit ratios for the intermediate and topmost nodes, they possess two main drawbacks: (i) the benefit (in terms of response time) for end-users is not always possible (especially if the topmost cache lies behind a slow link), (ii) the upper level nodes may become overloaded (BAKIRAS; LOUKOPOULOS; AHMAD, 2003). Mesh architectures have the merit of join equal systems homogeneously, nevertheless they present some drawbacks: (i) organizing N proxies in a mesh topology introduces scalability problems,

since the number of queries is of the order of N^2, (ii) the contents of some nodes may extensively overlap.

2.2.3 Intercache Protocols

Intercache protocols aims to enable communication among servers that compose a collection of web proxy caches. There are some different roles that a server can perform on such environment, like parent or sibling. Whatever the role, it is necessary the presence of a communication protocol that enables the exchange of information about objects stored in cache. Without communication protocols, mechanisms to query nearby caches about a given document are impracticable. Therefore, avoid using intercache protocols on web cache cluster environments precludes information exchange among nodes that compounds a cluster.

Squid provides implementation of four different intercache protocols: Internet Cache Protocol (ICP), Cache Array Routing Protocol (CARP), Hypertext Caching Protocol (HTCP), and Cache Digests. Their performance features related to hit ratio, response times and bandwidth, that allow optimization on web cache cluster environment, are discussed below, focusing on drawbacks and advantages presented on each protocol.

Internet Cache Protocol: ICP (WESSELS; CLAFFY, 2014a,b) is a lightweight protocol of URL's location. It is used to message exchange about URLs existence between neighbors caches. Nodes that composes the cache exchange requests and ICP answers to catch information about the most appropriate local to recovery objects. ICP was designed to be simple, small, and efficient. Although it is possible to maintain cache hierarchies without using ICP, the lack of ICP or something similar prohibits the existence of mechanisms to query nearby caches about a given document.

Known issues that affect ICP performance includes:

- scaling problems (COOPER; DILLER, 2014), since ICP exhibits $O(n^2)$ scaling properties, where n is the number of participating peer proxies. This can lead ICP traffic to dominate HTTP traffic within a network, resulting in a large number of messages that need to be exchanged to gather object presence on another cluster nodes. This issue represents a negative impact on response time and increasing on local bandwidth consumption, nevertheless, it does not presents drawback, to the local application of web cache cluster;

- peer selection problems (COOPER; DILLER, 2014): caching proxy peer selection in networks with large variance in latency and bandwidth between peers can lead to non-optimal peer selection. On hierarchical architecture, nodes with high bandwidth could be used as a parent and, not be used as siblings, of slower nodes. In such situation, mesh architectures are not recommended.

Cache Array Routing Protocol: CARP (VALLOPPILLIL; ROSS, 2014) was designed to address the issue of how to achieve efficient and scalable load balancing while maximizing hit ratios and minimizing latency (WESSELS, 2001). CARP use a algorithm which subdivides the objects with URLs joint in different cache nodes, with the goal to optimize the hit ratio and minimize the objects duplication between nodes that make up the grouping. It proposes to increase hit ratio by allocate and intelligently route requests for the correct URLs to any member of the proxy array. Due to the resulting sorting of requests through these proxies, duplication of cache contents is eliminated and cache hit rates can be improved. Once that main goal of CARP protocol is increasing hit ratio, it consequently improves response time and saved bandwidth.

Knowns issue that affect CARP performance includes:

- CARP is not usable in a sibling relationship: mesh architectures that use CARP will not work properly, since the complementary memories used on this architecture is fully restrictive to main features of CARP. So, hit ratio shows a clear downward trend.

Hypertext Cache Protocol: HTCP (VIXIE; WESSLES, 2014) was designed for discovering HTTP caches and cached data, managing sets of HTTP caches, and monitoring cache activity. HTCP main improvement, related to another intercache protocols, is the capacity to send full HTTP headers (not just the URL) in requests and responses. This makes it possible to improve responses made to servers, since valuable informations about objects age can be transmitted on HTCP messages. So, HTCP servers are able to reply correctly with a hit or miss.

Known issues that affect HTCP performance includes:

- necessity for additional system resources: due to the capacity of transmit HTTP headers, processing a more complex message requires additional processor power. This may affect the rate at which a cache can send and receive HTCP queries, negatively impacting on response times and hit ratios;

- test messages: HTCP includes a test message type, called TST, that aims to test for the presence of a specified content entity in a proxy cache. If HTTP header are stored on disk, it may take a significant amount of time to read them from disk, increases response times.

Cache Digest: Cache Digests (HAMILTON; ROUSSKOV; WESSELS, 2014) provide a mechanism for (primarily) WWW cache servers to cooperate without the latency and congestion problems associated with previous discussed protocols. Cache Digests are summaries of the contents of an web cache server. It contains an indication of whether or not particular URLs are in the cache in a compact format. Its using is motivated by lack of capacity of other intercache protocols in provide a prediction about presence of a object in cache. Cache servers periodically exchanges their digests with each other. Once peer objects are known, checks performed on digests are fast, eliminating the need for individual emission of queries to another peer servers. This feature represents differential behavior with respect to other intercache protocols: it is

proactive. Summaries are created through an algorithm called Bloom filter (BLOOM, 1970). It uses a collection of hash function that uses URI object and HTTP method provided by web cache server client to create keys. These keys are used for state if required object is available on some web cache server that composes cluster servers.

Positive features of cache digest on cluster environments are:

- reduce latency on searching by stored objects: Once that a faster searching method is used to state presence of cached objects, response times can be decreased;
- reduce congestion problems: previous intercache protocols, like ICP, presents transmission problems due to overload the network with request/response messages. Once that the need for such messages are eliminated, bandwidth consumption during querying the peers is saved.

Known issue that affect cache digest performance includes:

- increasing on processor power. During generation of summaries, there are peak loads of CPU utilization due to execution of hash functions used for keys generation;
- presence of peaks of bandwidth utilization: cache digest protocol uses bandwidth in proportion to the size of a cache. Furthermore, during summaries transmissions, cache digest presents a peak of bandwidth utilization, that increases as it grows memory.

2.3 Workload for Web Cache Service

An understanding of how users typically behave when interacting with a system helps to identify system features that are more useful and attractive (ALMEIDA; ALMEIDA, 2011). Such understanding can be achieved through the concept of workloads. The requests made by the users of the system are called workloads (JAIN, 1991). Formally, a workload of a system can be defined as a set of all inputs that it receives from its environment, during any given period of time.

Following (JAIN, 1991): “In order to test multiple alternatives under identical conditions, the workload should be repeatable. Since a real-user environment is generally not repeatable, it is necessary to study the real-user environments, observe the key characteristics, and develop a workload model that can be used repeatedly. This process is called workload characterization. Once a workload model is available, the effect of changes in the workload and system can be studied in a controlled manner by simply changing the parameters of the model.”

Workload characterization of web cache proxy servers was performed through study of previous research works that kept track of the operation on web servers and clients’ traffic. This survey allowed to identify features and behaviors of such servers, in order to enable the

reproduction of similar behavior artificially. Several web traffic workload characterization studies have been conducted (ARLITT M.F.; WILLIAMSON, 1997; ARLITT; WILLIAMSON, 1996; DYKES; JEFFERY; DAS, 1999), aiming to state common properties that can be measured and analyzed to yield configuration recommendations of web proxy cache servers. This careful analysis of web workloads performed on literature reveals that request streams performed to web servers follow Zipf and Zipf-like distribution (ADAMIC; HUBERMAN, 2002; BRESLAU et al., 1999), while file size distribution are heavy-tailed, like Pareto's one (ARLITT; WILLIAMSON, 1996). Workloads based on such probability distributions were applied during experiments that were conducted to analyze the behavior of selected metrics of virtual web cache cluster environment.

2.3.1 Probability Distributions

There are specific characteristics of web traffic, that were previously observed on other fields of research, like word frequencies in texts and city sizes (ADAMIC; HUBERMAN, 2002): few sites consist of millions of pages, but millions of sites only contain a handful of pages; few sites contain millions of links, but many sites have one or two; millions of users access few select sites, giving little attention to millions of others. Main findings about web traffic behavior can be extracted by observation that the requests made by web users follows zipf law. In other words, this behavior can be mathematically stated by a Zipf power law model and the frequency of accessed web pages can be expressed by Zipf probability distribution.

A careful analysis of web workloads was performed on literature, revealing that request streams follow Zipf-like distribution(BRESLAU et al., 1999)(CHLEBUS; OHRI, 2005)(ADAMIC; HUBERMAN, 2002) and file size distribution are heavy-tailed, following Pareto's distribution (ARLITT M.F.; WILLIAMSON, 1997)(PITKOW, 1998). Zipf's law, which gives rise to Zipf distribution, when applied to web traffic, states that the relative probability of a request for the i'th most popular page is proportional to 1/i, i.e., the probability of an object being requested is proportional to its ranking. Breslau et al. (BRESLAU et al., 1999) proposes the nomenclature used on this paragraph, calling Zipf's law to refer to request distributions following:

$$\Omega/i^{\alpha} \tag{2.3}$$

with α typically taking on some value less than unity. Graphically, on a log-log scale, it represents the slope of data distribution. Zipf-like probability distribution was used along experiments to adjust incoming web page requests on benchmarking tool.

The Cumulative Distribution Function (CDF) of Pareto's Distribution was used as basis to model file size provided by web servers. It is defined as:

$$F(x;a,b) = 1 - (b/x)^{a} \tag{2.4}$$

Pareto's CDF was implemented on Web Polygraph benchmarking tool, used on experimental steps, to model file size of traditional web traffic objects. The larger *a* parameter value, called *shape*, the smaller the proportion of very high file sizes.

2.4 Server Virtualization

The term server virtualization describe the ability to run an entire virtual machine, including its own operating system, on another operating system (HAGEN, 2008). To apply server virtualization, it is necessary to select a virtualization approach. By the end of the 1990s, virtualization was unheard in the x86 industry. In the middle half of the 2000s, there has been a exponential growth in the virtualization market both in terms of customer adoption as in terms of the rise of the number vendors in the virtualization space.

For industry standard x86 systems, virtualization approaches use either a hosted or a hypervisor(also called Virtual Machine Monitor - VMM) architecture. A hosted architecture installs and runs the virtualization layer as an application on top of an operating system and supports the broadest range of hardware configurations. In contrast, a hypervisor (bare-metal) architecture installs the virtualization layer directly on a clean x86-based system.

Originally, on 1976, Goldberg (GOLDBERG, 1973) stated Virtual Computer Systems(VCS), as an important construct which arises a solution to a particularly vexing problem: with multi-access, multi-programming, multi-processing systems has simplified and improved access to computer systems by the bulk of the user community, there has been an important class of users unable to profit from these advantages: system programmers, whose programs must be run on a bare metal machine and not on an extended machine, e.g., under the operating system.

On Goldberg work, it was highlighted the problem of addressing: system programs, e.g., other operating system or different versions of the same operating system, require direct addressability to the resources of the system and do not call upon the operating system to manage these resources. Thus, system programs may not co-exist with normal production uses of the system.

And concludes: Recently, a technique for resolving these difficulties has been advised. The solution utilizes a construct called virtual computer system or virtual machine which is, basically, a very efficient simulated copy (or copies) of the bare host machine. These copies differ from each other and from the host only in their exact configurations, e.g., the amount of memory of particular I/O devices attached. Therefore, not only standard user programs but system programs and complete operating systems that run on the real computer system will run on the VCS with identical results. Thus, the VCS provides a generalization over familiar systems by also being multi-environment system. Since a VCS is a hardware-software duplicate of a "real existing computer system" there is always the notion of a real computer system, RCS, whose execution is functionally equivalent to the VCS. The program executing on the host machine that creates the VCS environment is called **Virtual Machine Monitor, VMM**.

Two categories of VMMs are defined by Goldberg:

- type 1 - the VMM runs on a bare metal;
- type 2 - the VMM runs on a extended host, under the host operating system.

Actually, there are four main accepted approaches that can be applied to implement server virtualization in x86 architecture:

i. Full Virtualization: That is the particular kind of virtualization that allows an unmodified operating system, with all of its installed software, to run in a special environment, on top of existing operating system. This environment, called a virtual machine, is created by the virtualization software by intercepting access to certain hardware components and certain features. The physical computer is usually called the host, while the virtual machine is often called a guest. Most of the guest code runs unmodified, directly on the host computer, and it runs transparently on real machine: the guest is unaware that it is being virtualized. Virtual Box (ORACLE, 2014), VMWare Virtualization (VMWARE, 2014) softwares and (QEMU, 2014) are examples of full virtualization products approach. KVM (KVM, 2014) kernel-level virtualization is a specialized version of full virtualization, where the linux kernel serves as the hypervisor. It is implemented as a loadable kernel module that converts linux kernel into a bare-metal hypervisor. It was designed after the advent of hardware assisted virtualization, it did not have to implement features that were provided by hardware. So, it requires Intel VT-X or AMD-V (see Hardware Virtualization below) enabled CPUs.

ii. Paravirtualization: the approach with paravirtualization is to modify the guest operating system running in the virtual machine and replace all the privileged instructions with direct calls into the hypervisor. In this model, the modified guest operating system is aware that is running on a hypervisor and can cooperate with it for improved scheduling and I/O: it includes code to make guest-to-hypervisor transitions more efficient. Paravirtualization does not require virtualization extensions from the host CPU. Xen hypervisor (XEN, 2014) was the precursor of paravirtualization products.

iii. Container-based Operating System virtualization, also known as operating system level virtualization, is a lightweight alternative. This kind of virtualization partitions the physical machines resources, creating multiple isolated user-space instances. While hypervisor-based virtualization provides abstraction for full guest OS's (one per virtual machine), container-based virtualization works at the operation system level, providing abstractions directly for the guest processes. OpenVZ (OPENVZ, 2014), LXC(LXC, 2014), and Linux V-Servers (POTZL et al., 2014) are examples of container-based virtualization solutions.

iv. Hardware Virtualization: this approach is only available on systems that provide hardware support for virtualization. Virtual machines in a hardware virtualization environment can run unmodified operating systems because the hypervisor can use the native hardware support for virtualization to: handle privileged and protected operations and hardware access requests; to communicate with and manage the virtual machines (HAGEN, 2008). Both AMD and Intel implement this approach, that are called, respectively, AMD-V and Intel VT-X.

Relevant improvements achieved by server virtualization includes:

- Raise hardware utilization: many data centers have machines running at only 10 or 15 percent of total capacity . Nevertheless, a lightly loaded machine still takes up room and draws electricity. The operating cost of actual underutilized machine can be nearly the same as if it were running in top capacity. Virtualization performs a better way to match computing capacity with load, enabling a single piece of hardware to seamlessly support multiple systems and making much more efficient use of corporate capital (GOLDEN, 2007).

- Better energy efficiency: The impact of green initiatives has meant that companies are increasingly looking for ways to reduce the amount of energy they consume. The cost of running computers, coupled with the fact that many of the machines filling up data centers are running at low utilization rates, means that ability of virtualization to reduce the total number of physical servers can significantly reduce the overall cost of energy for companies (GOLDEN, 2007).

- Contain physical expansion of data centers: virtualization, by offering the ability to host multiple guest systems on a single physical server, allows organizations to reclaim data center territory, thereby avoiding the expense of building out more data center space. This is an enormous benefit of virtualization, because data centers can cost in the tens of millions of dollars to construct (GOLDEN, 2007).

- Reduce overall system administration cost: virtualization can reduce system administration requirements drastically, making it an excellent option to address the increasing cost of operations personnel (GOLDEN, 2007).

- Reduce difficult to deploy applications: since virtualization adds a layer of abstraction that eliminates the need to configure the necessary software environment to all physical machines on which all applications will be run, a single VM image is created and deployed on any machine with a compatible VMM (DELGADO et al., 2011).

2.4.1 Applied Virtualization Solutions

2.4.1.1 KVM

Kernel-based Virtual Machine is a full virtualization solution for Linux on x86 hardware containing virtualization extensions (Intel VT or AMD-V). It consists of a loadable kernel module, kvm.ko, that provides the core virtualization infrastructure and a processor specific module, kvm-intel.ko or kvm-amd.ko. KVM also requires a modified QEMU although work is underway to get the required changes upstream(KVM, 2014).

KVM intermediates code execution of guest operating systems. It relies on Intel-VT and AMD-V hardware virtualization technologies to achieve a better performance. As a full virtualization solution, there are no requirements about kernel similarity between guest OS and Host OS. KVM creates virtual machines as Linux processes which can then run either Linux or Windows as a guest operating system.

2.4.1.2 OpenVZ

OpenVZ is an container-based virtualization solution for Linux. OpenVZ allows a physical server to run multiple isolated domains, through operating system instances, called containers(CTs), Virtual Private Servers(VPS), or Virtual Environments(VE) over a single host operating system modified kernel. Each CT performs and executes exactly like a stand-alone server for its users and applications, as it can be rebooted independently and has its own root access, users, IP addresses, memory, processes, files, applications, system libraries, and configuration files (KOLYSHKIN, 2014a). OpenVZ claims to be the virtualization tool that introduces less overhead, because each container shares the same host operating system kernel, providing a high-level virtualization abstraction. Despite the small overhead introduced by OpenVZ, it is less flexible than other virtualization software solutions, such as KVM, VMWare and Xen, because OpenVZ execution environments must be a Linux distribution, based on same operating system kernel of physical host server.

The OpenVZ kernel is a modified Linux kernel which adds the following functionality (KOLYSHKIN, 2006):

- virtualization and isolation: various containers within a single kernel;
- resource management, that limits subsystem resources(and in some cases guarantees), such as CPU, RAM and disk access, on a per-container basis. It is composed by three components:
    1. Two-level disk quota: OpenVZ server administrator can set up per-container disk quotas in terms of disk space and number of inodes. This is the first level of disk quota. The second level of disk quota lets the container administrator (container root) use standard UNIX quota tools to set up

per-user and per-group disk quotas. OpenVZ virtual disk is a partition of the host file system.

2. "Fair" CPU scheduler: The OpenVZ CPU scheduler is also two-level. On the first level it decides which container to give the time slice to, taking into account the container's CPU priority and limit settings. On the second level, the standard Linux scheduler decides which process in the container to give the time slice to, using standard process priorities.
3. User Beancounters: This is a set of per-container counters, limits, and guarantees. There is a set of about 20 parameters which are carefully chosen to cover all the aspects of container operation, so no single VE can abuse any resource which is limited for the whole computer and thus cause harm to other containers. The resources accounted and controlled are mainly memory and various in-kernel objects such as IPC shared memory segments, and network buffers.

- checkpointing: it saves container's state to a disk file, making container migration possible.

Figure 2.2 represents software architectural structure of the 3 physical server. Regardless of number of servers, each one have similar architecture. OpenVZ can be installed on an already installed Linux system, or it can be provided by ready to go distributions, such as Proxmox. Its customized configuration shall include the creation of a /vz partition, which is the basic partition for hosting containers and which must be way larger than the root partition. OpenVZ Layer, highlighted on Figure 2.2, is responsible for providing aforementioned functionalities. Nevertheless, to enable containers creation, it is necessary to install a OpenVZ OS template. Templates are a set of package files to be installed into a container. Operating system templates are used to create new containers with a pre-installed operating system. Therefore, it is necessary to download at least one OS template and save it on the Hardware Node. After install at least one OS template, creation of containers is enabled. It is possible to create any number of containers with the help of standard OpenVZ utilities, configure their network and/or other settings, and work with these containers as with fully functional Linux servers. On Figure 2.2, resulting containers are depicted on top of each physical server, labelled as VPS, where unmodified applications softwares can be executed.

OpenVZ allows containers to directly access memory, in a flexible way: during its execution, the memory amount dedicated to one container can be dynamically changed by host administrator. OpenVZ kernel manages containers memory space in order to keep in physical memory the block of the virtual memory corresponding to the container that is currently running.

Each container has their own network stack. This includes network device(s), routing table, firewall rules, network caches, and hash tables. From the perspective of container owner, it looks like a standalone Linux box. OpenVZ offers three major networking modes of operation,

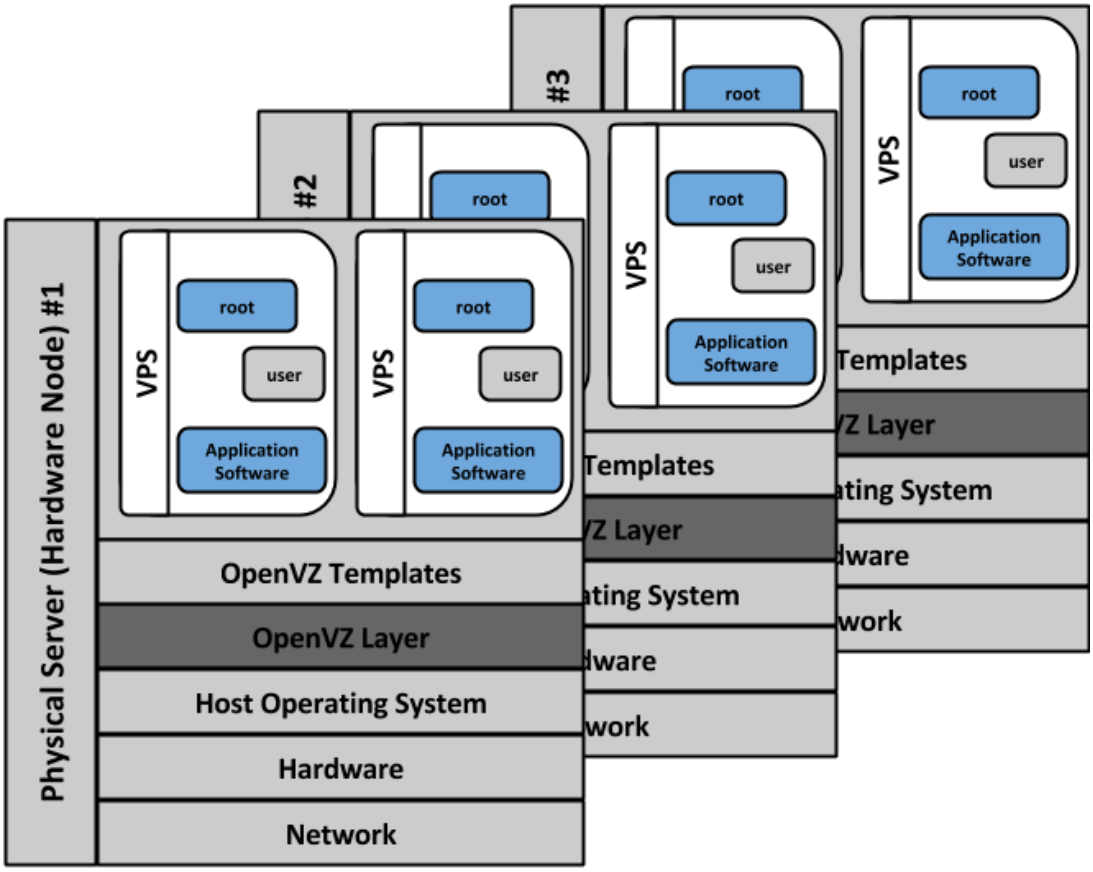


Figure 2.2: OpenVZ Architecture

as depicted on Figure 2.3:

- Route-based (`venet`)
- Bridge-based (`veth`)
- Real network device (`eth`) in a container

The main differences between them is the layer of operation. While Route-based works in Layer 3 (network layer), Bridge-based works in Layer 2 (data link layer) and Real Network in Layer 1 (physical layer). In the Real Network mode, the host system administrator can assign a real network device (such as eth0) into a container, providing the better network performance.

For the two modes of virtual network interfaces offered by OpenVZ to a container, virtual network device (`venet`) has lower overhead, but with limited functionality, serving simply as a point-to-point connection between a container and the host OS. It does not have a MAC address, has no ARP protocol support, no bridge support, and no possibility to assign an IP address inside the container. By contrast, a virtual Ethernet device (`veth`) has slightly higher (but still very low) overhead, but it behaves like an Ethernet device. A virtual Ethernet device consists of a pair of network devices in the Linux system, one inside the container and one in the host OS. Such two devices are connected via Ethernet tunnel: a packet goes into one device will come out from the other side (XAVIER et al., 2013).

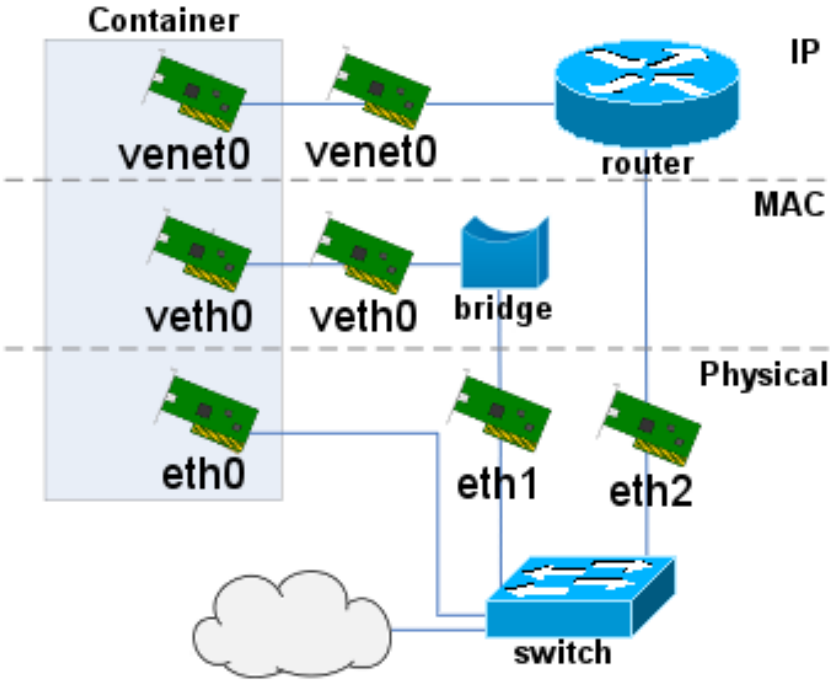


Figure 2.3: Containers networking: modes of operation

Host system administrator can assign a network device (such as `eth0`, as showed on Figure 2.3) into a container. Container administrator can then manage it as usual. Although it provides best performance, container assignment to a network device ties it to hardware.

3

TISVEP - Tuning Infrastructure for Server Virtualization Experiment Protocol

My favorite things in life don't cost any money. It's really clear that the most precious resource we all have is time.

—STEVE JOBS .

Activities associated with configuration of physical host machines, guest virtual machines and containers, analyzed services, and monitoring machines, to produce desired high availability, are tremendously time consuming. Automating these activities, and reducing configuration time intervals, becomes a core feature for this research. Without automating the necessary steps to execute measurement experiments, it would not be possible to perform the presented Chapter 4 experiments, due to time constraints.

The current chapter describes TISVEP - Tuning Infrastructure for Server Virtualization Experiment Protocol. Due to the large number of procedures for execution of an experiment, it was of paramount importance to automate such activities. TISVEP was designed to enable fast configuration and execution of experiments, performing automatic:

- configuration of all necessary levels of analyzed factors, such as those presented in Table 4.7, concerned with execution of experiments;
- initialization of the web cache servers, Web Polygraph tool, and monitor processes responsible for gathering system status;
- export of all results of the experiments for further analysis.

Regardless of the physical or virtual feature of the destination server, TISVEP is able to perform the aforementioned functionalities automatically. It decreases the time required for the previous manual experiment configuration sharply, from not less than 40 minutes, to not more than 3 minutes.

3.1 TISVEP Features

The improvement of the performability in virtualized servers, at first, required a large number of configuration tunings. as will be seen in Chapter 4, changes in the provided service are not rare, such as increasing storage capacity, and may result in additional improvements to maintain target availability of five 9's. Therefore, in such scenarios aimed at automatically providing resources to conduct experimental evaluations related to performability, these key design features were proposed:

- Lightweight: it should be assumed that computational resources are scarce. TISVEP Protocol is characterized by low overhead in the processes of message generation and transmission. It was developed in the Bash Script Language, applying netpipes TCP/IP streams sockets (FORSMAN, 1999; BERNIER, 2004) as the communication technology between the Cloud NMS node (refer to Figure 4.3) and destination targets. In addition, except for Internal Field Separators (IFSs), all transmitted data in the application layer is formed by payload, minimizing meta-data.

- Extensibility: due to the nature of the issue tackled, it is necessary to deal with new functionalities frequently. In the experimental environment of this research, based on the tuning of virtualized servers, it is common to perform adjustments, as well as to analyze new factors and their respective levels. Extensibility, the ability to have new functionality, taking future changes into consideration, was a key requirement identified due to the purpose of the protocol.

- Simplicity: the communication process among physical machines, virtual machines, and containers presented in the testbed (a representative of private clouds) that supports execution of experiments, can be accomplished simply, even in the context of such sophisticated infrastructure. The simpler the generation, transmission and processing of the messages remains, the more conducive the environment will be to accommodate the two aforementioned key features.

Previously in TISVEP development, netpipes was employed in a simple approach during the initialization of experiments: through a Bash script executed on a Web Polygraph server machine, a `polygraph-server` process was started and thereafter, the Web Polygraph client machine received a start call from a netpipes stream socket, beginning the experiment through the execution of a `polygraph-client` process. All other required procedures, like configurations and data export, were made manually.

The previous communication process performed between Web Polygraph machines was extended and applied through TISVEP, broadly employing netpipes to perform automatic configuration, execution, monitoring, and storage routines of the experiments.

The proposed approach of automatic experiment execution is similar to that of many previous approaches, as discussed in Chapter 1.

Small capacity infrastructures are interesting because they force us to reinvestigate problems that are already thought to be solved. Instead of proceeding with the installation of several third-party tools, it was decided to design and develop a single protocol that is suitable for the testbed infrastructure and is able to perform all required functionalities through the configuration of one unique centralized file. With a simple and lightweight underlying infrastructure, TISVEP is able to configure and monitor all required resources, as well as start all required services and export resulting data. Furthermore, the proposed methodology of TISVEP is so generic that it can be applied to configure, monitor and store results of any applications that can be executed on systems with support for netpipes. So, it was expected that it would maintain low overhead over the testbed.

3.1.1 Implementation Technology

The TISVEP communication technology is based on the netpipes software package. The netpipes package makes TCP/IP streams usable in shell scripts. It is a suite of utilities built on the idea of conventional pipes, allowing different processes to communicate and share data using TCP sockets across the network (BERNIER, 2004). The main realized advantage consists of simplifying client/server implementation, allowing people to skip the programming related to sockets and concentrate on writing services. From this suite, TISVEP is based on the following commands:

- **`faucet`**: it is the server end of a TCP/IP stream. It listens on a port of the local machine, waiting for connections. Every time it gets a connection, it forks a process to perform a service for the connecting client.
- **`hose`**: it is the client end of a TCP/IP stream. It actively connects to a remote port and executes a process to request a service.
- **`sockdown`**: selectively shuts down all or a portion of a socket connection; it is used to close the output half of the network pipe, releasing the processing flow to the destination target.

Aiming at detailing the netpipes software package, the first used TISVEP message, `hdsConfig`, will be used as an example and can be observed in Appendix B.5.

For the conducted experiments, there are four possible destination targets where web cache server are provided and TISVEP server end is executed(script `autExperiment.sh`). They are exhibited in Figure 3.1.

For physical machines (PM - represented by Nodes 01, 02, and 03) and multiple instance (I) destination targets, the server end of TISVEP runs directly on the host operating system.

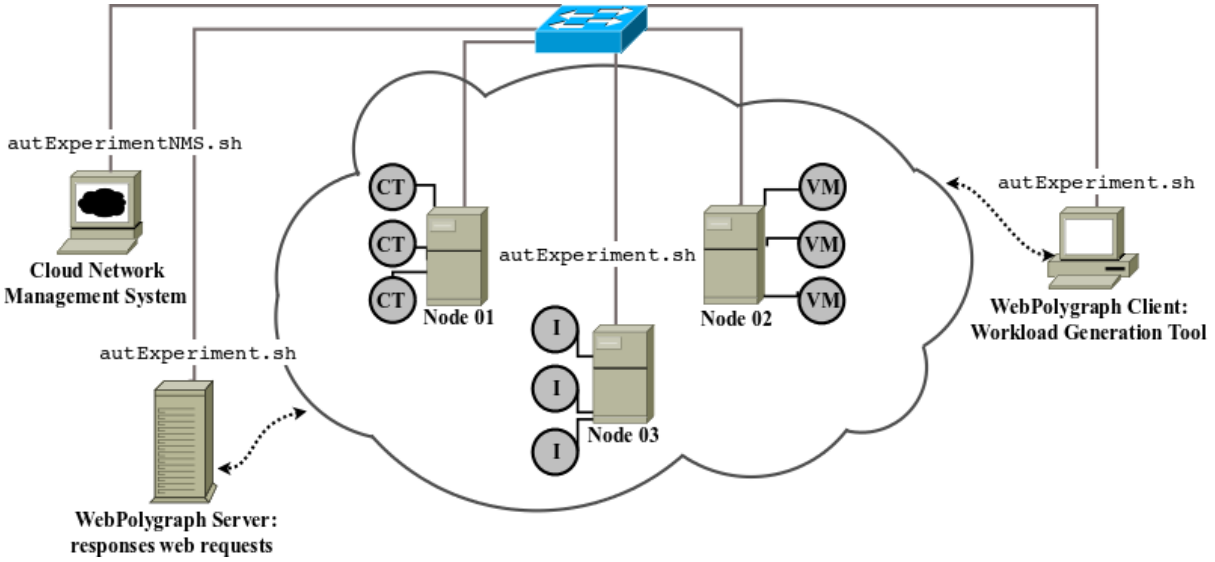


Figure 3.1: Supported web cache servers: physical machines(Nodes 01, 02, and 03), containers(CT), virtual machines(VM), and multiple-instances(I)

For virtual machines (VM), the server end of TISVEP runs on the KVM guest operating system. Finally, for containers (CT), the server end of TISVEP runs on an OpenVZ guest container.

3.1.2 Start up and halt flows

Functionally, to execute one replication of an experiment, TISVEP has a start up flow, depicted in Figure 3.2, and a halt flow, showed in Figure 3.3. The flows presented in such Figures depict transmitted messages for container-based virtualization experiments (the message flows to all other types of experiments are subsets of the container-based experiment flow).

A start up message flow is responsible to configure web cache storage, configure and initialize web cache servers, start monitoring functionalities, and configure and execute Web Polygraph processes.

A halt flow is responsible to stop the Web Polygraph server-side, stop monitoring tools, export generated data that will be analyzed, and clean up all necessary storage for posterior experiment executions.

Both start up and halt flows are controlled by `autExperimentNMS.sh` shell script executed on cloud NMS (Figure 3.1). In Figures 3.2 and 3.3, the central vertical time line labeled by NMS represents the execution flow of the `autExperimentNMS.sh` script. All necessary messages are forwarded to their destination components: physical machines (PMs), containers (CTs), for NMS machine itself, and to Web Polygraph machines. An introduction to TISVEP messages is provided in Section 3.3 and complete descriptions of the above shown TISVEP messages are provided in Appendix B.

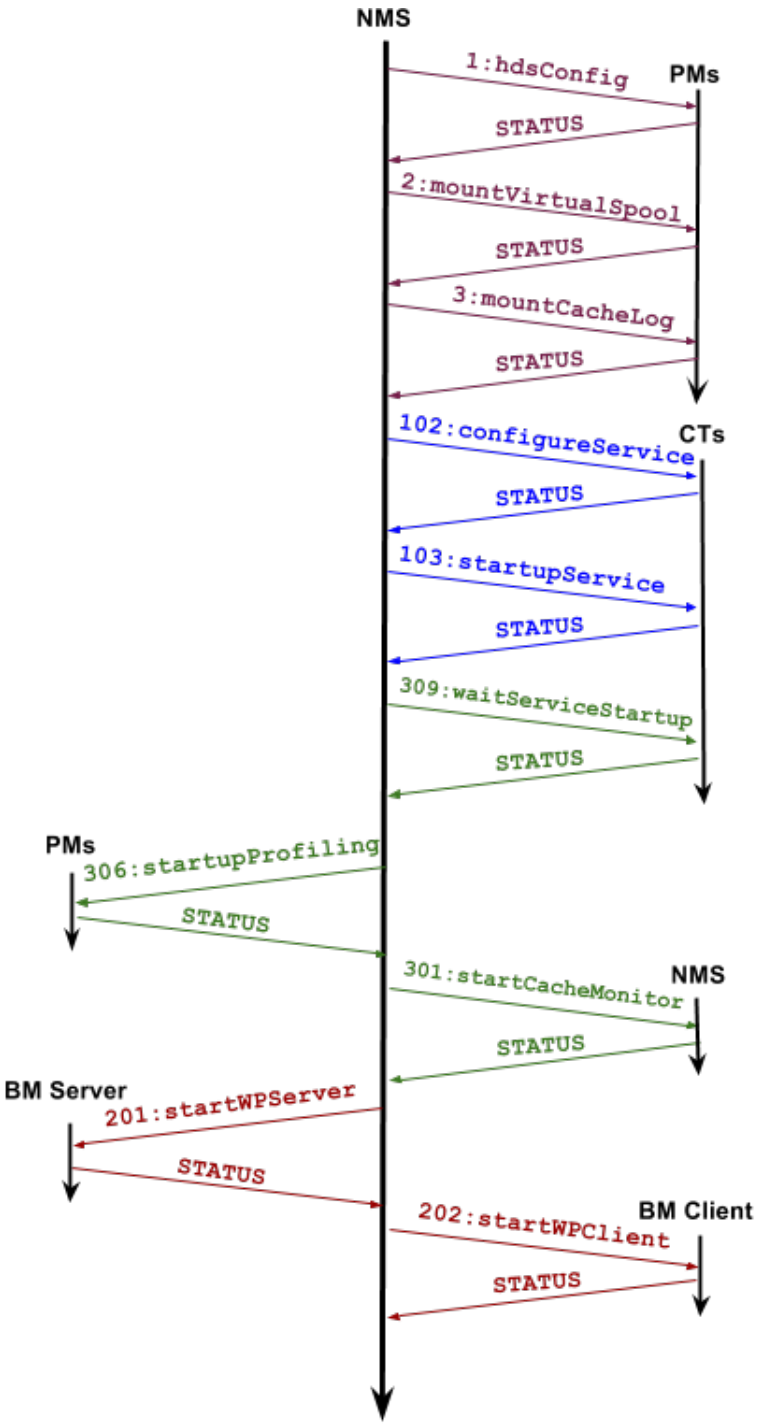


Figure 3.2: TISVEP startup flow for CTS experiment

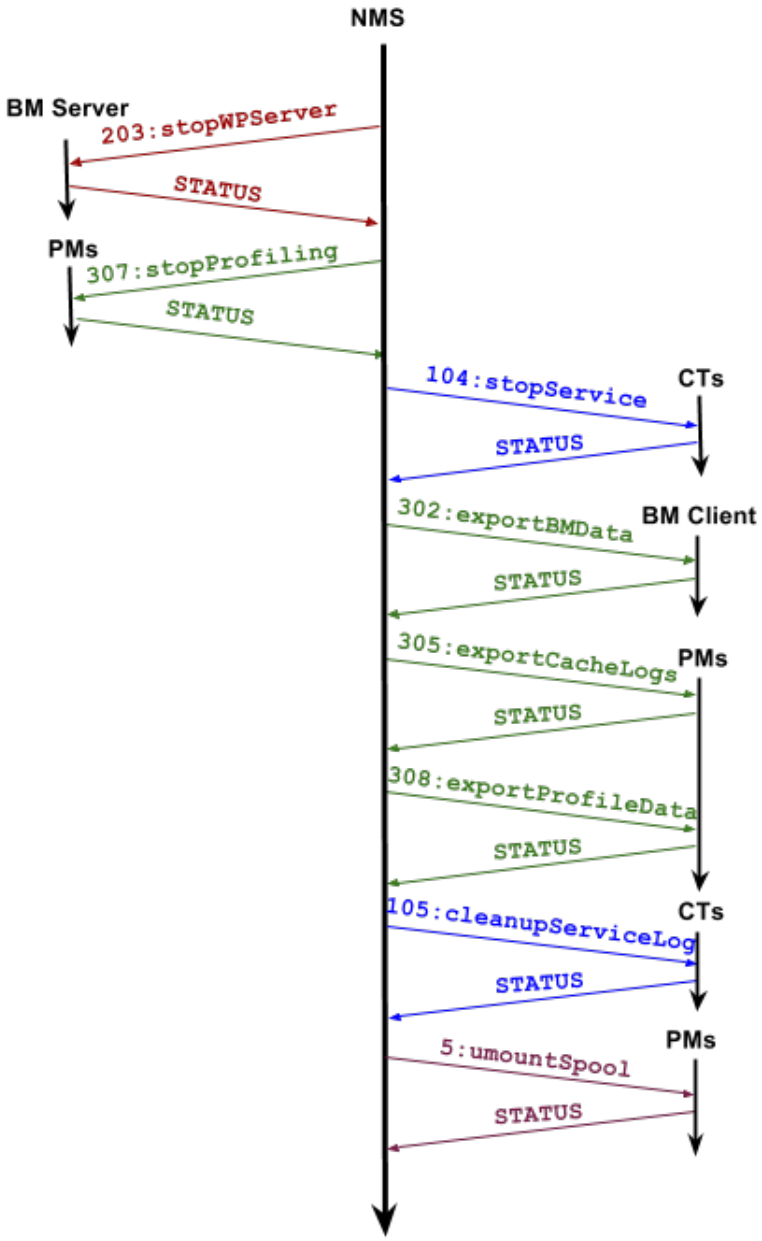


Figure 3.3: TISVEP halt flow for CTS experiment

3.2 TISVEP Static Entities

An experiment is designed in a simple and unique centralized text file, `config.properties`, that contains all the configuration parameters used to execute an experiment. The syntax applied in the `config.properties` file matches the Bash scripting language, aiming at exporting its content in the `autExperimentNMS.sh` script that is responsible for the execution of experiments. It is a simple method to import experiment configurations. So, to explain the contents of the `config.properties` file is of paramount relevance for understanding how an experiment is executed.

3.2.1 TISVEP config.properties file

The parameters that define the configurations of experiments are declared in the `config.properties` file as Bash variables of the following types: integer, string, and array (as well as associate arrays, that employ strings as indexes of the arrays). Such an approach in the definition of parameters was favorable to simplify the generation process of TISVEP messages, since it is only necessary to reference the variable identifier to use the parameter during the message assembling process, meeting the proposed key design feature of simplicity.

Based on their applicability, the configuration parameters were classified and divided into 6 different groups. Each of them is described and explained below.

1. Physical Machines Section: the parameters of this section store the required features of physical machines as well as their components that can be modified between the executions of different experiments. For example: hard disk formatting is required between experiments. Parameters such as disk label and partition size are imperative to perform this routine. All the parameters that make up Physical Machines Section are listed and explained below:

    - PMs: an array of IP addresses of the testbed physical machines. This parameter is used as source to get IP addresses of destination PMs when TISVEP messages must be sent to them.
    - HD_LABELS: string that contains the file names of hard drives as defined by the operating system.
    - PARTITIONS_NUMBER: it is the size of the partition that will be created during the first iteration of a set of experiments. For the current version of TISVEP, all partitions would have the same size.
    - PARTITION_SIZE: it is the size of the partition that will be created during first iteration of a set of experiments. For current version of TISVEP, all partitions would have the same size.

- FS_CONSISTENCY: logical value; defines if the file system consistency mechanism will be enabled or disabled. Possible values are:
  - 0: disabled;
  - 1: enabled.
- FS_TYPE: file system type to be applied during its building. Two file systems were used during the execution of experiments: ext3 and ext4.
- NO_CONSISTENCY_MOUNT_OPTIONS: this parameter contains the mounting options for the file system when the FS_CONSISTENCY parameter was equal to 0 (zero).
- SPOOL_LABEL: web cache server spool label, applied during the routine of partition mounting.

2. Service Section: The configuration parameters of the web cache server cluster are declared in this section. Regardless of whether the experiment runs in a non-virtualized or virtualized cluster, its configuration parameters are defined here.

   - SQUID_CONF_FILE: full pathname for the web cache server configuration file, used to perform required adjustments on provided web cache servers.
   - SQUID_BIN: full pathname for web cache server binary file. It is determined in the compilation of the Squid web cache server and is sent to destination servers aiming at executing such service.
   - CACHE_LOG_DIR: ull pathname for log web cache directory. It is mapped to a partition that is not on the same hard disk of spools used by the web cache server. It aims at reducing the concurrency of hard disk access over hard disks used as web cache server spools.
   - ENABLE_CACHE_LOG: as described in Section 4.2.3, disabling the cache log functionality was also favorable to reducing the concurrency of hard disk access. This parameter is applied with such a goal. It must be set as 0 (zero) to disable a cache log and as 1 (one) to enable it.
   - DEFAULT_PORT: states the default web cache server listen port. In experiments that applied multi-instances of the web cache server in a unique physical machine, several port numbers were required. Otherwise, the default port 3128 was applied.
   - CACHE_DIR_SUFFIX: it is used to form the full pathname of the web cache server spool directory in experiments that apply several hard disks and directories. As the spool's full pathname is formed programmatically,

through a loop during the message generation process, this suffix is used to complete the full pathname. An example was `/spool3/squid3`, where `squid3` is the suffix.

- CACHE_SIZE: volume of disk space (MB) to be used as spool by a web cache server. Increasing this value was one of the main challenges of this research.
- L1_CACHE_LEVEL: number of first-level subdirectories which will be created under CACHE_DIR, as explained in 4.3.3.
- L2_CACHE_LEVEL: number of second-level subdirectories which will be created under each first-level subdirectory, also explained in Section4.3.3.
- MI_AMOUNT: number of multiple instances of a web cache server, per physical machine. Used for execution of multi-instance experiments.

3. Benchmarking Tool Section: all the parameters that are required to establish communication with Web Polygraph machines, as well as to configure their functional properties that may be changed between experiments are defined in this section.

   - WP_SERVER_IP: IP address of Web Polygraph server physical machine; it is used to establish communication with the Web Polygraph server-side machine and start the `polygraph-server` process.
   - WP_CLIENT_IP: IP address of Web Polygraph client physical machine. It is of similar use to the WP_SERVER_IP, however on the client side. It aims at executing the `polygraph-client` process.
   - EXPERIMENT_DURATION: amount of time for each experiment replication; it is used to configure this value in the Web-polygraph configuration file.
   - WP_CONFIG_FILE: full pathname for Web Polygraph configuration file. It is used during configuration of Web Polygraph's parameters.

4. Virtual Section: the parameters presented in this section are formed by associate arrays that map physical machines to the virtual servers as well as to their required resources.

   - PMsServerIPs: associative array with physical machine IP addresses as indexes and strings that contain virtual IP server addresses, associated with each physical machine, as values.
   - PMsCT_IDs: associative array with physical machine IP addresses as indexes and strings that contain a set of pairs of spool directory and

numeric identifiers of containers as values. It is used in experiments in which a web cache server has been provided in a container-based virtualization infrastructure, through OpenVZ.

- VMsSPOOLs: associative array with virtual machine IP addresses as indexes and strings that contains a set of pairs of spool directory and numeric identifiers of virtual machines as values. It is used in experiments in which a web cache server has been provided in a full virtualization infrastructure, through KVM.

5. Management Section: it contains parameters that configure the experiment itself, such as type of experiment, IP address of Cloud NMS, and more. They are all described below.

    - SIMULATION_TYPE: core parameter; it defines which type of experiment will be performed. TISVEP supports 4 types of experiments:
        - physical machines (PMS): web cache servers are provided on physical machines, with a relationship of one to one (1:1);
        - containers (CTS): web cache servers are provided through containers, applying container-based virtualization. It is possible to form a one to many (1:n) relationship between physical machines and containers;
        - virtual machines (VMS): web cache servers are provided through virtual machines, applying full virtualization. It is possible to form a one to many (1:n) relationship between physical machines and virtual machines;
        - multiple-instances (MIs): web cache servers are provided on physical machines, with a relationship of one to many (1:n).
    - MOUNTPOINT_NAMES: used to query for mounted partitions, as search keys.
    - NMS_IP: IP address of Network Management Server machine.
    - CACHE_MONITORING_STEP: time interval for external monitoring function that gathers performance metrics through web cache servers.
    - LOCAL_STORAGE: full pathname of the storage directory on NMS.
    - REMOTE_STORAGE:full pathname of the storage directory on web cache servers (physical of virtual).
    - ITERATIONS: number of replications of the configured experiment to be performed. Example: to test modes of operation of network in OpenVZ,

five replications were performed per mode of operation. The number 5 was the value of the ITERATIONS parameter.

6. Profiling Section: it contains `oprofile` related parameters, used to perform profiling of hardware events.

    - PROFILING_EXP: associative array with CPU's performance counter event types as indexes, and their respective counters as values.
    - OPERF: full pathname for the binary file of the `operf` profile tool, used on physical machines to start the profiler.
    - OCOUNT: full pathname for binary file of the `ocount` profile tool. It is used specifically to count hardware events of processes.
    - PROFILE_COD: numerical code for oprofile instrumentation tools. Possible codes are:
        - 1: `operf` tool;
        - 2: `ocount` tool.
    - OCOUNT_INTERVAL_LENGTH: time interval for printing collected results on output when `ocount` tool is used.

The Cloud Network Management System (NMS) is the entity responsible for yielding the TISVEP messages, using static data from the `config.properties` file, and for transmitting them to testbed components, as described in the following section that covers the dynamic functionalities of TISVEP. A complete example of an instance of the `config.properties` file is presented in Appendix A.2.

3.3 TISVEP Messages

For each replication of an experiment, there is a set of components that must be configured. These components were classified according to their roles in the experiments. According to these intrinsic component roles, messages transmitted by TISVEP were categorized, yielding the following Message Code Definitions:

- 1-100: Infrastructure Configuration Messages: they are intended to transmit hardware and system host parameters to configure web cache storage.
- 101-200: Service Configuration Messages: applied to configure web cache service settings and manage their life cycles.
- 201-300: Benchmarking Configuration Messages: used to configure Web Polygraph server and client machines, as well as their life cycles.

- 301-400. Experiment Monitoring and Management Messages: class of messages responsible for managing the system's monitors and injection of TISVEP updates (upload of newer versions).

TISVEP messages, that comprise the message code class, are reported in the Appendix B.

4

Performability Analysis of Virtual Web Cache Services

The current chapter presents contributions on how to achieve high performability for web cache server clusters provided in virtualized environments. First of all, the main issue tackled was reduce overhead in virtualized environments, aiming at eliminating the unresponsiveness phenomenon, reaching a similar value of five 9's of availability of the non-virtualized environments. The availability was chosen over reliability for performability analysis because the main interest is related time losses. With availability of five 9's, a fair comparison with non-virtualized and virtualized clusters can be performed.

Thereafter, a tuning performance phase was conducted, during which measurement experiments were executed until some analyzed performance metrics presented better results in virtualized environments when compared with non-virtualized ones. The application of screening fractional factorial experiment design was remarkable on reaching this goal.

Furthermore, this chapter also presents case studies and their findings regarding comparisons between virtualized and non-virtualized web cache server clusters. Initial experiments, that used a limited amount of storage space, were performed without TISVEP automation. Subsequently, for experiments applying larger amounts of storage space, until it reaches the total capacity, TISVEP was used to load scenario parameters and control the replication of experiments.

For TISVEP managed experiments, each scenario is presented on their case studies with particular features that differ on:

- The TISVEP `config.properties` file, as exemplified on Appendix A.2;
- testbed tunings.

4.1 Performability Analysis Methodology

Dealing with server virtualization environments requires the execution of a wide range of service configuration, administration, and even implementation actions. Development of

current performability analysis methodology aims at guiding the feasibility and execution of experiments. The proposed methodology is general enough to be used in both non-virtualized and in virtualized environments.

The proposed methodology is depicted in Figure 4.1. During **workload characterization of analyzed web service**, a state of the art study about web cache server behavior was conducted, intending to identify which probability distributions best fit to web objects' popularity pattern and size. Since web cache server studies have been widespread in the scientific community, analysis of related studies allow the identification of which probability distributions should be used to model workloads during performance evaluation experiments. As a result of the conducted study, Zipf distribution was identified for modeling web objects' popularity and Pareto distribution for modeling the size of web objects.

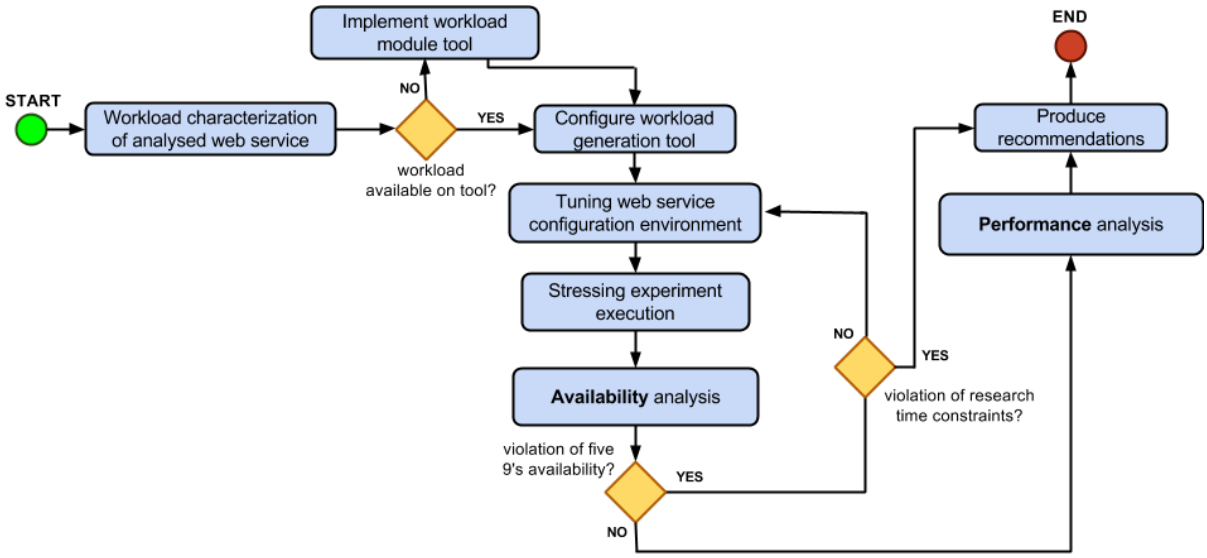


Figure 4.1: Methodology for Performability Analysis

As can be noted in Figure 4.1, a conditional step may be required: if the selected workload generation tool did not support the required workloads, it must be extended. There are several available workload generation tools intended to excite web servers. For web cache servers, three of them were identified: curl-loader (IAKOBASHVILI; MOSER, 2012), ApacheBench (FUNDATION, 2013), and Web Polygraph (POLYGRAPH, 2012). Due to the fine-tuning feature of Web Polygraph, it was selected as the web traffic workload generation tool used during conducted experiments. From two identified workloads for web cache servers, only Zipf is supported. It was necessary to implement a workload module tool. The workload based on Pareto Distribution was not implemented. So, Web Polygraph was extended with Pareto distribution to make it compatible with such a requirement of the current research.

With the required workloads available on Web Polygraph, they must be configured to excite the web cache server cluster whose performability will be analyzed. The **Configure workload generation tool** step of the proposed methodology illustrates this activity. Parameters that represent web objects' popularity frequency, web objects' types (such html and image files),

object sizes, level of file recurrence (how often a cached object will be revisited), variability of object modification (how often a cached object will be modified), and probability distribution models can be tuned on using the workload generation tool.

The next activity, illustrated as **tuning web service configuration environment** in Figure 4.1, aims at configuring web cache server cluster. As described in Section 2.2, web cache servers have several configuration parameters. As architectures, replacement policies and inter-cache protocols can impact on performability, they were identified as factors of the experiments. All supported levels of these factors were also described in Section 2.2. All combinations of levels from the aforementioned configuration factors were evaluated during a screening fractional factorial experiment (DOUGLAS C. MONTGOMERY, 2011). Furthermore, hardware tunings, as supply RAM and disk requirements for different cluster configurations, as well as network adjustments, were also performed during this step. Such procedure enabled the improvement of web cache server cluster performability in virtualized environments.

Stressing experiments can be executed after the configuration of the workload generation tool and of the web cache server cluster. Such configurations aim at filling all storage capacity of a web cache server cluster, submitting it to the stress of operating with the interaction of replacement policy for configured workloads. This procedure of methodology is illustrated as **Stressing experiment execution** in Figure 4.1.

Availability analysis is a core procedure of the proposed methodology. During adopted time intervals for analysis of the availability, the user experience was applied to measure downtime factor of expression 2.1. As described in Section 2.1, the User-Perceived Service Availability was adopted. It is the availability that the customer actually experiences: that under the point of view of the user receiving the service. All the presented availability ratings throughout this dissertation were measured in the client-side. The availability analysis evaluates whether or not 99.999% of availability was reached. The means for providing availability improvements, such as overhead reduction, have been investigated and applied to web cache server clusters. To the best of our knowledge, results greater than or equal to this threshold have not been presented in performability research into server virtualization. Results under such threshold were considered as unsatisfactory, triggering new iterations of "Tuning web service configuration environment" and "Stressing experiment execution" procedures, since the research time constraints are not violated. If so, the performance analysis will not be conducted and the recommendations will be related to availability only. Such threshold was hard to be reached on virtualized clusters due to reasons presented throughout this chapter. Five 9's of availability on virtualized clusters typify an important contribution of this research.

Having reached five 9's of availability, **performance analysis** takes place. Several performance metric comparisons, between non-virtualized and virtualized scenarios, were achieved. The objectives of this procedure are twofold: (i) to analyze hit ratio, response time, throughput, and byte hit ratio performance metrics on virtualized and non-virtualized environments to state the conditions that result in favorable metrics for virtualized web cache

clusters; (ii) to select, among non-virtualized (e.g. single instance and multi-instance) and virtualized (e.g. full virtualization and container-based virtualization) web cache server cluster configuration, those that present better performance, aiming at reducing the number of evaluated scenarios based on such performance features. Performance comparisons of confronted non-virtualized and virtualized scenarios are made during this procedure.

After all previous procedures of proposed methodology, it will be possible to **produce recommendations** on best practices to reach a better performability on virtualized web cache server clusters, in comparison with non-virtualized environments.

4.2 Application of Proposed Methodology: High Availability

One of the major efforts performed in this research was to reach five 9's of availability for web cache servers, when they were provided in environments in which server virtualization was applied.

The main steps of the process to reach high availability are described throughout following sub-sessions. A sequence of the key experiments, depicting its virtualization technology, time of execution and resulting availability, is portrayed in Figure 4.2, and is used to perform the presented elucidations.

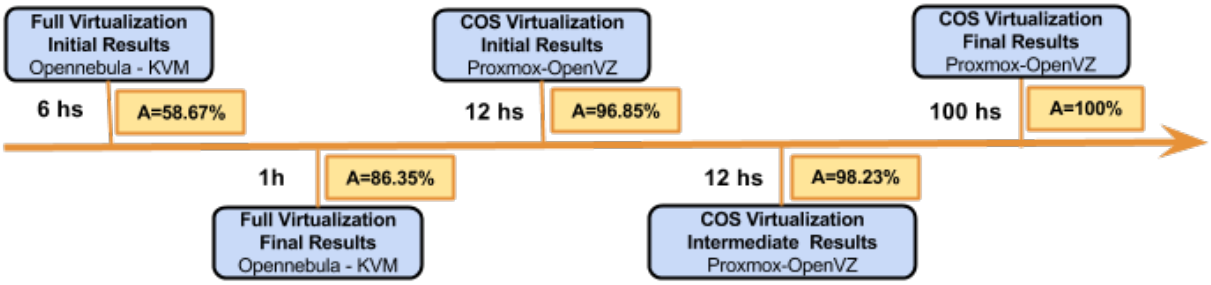


Figure 4.2: Summarized sequence of performed experiments

4.2.1 Non-virtualized initial experiments

Before conducting experiments on clusters of web cache servers over virtualized environments, a series of experiments on non-virtualized cluster scenarios was performed.

Initial experiments were performed on a testbed composed of 7 machines, as portrayed in Figure 4.3.

First configuration of the testbed for non-virtualized cluster scenarios was composed of 4 clusters' physical servers (Frontend and Nodes01-03), powered by AMD Phenom x86_64 Quad-Core 2.3 GHz processors, 4 GB RAM, with posterior upgrading for 8GB, discrete L1 and L2 cache structures for each core, with 128KB and 512KB each, respectively, in a total of 512KB for L1 and 2048KB for L2, shared L3 cache size of 2048KB, Gigabit Ethernet adapter, SATA hard disks (250GB of capacity, 7200RPM, transmission rate of 6Gb/s) and Linux Ubuntu Server

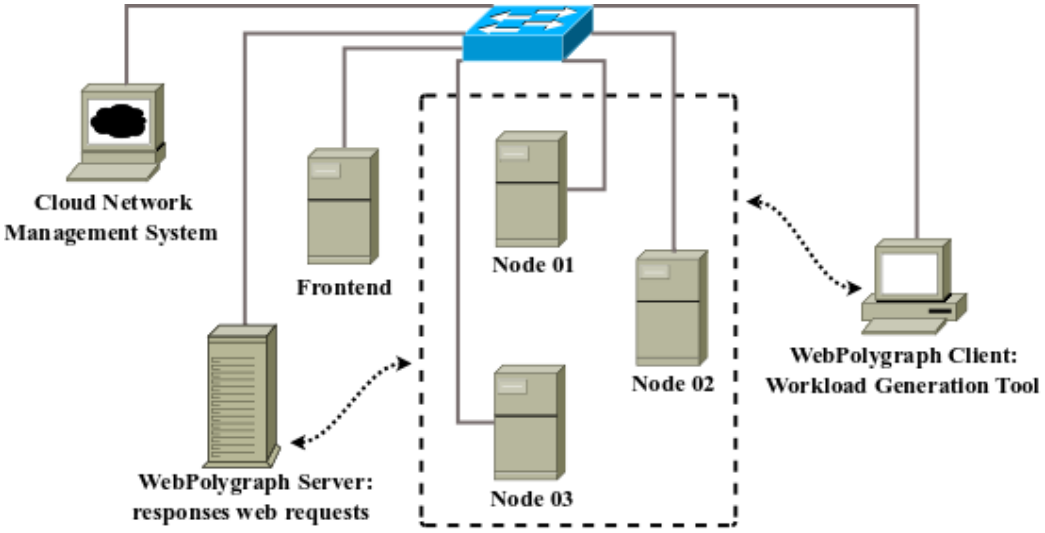


Figure 4.3: Testbed for initial non-virtualized warm up experiments

12.04 as operating system. Squid Web Cache 3.1.20 binary release was installed. Default ICP inter-cache protocol and LRU cache replacement policy were applied. Aiming at simplifying configuration on warm up stage, mesh architecture was selected. A Cloud Network Management System workstation was used to perform all necessary configurations and commands to execute the experiments, in addition to gathering resulting data and to performing data analysis from those experiments.

The Web Polygraph tool was installed on two other testbed workstations. The tool's client-side is responsible for generating workload that excites web cache servers running on the cluster's nodes. When the requested objects were not stored on the server's caches, these requests were forwarded to the Web Polygraph server-side, which represents the Internet, and contains all requested objects. Another feature of the client-side is that it is able to state the number of users that will make requests. Table 4.1 depicts a summary of configured parameters as well as the availability result of the first experiment performed with 100 web cache server clients.

Table 4.1: Warm up availability experiment parameters and result

Architecture	Inter-Cache Protocol	Replacement Policy	Cache size (per node)	Duration (h)	Availability
Mesh	ICP	LRU	1GB	6	100%

Aiming at fast observing the behavior of replacement policy during proposed time interval of 6 hours for this experiment, a limited cache size of 1GB per node was employed: once all the cluster's 3GB capacity had been filled, replacement policy starts to flush less used objects according to policy rules. Without free spool, the web cache server cluster reaches the steady state operation.

Availability is stated by the output log of the client-side Web Polygraph, that contains runtime statistics. If no responses were received during the configured output time interval, Web

Polygraph would log "-1.00" in the hit ratio field. The shorter the time interval was configured, the higher was going to be the probability of receiving no responses during such time interval. The default Web Polygraph time interval was 6 seconds. During execution of experiments, resolution was adjusted to 1 second. A successful response of a web cache client request is logged as shown below:

```
013.00| i-FirstRound 230310 151.57 258  53.29  0  100
```

There are 9 fields in each registered line:

1. Minutes since start (**013.00**);
2. "i-" and "p-" stand for interval and phase-based statistics(**-i**): conducted experiments are based on time interval statistics;
3. current phase name (**FirstRound**): identify phase name, a prerequisite of Web Polygraph. It can be configured aiming at favoring the identification of output based on previous configuration;
4. number of replies received so far (**230310**);
5. reply rate (**151.57**): received replies, per second;
6. mean response time in milliseconds (**258**): for each log event, hundreds of responses can be received;
7. hit ratio in percents (**53.29**);
8. number of transaction errors during that interval or phase(**0**);
9. number of open sockets(**100**). It includes UDP and other housekeeping sockets if any. This number is usually close to the number of pending transactions.

In cases where no responses are received, Web Polygraph log line is similar to:

```
013.02| i-FirstRound 230310   0.00    -1  -1.00  0  100
```

The two log lines presented above are consecutive. As there are no responses in time interval between the first and second presented log lines, the fourth field value, number of replies received, is not incremented. The reply rate is logged as 0.00, mean response time is logged as -1, and hit ratio is logged as -1.00. Only in failure occurrences, in which no responses are received, such values are logged on those fields. The availability is assessed as 100% when no lines with unresponsiveness events, such as that showed in the frame above, are reported. The resulting summed seconds of unresponsiveness, logged by any one of these fields with

failure values, can be used as the downtime factor of the availability expression. It was quite common to observe such log lines in virtualized experiments, due to the previously discussed unresponsiveness phenomenon [1].

Similar experiments to the one shown in the Table 4.1 were conducted. They aim at stating availability of web cache server clusters with various combinations of architecture, replacement policies, inter-cache protocol, number of users, and cache size. During such initial execution of the experiments, a small amount of cache size was provided by the cluster, intended to evaluate the behavior of the clusters in reduced time intervals, enabling to perform availability analysis in a wide range of scenarios. No more than 10GB of cache storage was applied per node of the cluster. For such described environments, conducted experiments based on non-virtualized web cache cluster invariability result on `A=100%`. These results are meaningful evidences of the high availability on non-virtualized environments. As they are above of established availability goal of 99.999%, non-virtualized clusters were enabled to performance evaluation, as was defined on proposed methodology depicted in Figure 4.1.

Since non-virtualized environments showed no unresponsiveness failures so common in virtualized environments, it is possible to proceed with performance analysis. With registered outputs of Web Polygraph log lines, it is possible to achieve performance analysis of the following metrics: hit ratio, response time, and throughput.

So, another round of experiments was performed, by varying the cluster's architectures. Such experiments aiming at stating whether or not some of the architectures will operate better according to selected metrics. Ten replications of the designed experiment were performed for each one of the distinct architectures: Mesh and Hierarchical.

The mean values of hit ratio for each replication were collected for the construction of the Confidence Intervals(CIs) shown in Table 4.2. Confidence intervals were obtained from R tool (FUNDATION, 2014; TEETOR, 2011). Several functions provided by R are able to produce confidence intervals, such as `t.test(x)`. For provided vector "x", this function outputs confidence interval for the mean of "x" values. All results of the experiments are suitably represented by a normal distribution, enabling the use of `t.test` function.

Lower Bounds(LB), Mean value, and Upper Bounds(UB) of the CIs are shown for each architecture in Table 4.2. As can be seen, the confidence intervals for the evaluated architectures do not overlap. Moreover, mesh architecture presented better performance for the hit ratio. The average for the Mesh architecture presented 15:31% most hit ratio compared to the hierarchical architecture.

[1] See section 2.1.2

Table 4.2: 95% Confidence Intervals for Hit Ratio(%) Means

95% Confidence Intervals for HRs(%)			
	LB	Mean	UB
Mesh	56.33541	56.66463	56.99385
Hierarchical	41.25849	41.35311	41.44773

With identical configuration for both architectures, this unfavorable performance difference related to hierarchical cluster is due to fact that parent nodes, those in the root position of cluster hierarchy (illustrated as Parent Cache in Figure 2.1a), can forward cache misses for their children (nodes in the leaf positions). However, sibling nodes are not allowed to forward cache misses. That is the fundamental difference between the two analyzed architectures. For the scenario illustrated in Figure 4.3, node labeled as Frontend was configured as parent, whereas Node 01, 02, and 03, were configured as children.

For response time, presented as milliseconds(**ms**) and throughput, presented as responses per second (**resp/s**), results were inconclusive, once that shown 95% confidence intervals of Tables 4.3 and 4.4 overlap. Such confidence intervals did not present meaningful evidences about which architecture performs better based on results of the experiment replications.

Table 4.3: 95% Confidence Intervals for Response Time (ms) Means

Confidence Intervals for Response Times(95%)			
	LB	Mean	UB
Mesh	295.0757	306.6300	318.1843
Hierarchical	291.6923	304.7958	317.8994

Table 4.4: 95% Confidence Intervals for Throughput (resp/s) Means

Confidence Intervals for Throughput(95%)			
	LB	Mean	UB
Mesh	313.0671	326.4681	339.8692
Hierarchical	308.9077	325.7344	342.5612

Based on presented evidence that hit ratio performs better with mesh architecture in a non-virtualized web cache cluster, the experiments of following sections 4.2.2 and 4.2.3 were performed applying Mesh architecture.

4.2.2 Full Virtualization High Availability Results

First configured virtualized web cache server cluster scenario was formed by full virtualization technology. KVM was used as hypervisor.

Aiming at managing virtual machines, OpenNebula was employed as the Virtual Infrastructure Manager. OpenNebula is able to orchestrate storage, network, virtualization, monitoring, and security technologies to deploy multi-tier services as virtual machines on distributed infrastructures. It offers a flexible solution to build and manage enterprise clouds and virtualized data centers. There are two server roles in the OpenNebula model: the Frontend server executes OpenNebula services, and the Nodes are used to execute virtual machines. Such OpenNebula server roles are shown in Figure 4.5.

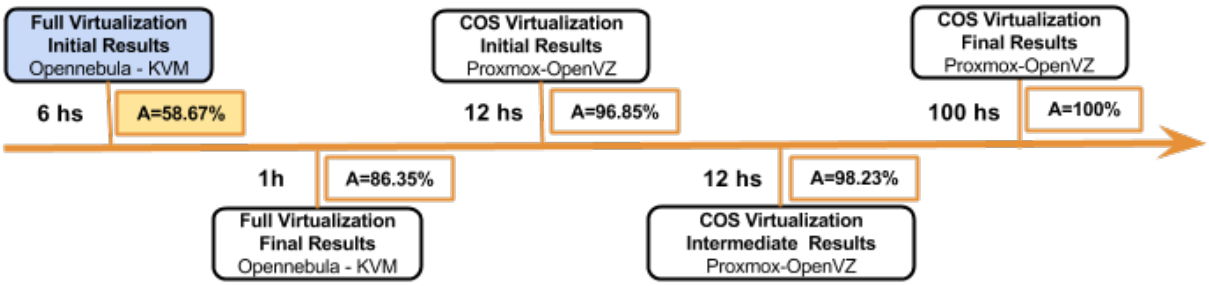


Figure 4.4: Full Virtualization Initial Analysis

Figure 4.4 highlights the initial reached availability (A=58.67%) for OpenNebula/KVM solution during 6 hours. The experiment was originally planned to be executed during 100 hours, nevertheless, with detected low availability, it was interrupted.

As there were 3 Server Nodes, to enable a future fair comparison between non-virtualized and virtualized environments performance metrics, and with server consolidation as a benefit towards virtualized servers, 3 VMs were created on one of these physical Node machines, as shown in Figure 4.5.

At least one experiment was executed for several full virtualization configurations of the web cache server cluster formed by 3 VMs, as exhibited in Figure 4.5, by varying:

- Squid web cache server, OpenNebula, and KVM versions;
- RAM amount associated to VM (with values from 32MB to 256MB);
- VCPU associated to VMs(from 1 to 4);
- storage infrastructure, with SSH and NFS protocols, used to distribute virtual machines images from FrontEnd to Nodes;
- web cache server architectures, inter-cache protocols and replacement policies.

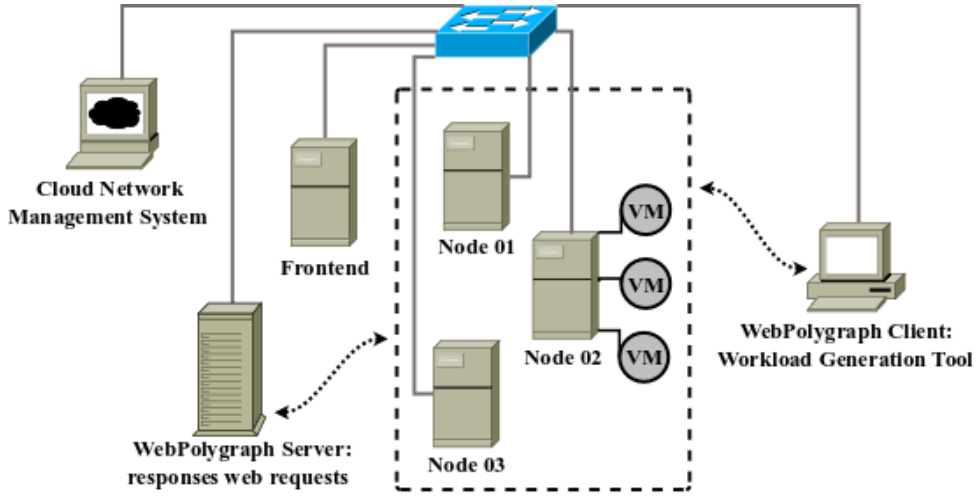


Figure 4.5: Virtualization Testbed Configuration: 3 VMs instantiated on 1 Physical Machine

Such variations of the cluster's configuration compound the procedure of **Tuning web cache configuration environment** of the proposed methodology (refer to Figure 4.1).

Even with those variations in full virtualization solution provided by OpenNebula/KVM, the highest availability reaches 86.35%, below of the five 9's goal. The time intervals used to execute the experiments are reduced, reaching only 1 hour, once the availability was increasingly degrading over time.

For fair comparison, all those variations that also could be applied on non-virtualized scenarios were tested, resulting in `A=100%` for similar evaluated time intervals of virtualized clusters. As depicted on Figure 4.6, initial applied time interval for experiments was of 6 hours. As goal availability of five 9's had not been reaching, the time interval of the experiments was reduced to 1 hour.

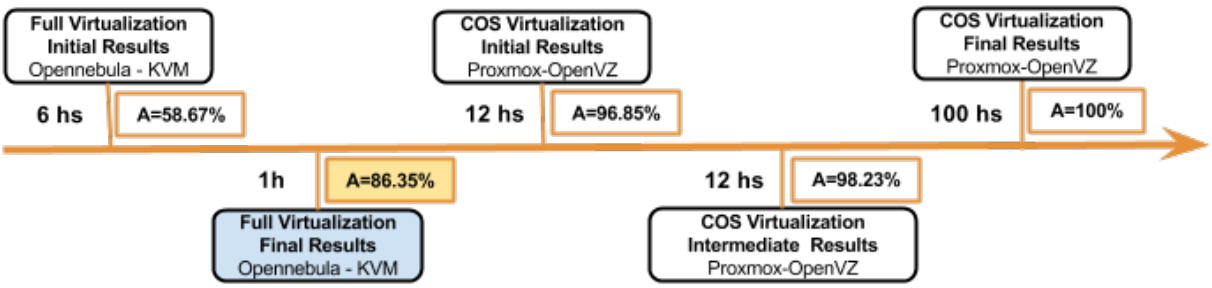


Figure 4.6: Full Virtualization Final Analysis

As shown in the next section, migration of the virtualization technology, from full virtualization to Container-based Operating System (COS) virtualization, causes a ready approximation to targeting the availability of five 9's.

4.2.3 Container-based Virtualization High Availability Results

Container-based Operating System virtualization enables multiple isolated execution environments within a single operating system kernel, partitioning physical machines resources and directly using OS system calls, while concurrent hypervisor solutions are required to provide a complete emulation of entire underlying hardware. A container is the proper isolated program execution environment, acting like a separate physical server.

To manage containers, Proxmox Virtual Environment (VE) was installed on testbed physical nodes. Proxmox VE is an open source software virtualization management solution for servers, optimized for performance and usability. It uses a Linux kernel and is based on the Debian GNU/Linux Distribution.

Physical nodes were assembled in a cluster configuration. In Proxmox parlance, cluster arrangement is called a data center. It can be used to quickly set up a virtualized datacenter, creating and managing:

- OpenVZ containers: preferred technology due to its low overhead, high performance and high scalability, compared with full virtualization solutions (XAVIER et al., 2013) and/or;
- KVM: Kernel-based Virtual Machine: provides robust flexibility, as it can host several guest operating systems (KVM, 2014).

A highlighted Proxmox VE feature is that it's easy to start; it is only necessary to download an ISO image and install Proxmox VE on the server hardware. The time required to create the first virtual machine or container was far less than with OpenNebula.

Initial COS Virtualization phase was conducted for **OpenVZ Containers**. The change from Full Virtualization technology to COS Virtualization technology brought us closer to the goal of achieving five 9's high availability. Applying an identical web cache server configuration file, just changing the virtualization technology from KVM to OpenVZ improves the availability from 86.35% to 96.85%, as depicted in Figure 4.7.

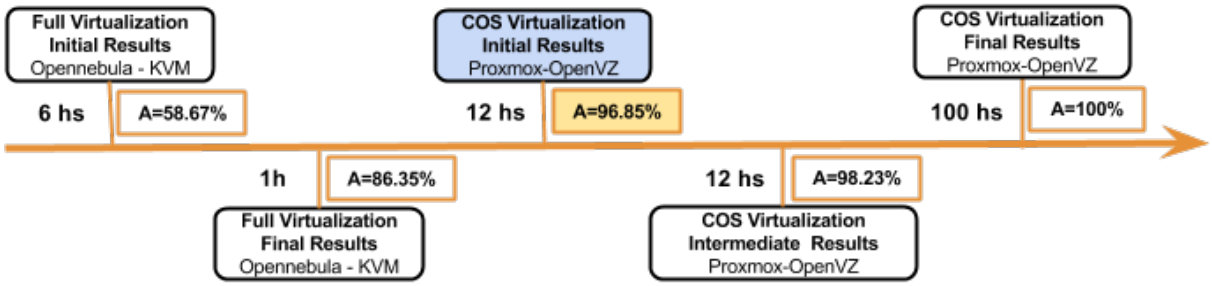


Figure 4.7: Container-based Operating System Virtualization Initial Analysis

As web cache systems are I/O bound (WESSELS, 2001), the web cache server spent most of the time performing disk access. The more concurrency on hard disks access exists, the

worse will be the capacity of the web cache server to answer requests from clients. To tackle this issue, the next tuning step aims at optimizing hard disk access, reducing overheads from any concurrent process.

The **intermediate COS virtualization** phase occurred with the reduction in hard disk access overheads. Once only the fundamental processes for kernel operation were enabled, in addition to the Squid web cache server, the primary solution to reduce the concurrency level for disk access was the **shutting down Squid web cache server logs**.

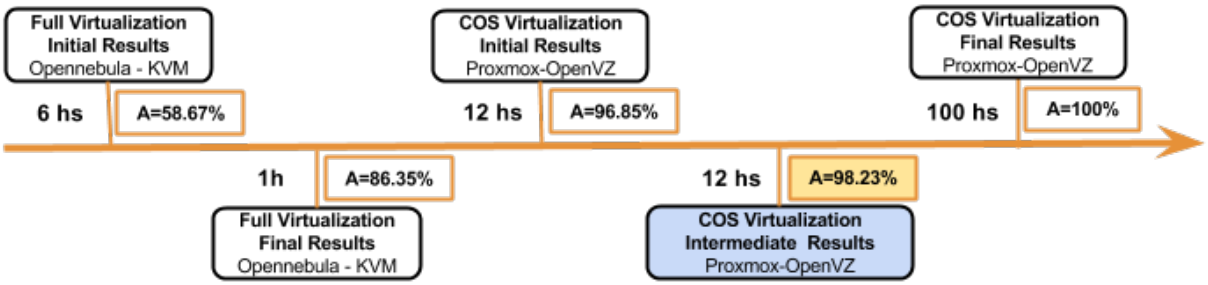


Figure 4.8: Container-based Operating System Virtualization Intermediate Analysis

Such tuning approached the desired five 9's of availability, going from 96.85% to 98.23%, as depicted in Figure 4.7. This improvement also enabled a higher time interval of the experiment: the degradation of the availability happened late.

The **final COS virtualization** phase occurred when an one-to-one CT to hard disk relationship was established. Without additional possibilities of decreasing hard disk access overhead, the final insight leading to the aimed availability of five 9's was:

- to associate one hard disk to each container that provides web cache service.

Previously, web cache I/O read and write flows were concurrent with kernel ones, where all 3 containers and the kernel perform operations with a unique host hard disk. With the new structure, depicted in Figure 4.9, running with a small cache capacity (web cache server spool was set to 1GB), the five 9's of availability were reached.

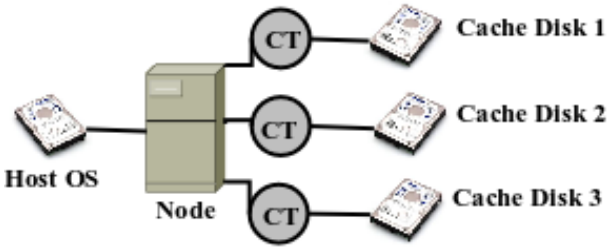


Figure 4.9: COS Virtualization Final Results: one-to-one CTs and Hard Disks relationship

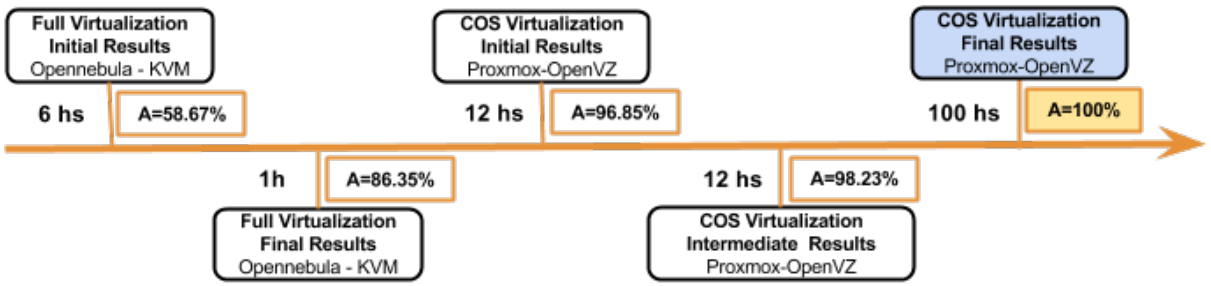


Figure 4.10: Container-based Operating System Virtualization Final Analysis

Each container shall have a unique hard disk, where each web cache server can perform read and write operations without interruptions from another web cache server.

To reach a high available service, with 99.999% of availability, only 3 seconds of unavailability are tolerated during an experiment of 100 hours. However, the combination of container-based virtualization, decrease of disk I/O operations, and containers performing exclusive disk I/O operations, led to 100 hours (as originally planned) of experiment without logged misresponses (absence of lines with -1.00 value in the log file of the Web Polygraph). The resulting availability is highlighted in Figure 4.10: A = 100%.

4.3 Tuning Performance Phase

With a viable infrastructure environment to provide high availability for an I/O bound virtualized web cache server cluster, it was the moment to proceed with the comparative performance analysis between the non-virtualized scenarios and virtualized ones, as well as to increase the size of disk space allocated for web cache storage. A small cache capacity of 1GB per cluster node is insufficient for a production web cache server, and must be upgraded. Furthermore, the time interval for each experiment was decreased from 100 hours to 80 hours, due to research time constraints. To reach five 9's of availability, only 2 seconds of unavailability are tolerated during an experiment of 80 hours.

4.3.1 Initial Metrics Analysis

The client-side of Web Polygraph logs three metrics of interest that were used for performance analysis: hit ratio, response time and throughput.

These metrics were analyzed and compared from two conducted experiments: the first with a baseline non-virtualized cluster, performed in 3 physical machines, and the latter with a container-based virtualized cluster, performed in a cluster with 3 containers in one physical machine, employing a relationship of 1:1 between containers and hard disks. The testbed was adjusted with similar configurations to those described in Section 4.2.1, with experiment time interval of 80 hours. Log time interval was configured to 1 second, and mean values of hit ratio were generated for each time interval of 1 hour.

Figure 4.11 shows mean points of hit ratios for time intervals of 1 hour. As can be observed, the hit ratio behaved better for baseline (non-virtualized) cluster. When cache storage was provided by a bind-mounted (virtualized) approach, the hit ratio presents a mean degradation of 11.64% compared to baseline. Due to the isolation properties of containers, it is not possible to mount external devices in a container's file system. Instead of mounting a block device into a particular path, the bind mount technique makes directories of the hardware node, which are attached to the host OS, visible to containers.

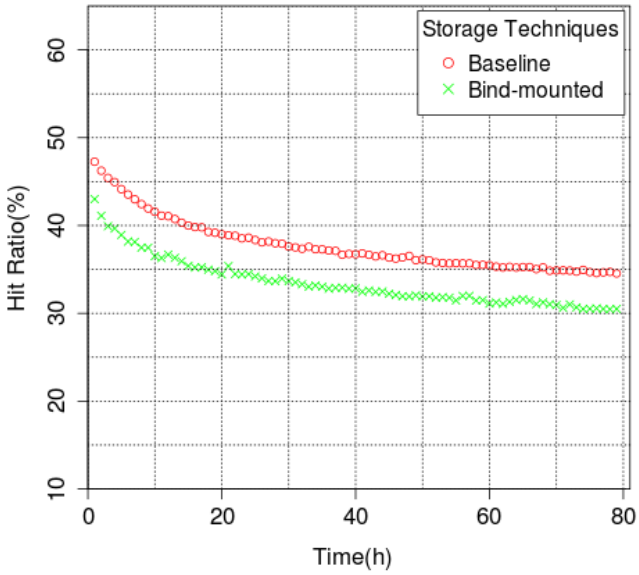


Figure 4.11: Performance metric: non-virtualized baseline storage presented better hit ratio than bind-mounted COS virtualization

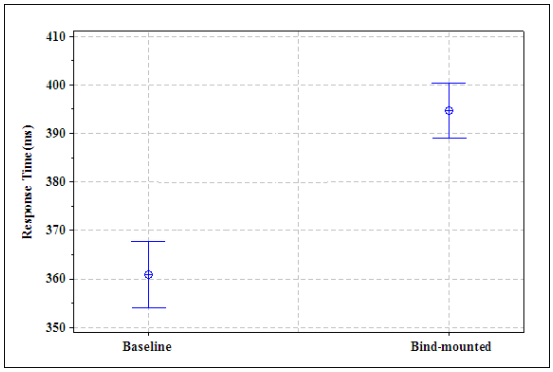
Considering the mean values presented in Table 4.5, virtualized bind-mounted storage response time was 9.43% greater than baseline. The degradation of the response time in the virtualized cluster, can be observed in the CIs depicted in Figure 4.12a.

Table 4.5: Comparison of performance metrics

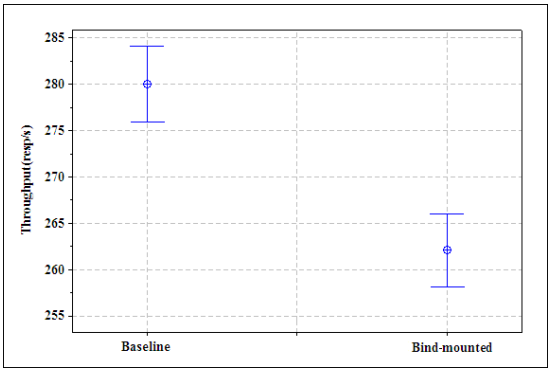
Throughput(responses/s)	**Min**	**Mean**	**Max**
Baseline	228.74	280.01	314.56
Containers (bind-mounted)	211.83	262.08	295.13
Response Time(ms)	**Min**	**Mean**	**Max**
Baseline	316.27	360.80	479.17
Containers (bind-mounted)	344.51	394.66	477.62

Virtualized bind-mounted storage degradation of throughput, compared to baseline, was

6.40%. The throughput degradation level can be observed in the CIs depicted in Figure 4.12b.



(a) Response Time comparative with confidence level of 95%.



(b) Throughput comparative with confidence level of 95%.

Figure 4.12: Performance metrics. (a) . Similar behavior was observed to (a) response time and (b) throughput.

Then, for the experiments configured as described in Table 4.6, the results depict that all metrics presents better performance for non-virtualized environment. Additional experiments should be conducted aiming at identifying possible causes of the deficit performance in the virtualized environments.

Table 4.6: Initial configuration of experiments during tuning performance

Architecture	Inter-Cache Protocol	Replacement Policy	Cache size (per node)	Duration (h)	Availability
Mesh	ICP	LRU	1GB	80	100%

From this point, it was necessary to investigate potentialities that could lead to improvements in the virtualized clusters. First of all, the web cache server features were investigated. As web cache server cluster have a wide range of parameters that can be exploited, performance analysis follows two phases:

1. A screening fractional factorial experiment (DOUGLAS C. MONTGOMERY, 2011) was designed and executed in a non-virtualized environment. It aimed to state which is the most appropriate combination, for performance optimization, maintaining high availability, of the web cache server factors used: architecture, replacement policy, and inter-cache protocol. The possible levels of each factor can be observed in Table 4.7. A complete discussion about them was performed on Section 2.2.

2. It is necessary to apply the better combination obtained in the first phase to state the impacts on availability and performance over an I/O bound web cache application provided in a virtualized cluster, when compared with an identical configuration over a non-virtualized cluster.

Table 4.7: Web Cache observed factors and levels

Factor	Levels
Web Cache Architecture	Mesh, Hierarchical
Web Cache Replacement Policy	LRU, heap LFUDA, heap GDSF, heap LRU
Intercache Protocols	ICP, HTCP, Cache Digest, CARP

4.3.2 Phase 1: screening fractional factorial experiment

The first phase of the performance experiments was executed, always observing the high availability requirement. During the screening fractional factorial phase, all supported Squid inter-cache protocols and replacement policy factors were combined and tested in a scenario without virtualization (baseline). Factor combinations result in twenty eight different scenarios (CARP protocol was not designed to work in mesh architecture), which are executed for 1 hour each (yielding 3600 samples for each combination) with resulting availability shown in Table 4.8.

Table 4.8: Availability for each evaluated scenario

	Hierachical				Mesh
	heap GDSF	heap LFUDA	heap LRU	LRU	All
ICP	97.6112%	99.6945%	98.9167%	98.8889%	
CARP	99.8612%	99.9445%	99.7778%	99.9723%	
Digest	97.3056%	97.1112%	97.9445%	98.1389%	100%
HTCP	98.1112%	97.9723%	98.6112%	98.3334%	

As can be seen, for hierarchical architecture, none of the web cache server factor combinations reached the five 9's availability requirement. For mesh architecture, all combinations were executed without unresponsiveness failures. Figure 4.13 presents the achieved confidence intervals of the hit ratio means only for mesh architecture scenarios: as stated by the applied methodology, performance evaluation for scenarios that violate the requirement of five 9's of availability must not be performed.

As shown in Figure 4.13, the combination of inter-cache protocol and replacement policies that maximized the hit ratio was ICP and heap GDSF. A similar result for heap GDSF replacement policy was achieved by Dilley et al. (DILLEY; ARLITT; PERRET, 1999). Thus, during the further analysis of the performance optimization, only mesh architecture was applied. With a mean hit ratio of 47.15%, this combination of factors was used to set up web cache clusters provided in a virtualized environment: the most favorably performability parameters, indicated in Phase 1 (without virtualization), were used as input for Phase 2 (with virtualization). The goal is to observe if the best combination of factors in non-virtualized cluster can improve

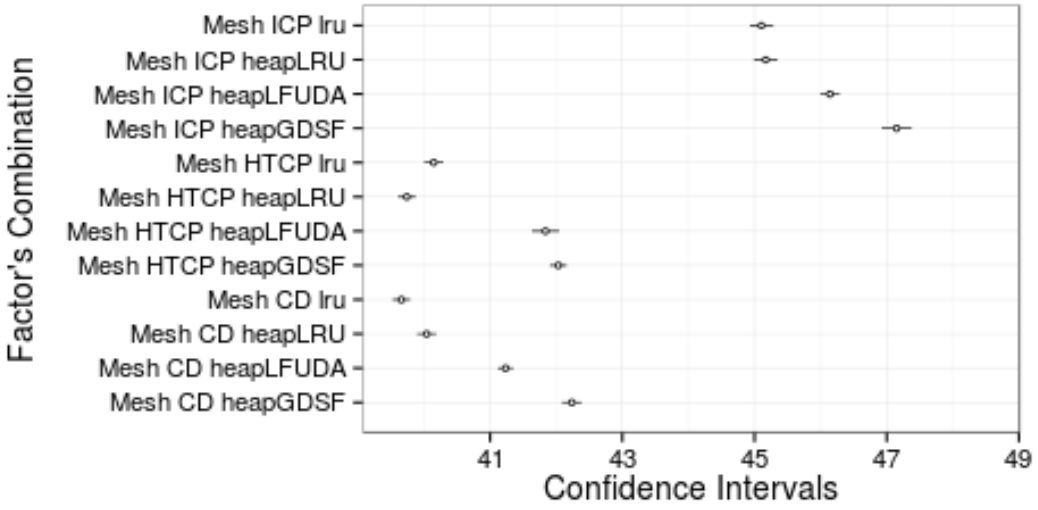


Figure 4.13: Hit ratio means 95% confidence intervals for factor's combination with mesh architecture

the performance of virtualized environments.

4.3.3 Phase 2: applying most favorable performability parameters

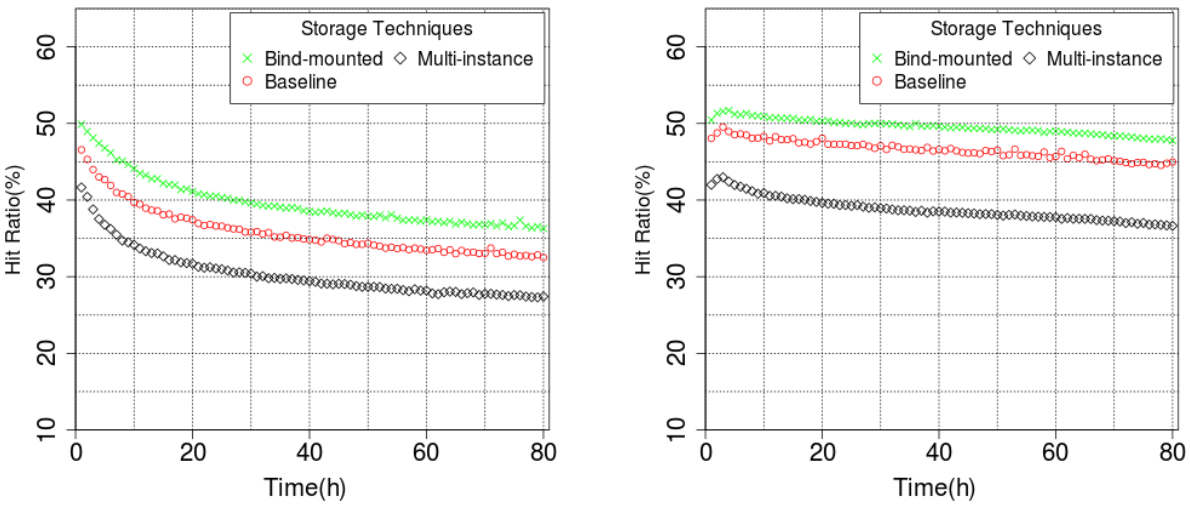
With level factors properly tuned, impact analysis on the performance of I/O bound web cache service, when offered in a virtualized environment, reached the second stage, in which it was decided to increment the analysis with one more non-virtualized web cache configuration: Multi-instance. Instead of only one Squid process, three of them were executed on an unique physical machine, as depicted in Node 03 of the Figure 3.1. With this new scenario, it was expected that greater coverage and fairness in comparisons would occur, especially regarding the implementation of server consolidation. A multi-instance configuration was employed with similar cache disk designs to the virtualized one, as shown in Figure 4.9: 4 disks, one for the local OS and others exclusively for each cache process instance.

A new round of experiments, with 1 GB of cache capacity per disk was applied. Results of hit ratio are shown in Figure 4.14a.

As can be observed, the screening fractional factorial experiment performed on Phase 1, with subsequent application of better levels of the factors, results in a higher hit ratio for a virtualized environment, both compared with baseline and as compared to multi-instance configurations.

Figures 4.15a and 4.16a are resultant confidence intervals for response time and throughput, respectively. The response time for virtualized storage technique was the lowest, whereas the throughput of virtualized solution was higher than multi-instance, although lower than the baseline one.

After, the disk cache capacity was increased by 10 times, resulting in 30GB of total cache space. This increment drops availability to 94.5366%. As a consequence, a deep investigation of



(a) Hit Ratio - 3GB of total cache

(b) Hit Ratio - 30GB of total cache

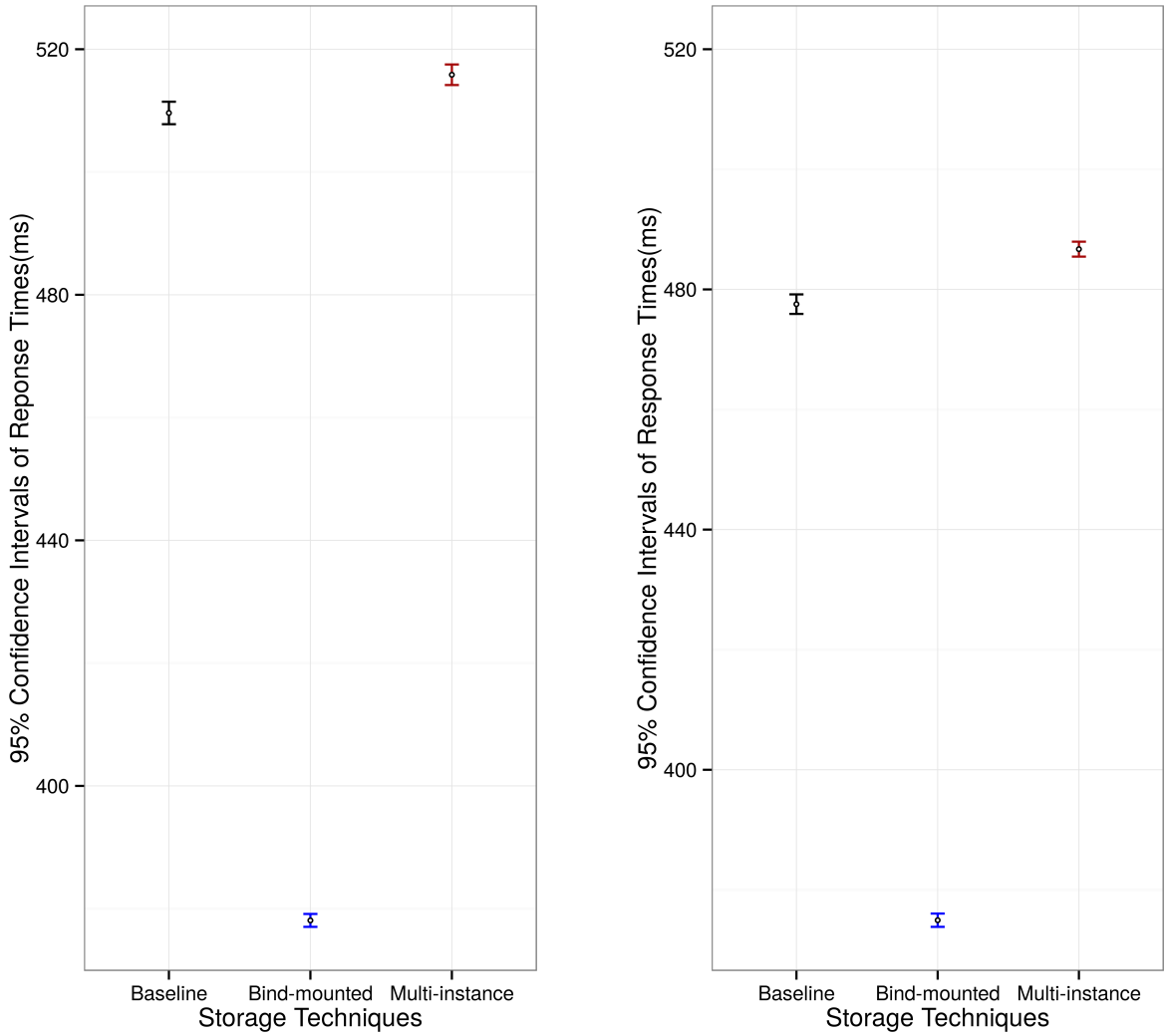
Figure 4.14: Hit Ratios comparative: 3GB and 30GB scenarios

possible performance improvements was made and a large combination of tunings was performed for several performance factors:

i. web cache server: it was updated, with the newest stable squid-3.4.2 release source code compiled and installed. Default configuration parameters were superseded. L1 and L2 cache levels of Squid server were tuned to contain a balanced number of stored files, resulting in a reduced search time for cached objects. Default values of 16 and 256 were replaced by 32 and 512, respectively, for higher cache size of 10GB per cluster's node.

ii. file systems in which the web cache server stores objects were updated. ext3 was replaced by ext4. Journaling feature, that is responsible for data consistency, was disabled. Without jbd2 (journaly process) running, it was noted that there was a significant lightening of overhead. As this work is related to cache systems, journaling can be disabled without catastrophic consequences of possible data corruptions. Furthermore, extents, delayed block allocation (delalloc), and multiblock allocation (mballoc) (KUJAU, 2013), that are performance improvements of ext4 that reduce latency, were also applied. *noatime* and *nodiratime* options were used during cache file system mounting. They avoid the update of inode access times on these file system, aiming at faster access.

iii. SMP affinity of IRQs: Foong et al.(FOONG; FUNG; NEWELL, 2004) presents the benefits of SMP affinity of IRQs. Interrupt-only affinity to NICs driver was applied, avoiding the use of CPU3, identified as the busier node's core. This movement was



(a) Response Times - 3GB of total cache

(b) Response Times - 30GB of total cache

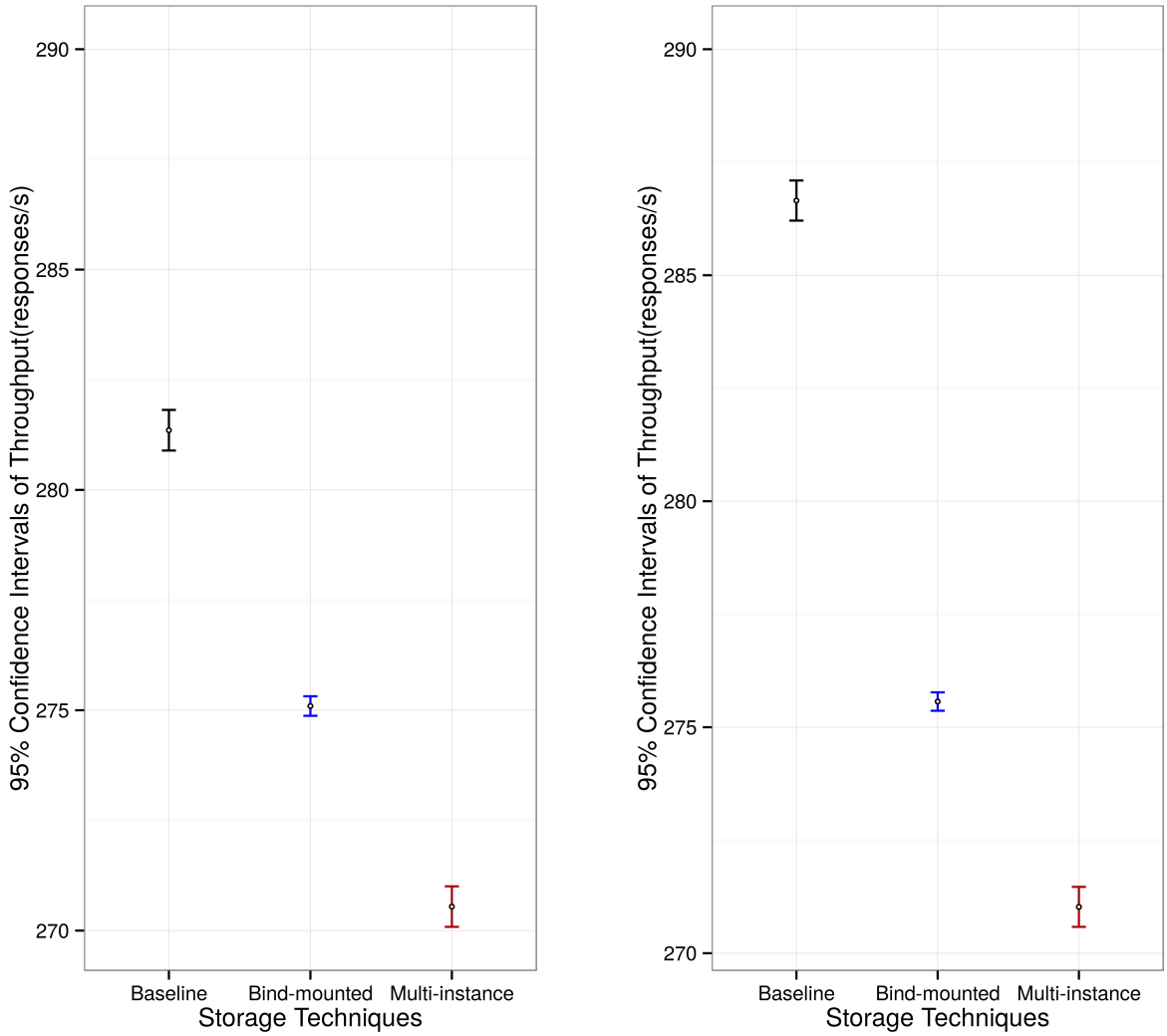
Figure 4.15: Response Times comparative: 3GB and 30GB scenarios

performed because CPU3 was presented with the most load during the execution of experiments. NICs driver affinity was set to not use CPU3.

iv. increase of web cache server processes scheduled priority: nice parameter was configured with the maximum allowed scheduled priority value, to heighten their time-slice.

After implementation of all performance improvements, none of 3 used storage techniques had failures caused by unresponsiveness, reaching 100% of availability during 80 hours of experiments.

As happened in the cluster with 3GB of storage capacity, hit ratio, depicted in Figure 4.14b, and response times, depicted in Figure 4.15b, presented a better performance with bind-mounted virtualized storage technique. The response times were, considering the average value, 27.36% higher in the non-virtualized baseline cluster and 29.81% higher in the non-virtualized



(a) Throughput - 3GB of total cache

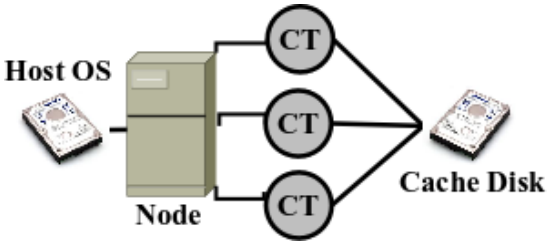
(b) Throughput - 30GB of total cache

Figure 4.16: Throughput comparative: 3GB and 30GB scenarios

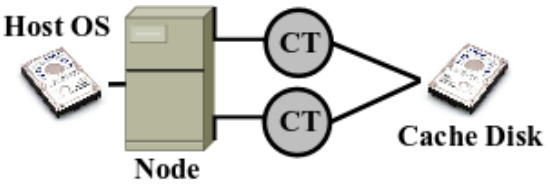
multi-instance cluster.

Throughput's behavior was also similar, with non-virtualized baseline cluster presenting a higher performance, as shown in Figure 4.16b. The average value of throughput in non-virtualized scenario was 4.02% higher than in virtualized cluster, and 5.76% higher than non-virtualized multi-instance cluster.

Following sections will present and discuss the case studies that were executed automatically, through the use of TISVEP protocol. The first case study analyzes the maximum number of containers that can be executed while five 9's availability is maintained. Moreover, through the acquired experience from the execution of the previous experiments, it was possible to reactivate the log mechanism of the web cache servers. The second case study was performed in order to determinate which of the three OpenVZ network modes of operation is most favorable to virtualized cluster performance. The third case study presents a comparison between the virtualized and non-virtualized clusters when half of the total storage capacity, 900GB, was used.



(a) 3 CTs to 1 HD relationship: A=99.0597%



(b) 2 CTs to 1 HD relationship: A=100%

Figure 4.17: Maximum number of CTs per HD for High Availability

Profiling of hardware events was used to improve the analysis of this case study. The fourth and final case study presents the behavior of both virtuliazed and non-virtualized clusters considering the total storage capacity: 1.8TB.

4.4 Case study 1: CTs per HD

This case study aims at defining the maximum number of containers that can be served by a single hard disk while maintaining the five 9's of availability. As shown in Figure 4.9, to reach five 9's of availability, a relation of 1:1 between containers and hard disks was required. The scenario of 3 CTs per hard disk, as depicted in Figure 4.17a, was previously tested, with a resulting availability of 99.0597%. This case study evaluated the scenario of 2 CTs per hard disk, as depicted in Figure 4.17b.

An experiment, as described in Table 4.9, was configured and executed. Their results are discussed below.

Table 4.9: Configuration for virtualized scenario: aiming at maximum number of CTs per HD for High Availability

Architecture	Inter-Cache Protocol	Replacement Policy	Cache size (per CT)	Duration (hs)
Mesh	ICP	heap GDSF	10GB	80

4.4.1 Two CTs per hard disk: discussion of the results

During the experiments of Section 4.2.3, the web cache server log functionality was disabled. The goal was to reduce the concurrence overhead to hard disks: the log files and the cached objects had been saving in the same hard disk. Without log files, it was not possible to make several performance analysis, such as determine bandwidth savings and Byte Hit Ratio.

In order to reactive the logging of the web cache servers, three actions were carried out:

1. the log files were stored on the hard disk of the host OS, eliminating the concurrency of disk access between the procedure to write the objects in the web cache and the writing of the log files;

2. a dedicated file system was created for storing log files;
3. the journal consistency mechanism of this dedicated file system was disabled, in order to eliminate the overhead associated with such a mechanism.

Together, these three actions enable to collected informations about Byte Hit Ratio (BHR), and in bandwidth savings. A better elucidation about the behavior of the analyzed web cache server clusters could be performed.

An experiment with the configurations shown in Table 4.9 was executed on the web cache cluster displayed in Figure 4.18. `A=100%` was reached during 80 hours of operation of such virtualized web cache server cluster.

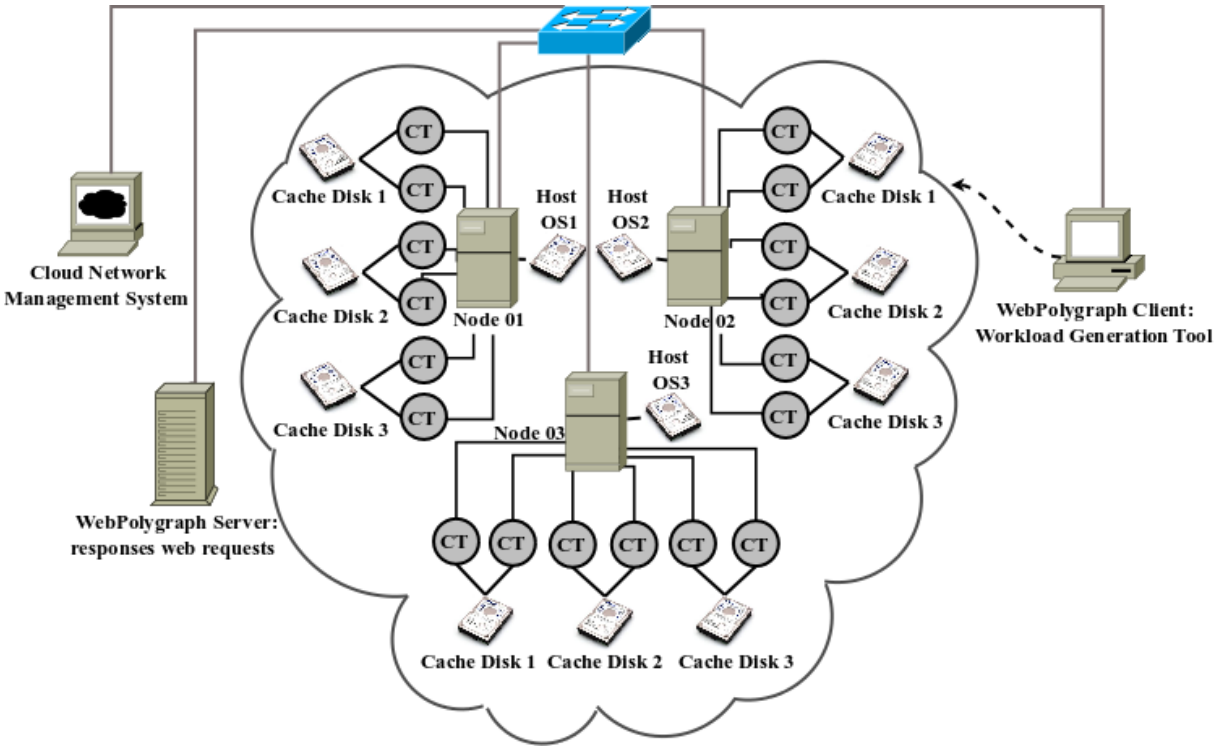


Figure 4.18: Testbed configuration with maximum number of CTs for high availability

All the SATA ports of the cluster were used to accommodate the twelve hard disks employed in the experiment: for the concerned cluster, it is no longer possible to include additional hard drives.

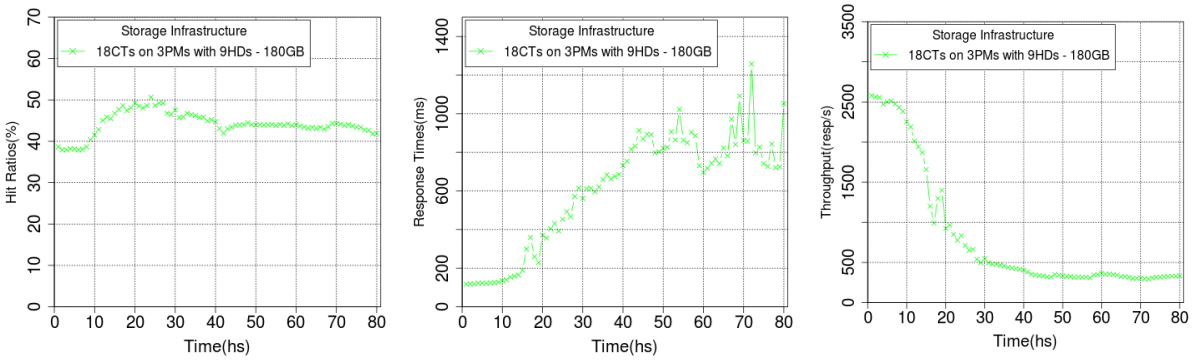
This hardware configuration allowed us to instantiate 18 containers, as depicted in Figure 4.18, resulting in a relationship of 2:1 between containers and hard disks. Even with two containers competing for the access of one hard disk, the availability remained at five 9's, proving the high availability of the proposed configuration.

The behavior of hit ratio, response time and throughput during the 80 hours of the experiment execution are shown in Figure 4.19.

The following similarities regarding the hit ratio behavior were identified between the current experiment (refer to Figure 4.19a) and that presented in Figure 4.14b:

1. hit ratio presented an initial behavior with low values, followed by an increasing trend to later reach a peak and finally show a declining tendency.
2. after 40 hours of operation, both clusters showed a hit ratio between 40% and 50%, with smooth variations inside this range.

These similarities will also be examined in Case Studies 3 and 4, in order to determine whether these hit ratio behavior patterns are maintained.



(a) Hit ratio behaviour **(b)** Response Time behaviour **(c)** Throughput behaviour

Figure 4.19: Performance metrics for 180GB of total storage cache capacity with 18 containers

Regarding the response time and the throughput, a steep degradation of these metrics can be seen in Figures 4.19b and 4.19c, respectively. Two possible factors that can lead to the reported degradation are: a high usage of the cache storage space and the consequent initialization of the replacement policy operation. Such arguments led us to the following hypothesis:

- in virtualized environments, as web cache storage space is depleted and replacement policy starts its operation, response time and throughput features a clear degradation.

Such hypothesis will be investigated in the 3rd and 4th case studies: additional parameters that can prove such hypothesis will be monitored.

To obtain the bandwidth savings and BHR, the message 305 `exportCacheLogs` of TISVEP was employed. It triggers a process responsible for storing the Squid log files on an external machine, as described in Chapter 3. These files contain information about the bandwidth savings and BHR.

The Squid log files have been processed through the use of the calamaris tool (BEERMANN; POPHAL, 2013), in order to determine the total bandwidth saving provided by the web cache cluster. Calamaris parses the log files of web proxy servers, including Squid, and generates reports about performance metrics.

The bandwidth savings of each container (CT1 to CT6), each physical machine (PM1 to PM3) that hosts the sets of six containers, as well as the cluster's bandwidth savings are displayed in the Table 4.10.

Table 4.10: Bandwidth savings for 18 containers and 180GB of cache

Bandwidth savings (GB)							
	CT1	CT2	CT3	CT4	CT5	CT6	TOTAL
PM1	61.852	85.156	80.033	81.546	81.147	82.374	472.108
PM2	50.734	83.882	66.320	80.262	73.656	74.061	428.915
PM3	47.545	63.666	53.830	55.931	53.185	53.145	327.302
						Total Cluster Savings (TB)	**1.228**

The Containers that were executed on PM1 saved 472.108GB of bandwidth; on PM2, 428.915GB; on PM3 327.302GB. With 18 containers and 180GB of overall cache storage capacity, a total of 1.228 Terabytes were saved during the 80 hours of conducted experiment. The total bandwidth savings will be used as a comparison factor regarding to following case studies of 80 hours.

Regarding byte hit ratio, the results are shown, in Table 4.11. The containers that were executed on PM1 presented a BHR of 13.20%, on average. For the containers executed on PM2 and PM3, the average BHR was 12.52% and 10.62%, respectively. The average BHR of the whole cluster was 12.11%.

Table 4.11: Byte Hit Ratio for 18 containers and 180GB of cache on cluster

Byte hit ratio(%)							
	CT1	CT2	CT3	CT4	CT5	CT6	Average
PM1	13.11	13.91	12.98	12.92	13.29	13.02	13.20
PM2	12.52	13.01	12.54	12.83	12.24	12.01	12.52
PM3	10.56	11.14	10.63	10.87	10.39	10.15	10.62
						Cluster Average BHR (%)	**12.11**

4.5 Case study 2: Analysis of OpenVZ network modes of operation

The network virtualization of OpenVZ can be provided by 3 different modes of operation: bridge-based, routed-based, and through real network devices (a complete discussion about OpenVZ network modes of operation was performed in Section 2.4.1). This Case Study aims at determining which of these network mode of operation presents best performance for hit ratio, response time, throughput, and bandwidth savings of a web cache server cluster. Then, a set of experiments was designed, as shown in Table 4.12.

The column “Cache size per container(GB)” depicts that each container that forms the cluster provided 3GB of cache space. The second column, “number of replications”, shows that

Table 4.12: OpenVZ modes of operation experiment design

Cache size per container(GB)	number of replications	number of CTs per PM	number of PMs	Duration (min)
3	5	3	1	30

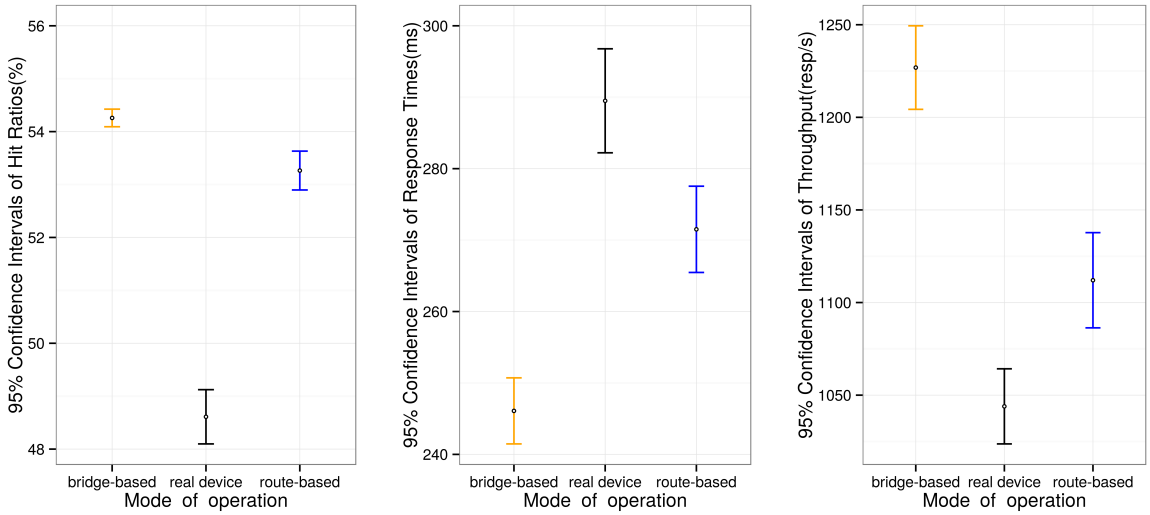
each of the OpenVZ network modes of operation (routed, bridged, and real device) experiment was replicated 5 times. They were executed on a web cache cluster of 3 containers deployed on a single PM. This restriction was applied because, to evaluate real network device mode of operation, a relationship of 1:1 is required between containers and network cards. Only one physical machine (Node 01 in Figure 4.3) of the testbed contained 4 network cards (one NIC was exclusively dedicated to administrative purposes).

The TISVEP configuration file was tuned to the designed scenario. Each experiment resulted in 1800 samples for each analyzed performance metric. Each sample mean was calculated. The results are presented through 95% confidence intervals of the means for each network mode of operation, as depicted in Figure 4.20.

Figure 4.20a shows that, for hit ratio, the most favorable OpenVZ mode of operation is the bridged-based (veth). The hit ratio for the bridged-based mode had its lower bound of the confidence interval above 54%. It is a higher value in comparison with routed-based (venet), whose upper bound of the hit ratio confidence interval were below 54%. The worst performance was observed for real network devices, whose upper bound of the confidence interval was below 50%.

Regarding response times, a similar ranking of modes of operation was observed, as shown in Figure 4.20b. The bridged-based mode of operation answered faster: below 260ms. The routed-based operational mode was was the second fastest, with bounds of the confidence interval between 260ms and 280ms. Real network devices presented slower response times, with the lower bound of the confidence interval above 280ms.

Finally, throughput obeys the previous performance metric behaviors, as depicted in Figure 4.20c. The virtualized web cache cluster is capable of generating the highest throughputs using bridged-based network operational mode, with more than 1175resp/s, followed by routed-based, with bounds of the CI between 1150resp/s and 1075resp/s, and by real network devices, whose upper bound of the CI was below 1075resp/s.



(a) Hit Ratio(%) **(b)** Response Time(ms) **(c)** Throughput(resp/s)

Figure 4.20: OpenVZ modes of operation analysis through web cache performance metrics behaviour

Regarding bandwidth savings, 95% confidence intervals are depicted in Figure 4.21. As was observed to all the other performance metrics, bandwidth savings also presented the best results to bridge-based mode of operation.

The mean values of bandwidth savings are depicted in Figure 4.21. The relative difference of the means between bridge-based and route-based modes of operation reveals that bridge-based performed 12.55% better than route-based mode, whereas, regarding real network devices, bridge-based presented a 34.51% better performance.

Then, as the goal of this Case Study was to determining which of the network mode of operation would present best performance for hit ratio, response time, throughput, and bandwidth savings of a web cache server cluster, based on the performed analysis, it is possible to say that bridge-based operation mode performs better for all the performance metrics. It will continue to be used in the further case studies.

4.6 Case study 3: Scaling and profiling web cache server

In previous experiments, a maximum of 10GB of storage per HD was employed. It corresponds to less than 10% of available space. In this case study, it was decided: (i) to increase the HD size from 10GB to 100GB in order to analyze the behavior when scaling the cache server storage; and (ii) to increase the memory from 4GB to 8GB, in order to avoid Out of Memory (OOM) failures[2].

[2]The OOM failure means that there was not enough available memory to execute the process.

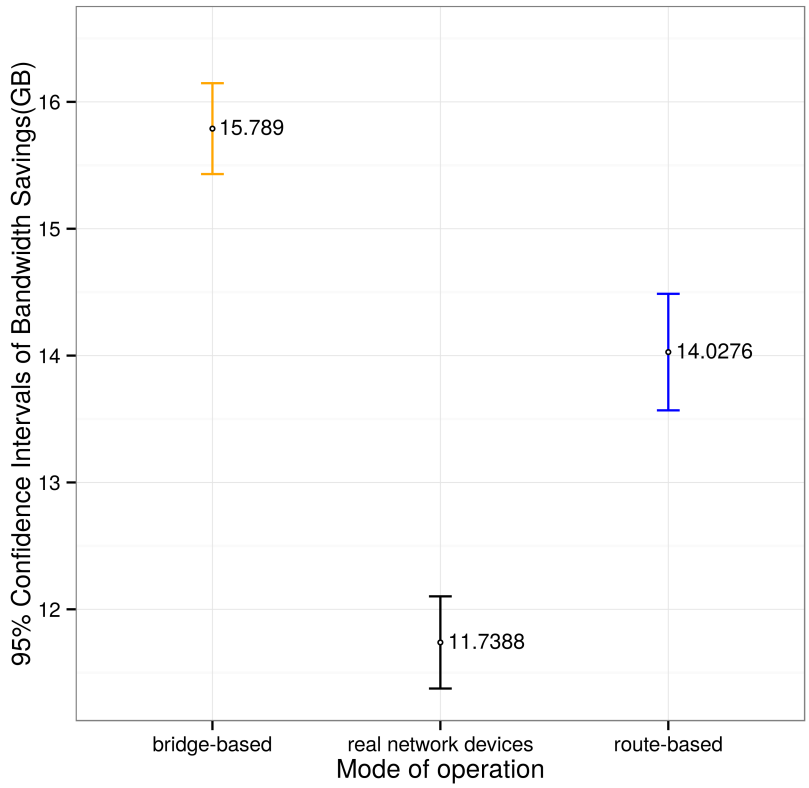


Figure 4.21: Bandwidth savings for OpenVZ network modes of operation

4.6.1 Performability analysis with 900GB of storage

The purpose of this experiment is to answer the following questions: (i) is it possible to maintain five 9's of availability with 900GB of cache storage? (ii) will the virtualized environments produce better performance metrics with storage scaling? and (iii) is profiling a suitable technique to investigate the overheads of virtualized environments?

The Table 4.13 depicts the design of the experiment. 9 containers (3 CTs per PM) with 100GB were employed, and the experiment was executed during 80 hours. The workload is the same of previous experiments in order to make a fair comparison.

Table 4.13: 900GB experiment design

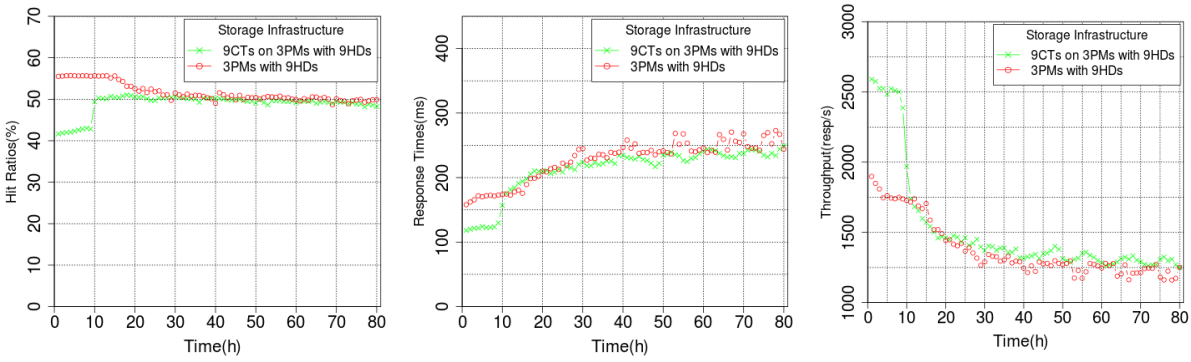
Cache size per containers(GB)	number of PMs	number of CTs per PMs	number of replications	Duration (h)
100	3	3	1	80

As result of this scenario, the observed virtualized web cache server cluster, in a relationship of 1:1 between containers and DAS hard disks, had no failures, obtaining A=100%, i.e., five 9's of availability was accomplished in a virtualized environment. Next, the same experiment was executed in a non-virtualized cluster, also resulting A=100%.

Regarding to performance analysis, the hit ratio is shown in Figure 4.22a. Here, a transient state means that the cache is not full (during the first 10 hours of our experiment). As

it can be seen, the 9CTs presented worst hit ratio than 3 PMs in non-virtualized environment during the transient state, and after the 30th hour, their behavior can be considered similar.

The response time is shown in Figure 4.22b and 9 CTs had better response time than 3 PMs during almost all experiment. Figure 4.22c shows the throughput and 9 CTs also presented better performance when compared to 3 PMs.



(a) Hit ratio behaviour **(b)** Response Time behaviour **(c)** Throughput behaviour

Figure 4.22: Performance metrics for 900GB of total storage cache capacity

In order to understand the sharp variation in CT behaviors after the transient state, an analysis of the Storage Swap capacity was performed, that is one of many parameters monitored using TISVEP (refer to Chapter 3), and reports the amount of data cached by each server.

Figure 4.23 shows Storage Swap reached 95% of capacity in the period between the 9th and 10th hours for all CTs. Comparing Figure 4.22a and Figure 4.23, one can note storage swap capacity affected the metrics behavior.

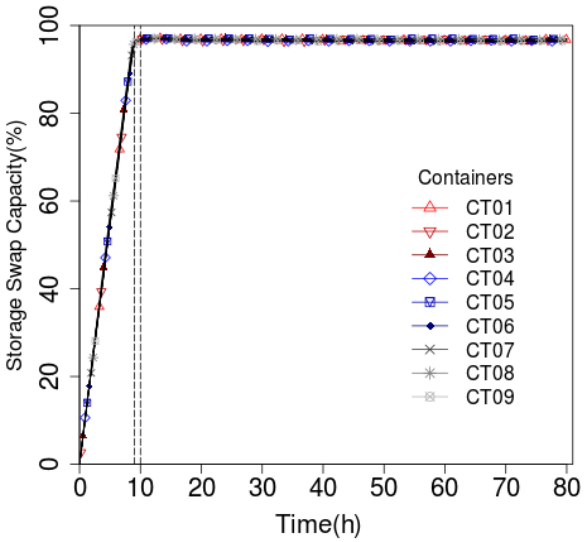


Figure 4.23: Cache storage capacity over time - Containers (CTs) configuration

Storage swap capacity of 95% also affects response time and throughput. After 9 hours

of virtualized experiment, when the storage threshold of 95% was reached, response time and throughput suffer a steep degradation, as observed in Figures 4.22b and 4.22c, respectively.

To normalize how each performance metric behaves in non-virtualized and virtualized environment, the following equations to calculate the relative change (LILJA, 2005) was used, expressed in percent:

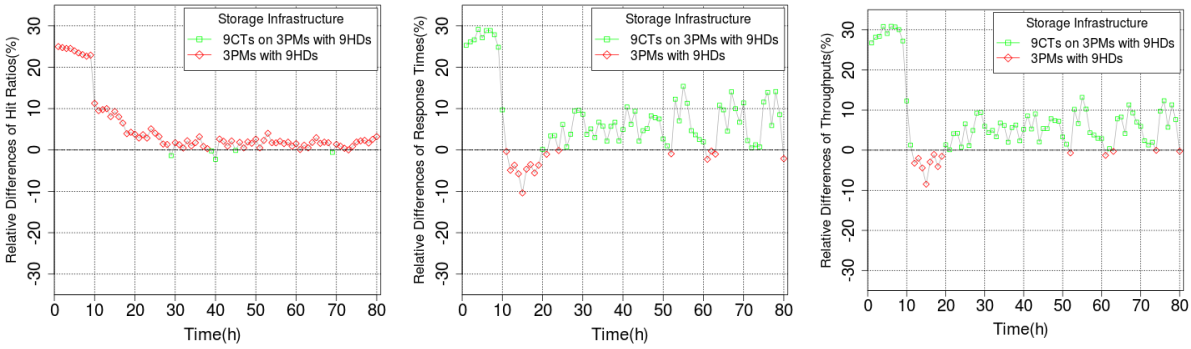
$$\texttt{RC\_HR} = \frac{(\texttt{HR\_NV} - \texttt{HR\_V})}{\texttt{HR\_V}} \times 100 \tag{4.1}$$

$$\texttt{RC\_RT} = \frac{(\texttt{RT\_V} - \texttt{RT\_NV})}{\texttt{RT\_NV}} \times 100 \tag{4.2}$$

$$\texttt{RC\_THR} = \frac{(\texttt{THR\_V} - \texttt{THR\_NV})}{\texttt{THR\_NV}} \times 100 \tag{4.3}$$

In Eq. 4.1, the RC_HR stands for relative change of hit ratio, where HR_NV and HR_V are hit ratio values for non-virtualized and virtualized clusters, respectively. The same is applied to response time in Eq. 4.2 and to throughput in Eq. 4.3.

Figure 4.24 depicts relative changes related to hit ratio. The 80 points related to each 1 hour of experiment. Non-virtualized clusters presented hit ratio better than virtualized. On the other hand, regarding to response time and throughput, virtualized cluster outperforms non-virtualized one.



(a) Hit ratio behaviour **(b)** Response Time behaviour **(c)** Throughput behaviour

Figure 4.24: Relative Changes of performance metrics for 900GB of total storage cache capacity

Table 4.14 show the results about bandwidth savings for the current experiments. During 80 hours of experiments, non-virtualized clusters saved 2.609% terabytes more than virtualized solution: a better total absolute saving of 67GB.

During 80 hours of experiments, non-virtualized web cache server clusters saves 2.609% more terabytes than virtualized solution: a better total absolute saving of 67GB.

Table 4.14: Saved bandwidth(TB)

Cache infrastructure	Bandwidth savings (TB)
non-virtualized cluster	2.568
virtualized cluster	2.501

4.6.2 Profiling web cache server clusters

Three TISVEP messages are related to enabling profiling on Squid processes. Processing of a 306 message `startupProfiling` in a target destination triggers an `ocount` process. It is an event counting tool of the `oprofile` system-wide profiler for Linux systems. Processing of a 307 message `stopProfiling` in a target destination halts `ocount` processes, whereas processing of message 308, `exportProfileData`, saves resulting profiling data in an external storage for further analysis.

Two hardware events were collected: CPU_CLK_UNHALTED and L2_CACHE_MISSES. For each time interval of one minute, the resulting counts from such hardware events are collected and stored.

- **CPU_CLK_UNHALTED**

This event collects the number of CPU cycles running outside of halt state. The number of samples collected in this routine is proportional to the time spent by the processor to execute instructions. The more samples collected, the more time the processor has spent executing those instructions.

Figure 4.25 depicts the CPU cycles comparison between web cache server on containers and on physical machines, node by node. The container results were grouped in the same bar in order to make a clear comparison with physical performance. As can be observed, each container, singly, presents less unhalted clocks than the physical machine; and all together sums more.

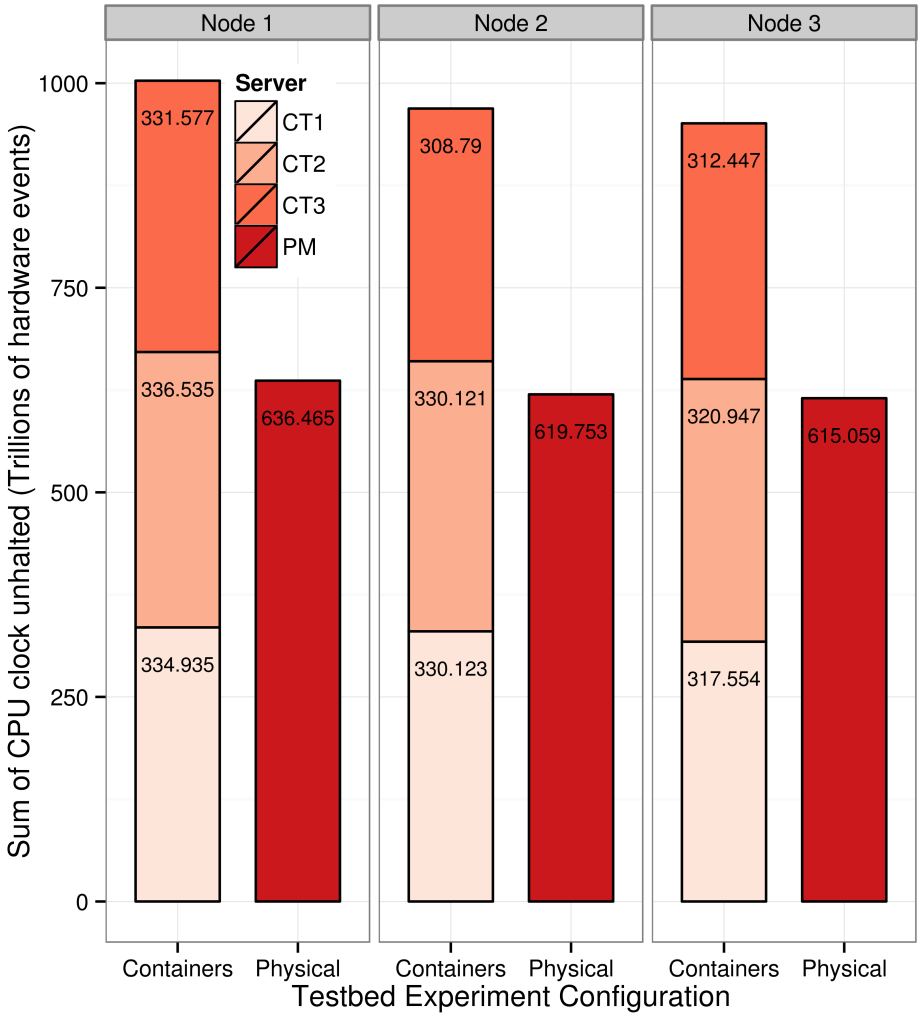


Figure 4.25: CPU_CLK_UNHALTED: 3PMs x 9CTs on 3PMs - 900GB of total cache space

Analyzing the Figure 4.25, it is possible to conclude that virtualized web cache server processes are able to execute their instructions in a smaller time than non-virtualized.

In Node 1, comparing containers individually, the CT2 presents the highest clock achieving 336.535 x 10^{12} events, while PM marks 636.465 x 10^{12}. Unhalted clock of physical machine in Node 1 was 89.12% bigger than CT2. A similar behavior can be observed in Nodes 2 and 3, where PM unhalted clocks perform 87.73% and 91.64% worst than the higher container results. The better results of unhalted clocks in web cache server cluster on container clusters represent an impact factor for better response times and throughput in this case study.

For virtualized servers, web server processes are executed simultaneously consuming more clocks and avoiding resource idleness; consequently it results in a better usage efficiency.

The cumulative CPU_CLK_UNHALTED represent the second advantage of virtualized cluster: a better efficiency in resources usage, which is favorable to the server consolidation process. For Node 1, cumulative CPU_CLK_UNHALTED for containers cluster is 57.6% better than the non-virtualized cluster. For Nodes 2 and 3, the better efficiency of CPU usage for

containers cluster reached 56.36% and 54.61%, respectively.

- **L2_CACHE_MISSES**

This hardware event counts the number of times that a search in an L2 cache level results in a missing.

Figure 4.26 shows the L2_CACHE_MISSES for each node. Each virtualized server presented L2_CACHE_MISSES smaller than each physical machine.

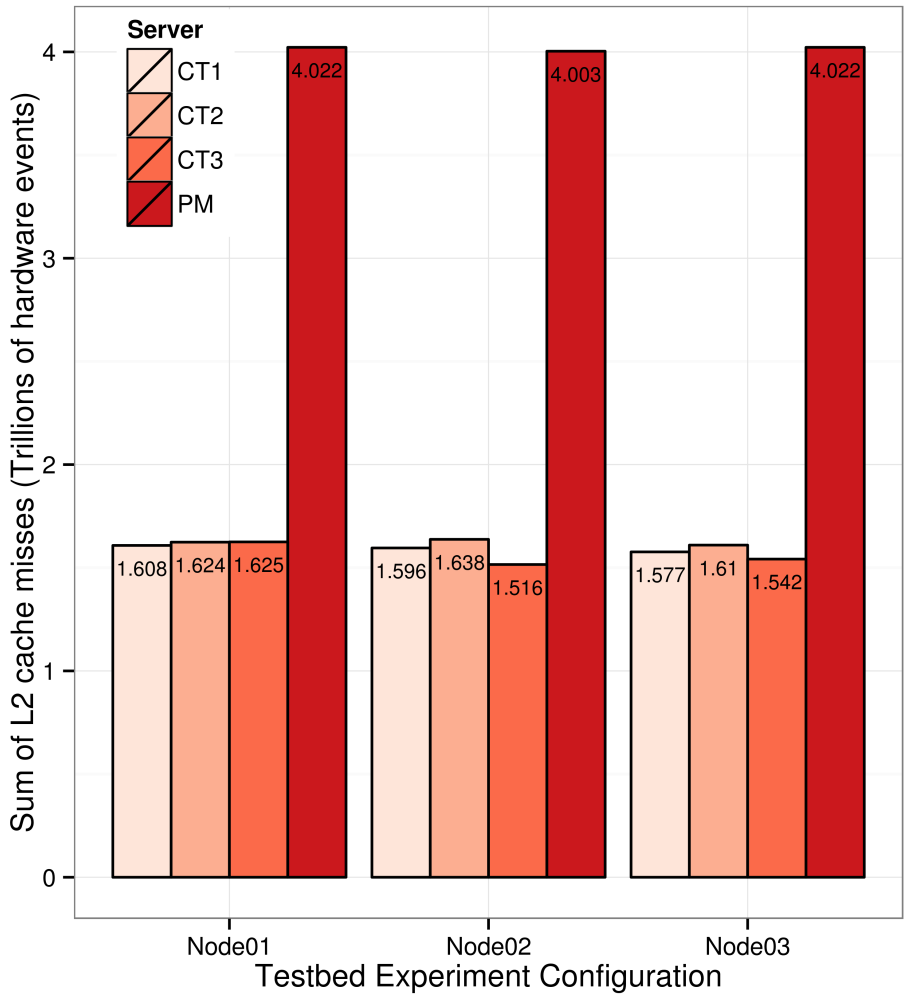


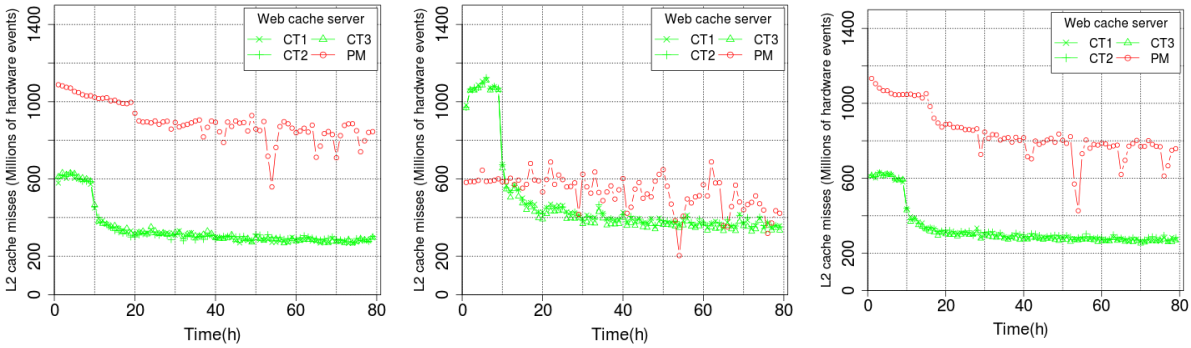
Figure 4.26: L2_CACHE_MISSES: 3PMs x 9CTs on 3PMs - 900GB of total cache space

The results of Node01 showed CT3 reached the highest value of L2_CACHE_MISSES, with 1.625 x 10^{12}, while PM presented 4.002 x 10^{12}, a 147.48% higher value of L2 cache misses. Similar behavior was observed on Nodes 02 and 03, where CT2 presented smaller L2_CACHE_MISSES values. Node02 and Node03 behaviors were 144.47% and 149.86% higher than CT2s servers, respectively.

As can be noted, L2_CACHE_MISSES event has an inversely proportional relationship with response times and throughput for a container-based cluster. It is another evidence for better behavior of response time and throughput in comparison with a non-virtualized cluster.

The behavior of the L2_CACHE_MISSES event, during the 80 hours of experiments, is shown in Figure 4.27. Each point in Figure 4.27. represents the average of 60 collected values, one per minute, of L2_CACHE_MISSES. As one can noted, Node 01 and Node 03 presented the lowest L2 cache misses during all experiments, whereas Node02, presented the worst averages during the transient state, and in other two moments.

Such analysis reinforce the evidence of a relationship between lower L2_CACHE_MISSES and better behavior of response time and throughput for virtualized web cache cluster, in comparison with non-virtualized.



(a) Node 01 **(b)** Node 02 **(c)** Node 03

Figure 4.27: L2 cache miss hardware events behaviour for 900GB of total cache capacity

4.7 Case study 4: Full spool performability analysis

This case study aims to answer the following questions: (i) is it possible to maintain five 9's of availability for all the storage capacity? (ii) will virtualized environments produce better performance with full storage capacity? and (iii) what can profiling reveal about overheads of virtualized environments in the full storage capacity scenario?

According to Squid web cache documentation[3], one recommends saving 20% of the disk space for operational purposes; and according to (WESSELS, 2004), a Squid web cache server requires a small amount of memory for objects on the disk since it uses the memory as an index to the on-disk data. For instance, in systems with 64 bit CPUs, each object takes 112 bytes. As mentioned in Section 3.2.3, each physical machine was composed of 3 dedicated hard drives for cache storage, each one with 250GB. In this way, to answer the previous questions, experiments were elaborated using all storage capacity of our cache servers in order to verify their behaviors.

A default applied practice is to evaluate the non-virtualized before the virtualized scenarios in order to create a baseline reference without virtualization. Take this into consideration, in this case study with full storage capacity per hard disk, were executed the same experiments done in Section 4.6, but following these steps:

[3] `http://www.squid-cache.org/Doc/config/cache_dir/`

1. PM-1: two replications with 3PMs, restarting the cluster nodes;
2. VIRT-1: one replication with 9CTs, restarting the cluster nodes;
3. PM-2: one replication with 3PMs, without restarting the cluster nodes.

In order to monitor the cache server operation, the following steps were defined:

- application of a TISVEP message, named `startCacheMonitor`, for starting a cache monitoring process in each PM;
- the Resident Set Size (RSS) for a Squid process was stored. The maximum RSS is the maximum amount of physical memory used by the process at any time. Squid's process size may be larger than the RSS, in which case some parts of the process are actually swapped to disk (WESSELS, 2004).

As result, it was observed the PM-1 replications did not reach the five 9's of availability due to the amount of primary memory in each PM. Table 4.15 shows the Maximum RSS and the Final RSS for each node in both replications of PM-1. At the end of the first replication, the Maximum RSS reached 11.32GB, 11.54GB, and 11.56GB, for Node 01, Node 02, and Node 03, respectively. Therefore, the availability decreased from `A=100%` to `A=99.9754%`, registering 71 failures. Here, it is important to highlight only 2 failures can occur to reach five 9's of availability during 80 hours of experiment.

For the second replication, the maximum RSS reached 8.09GB on Node 01, 10.96GB on Node02, and 11.11GB on Node03 when the first failure occurred. In total, 34 failures were registered during the 80 hours of this second experiment, resulting in `A=99.987%`.

Table 4.15: Maximum RSS for non-virtualized full spool experiments

Squid Maximum RSS Analysis - PMs		
1.8 TB - First Replication		
	Max RSS on first failure (GB)	Final Max RSS(GB)
Node 01	3.78	11.32
Node 02	3.01	11.54
Node 03	5.09	11.56
1.8 TB - Second Replication		
Node 01	8.09	11.26
Node 02	10.96	11.53
Node 03	11.11	11.56

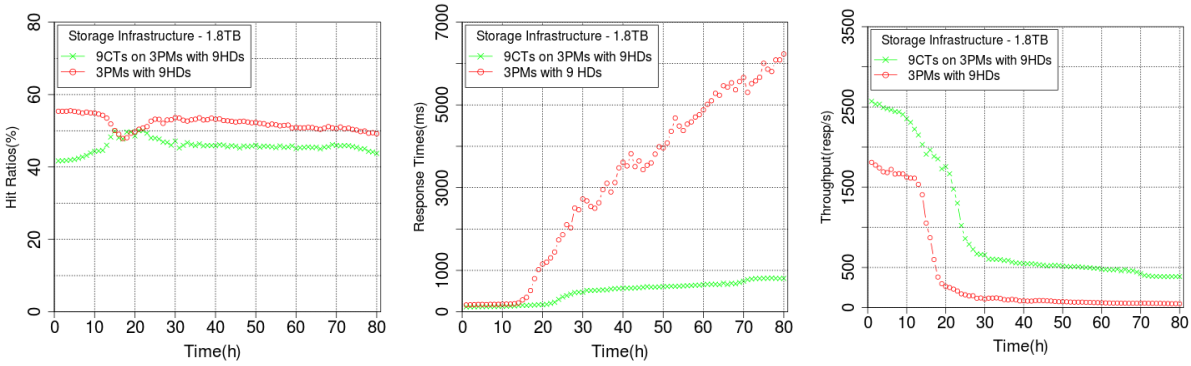
As can be noted in Table 4.15, at the end of experiments, the Squid processes of all physical machines presented maximum RSS largest than the available memory, requesting for swap utilization. So, the source of the failures was environmental, as discussed in Section 2.1.2.

Contrary to the default procedure of evaluating the virtualized environment after checking five 9's of availability in the non-virtualized one, the next experiment of this case study was

conducted in a virtualized environment. VIRT-1 resulted in availability of A=100% even without a corresponding non-virtualized cluster reaching such result. It was the first experiment using 1.8TB of cache and successfully resulted in five 9's of availability.

Finally, the PM-2 was conducted; remembering the main difference is the node restarting process that was not executed. The main objective of restarting cluster nodes is to flush primary memory. Even performing a set of well known commands to drop caches (MORREALE, 2008), the amount of free primary memory did not return to higher values than 7GB, that was the observed free RAM after machines restarting. In this experiment, PM-2 reached A=100%. So, it was possible to compare the performance of full spool capacity on virtualized and non-virtualized clusters.

Figure 4.28 shows the behavior of hit ratio, as well as response times and throughput. During the transient state, the hit ratio presented a similar behavior to the previous case study. Nevertheless, when cache spool was completely filled, the hit ratio for the non-virtualized cluster presented a better performance (Figure 4.28a). In contrast, response time and throughput were favorable to the virtualized cluster in both transient and persistent states, as can be noted in Figures 4.28b and 4.28c, respectively.



(a) Hit ratio behaviour **(b)** Response Time behaviour **(c)** Throughput behaviour

Figure 4.28: Full Spool performance metrics - 1.8TB of total cache capacity

Through the monitoring of Maximum RSS on the PM-2, a different result was obtained in comparison with the PM-1: the maximum RSS did not increase beyond the machine's total RAM. The Maximum RSS for Nodes 01, 02, and 03 reached 4.727GB, 4.834GB, and 4.866GB, around 17^{th}, 15^{th}, and 15^{th} hours of experiment, respectively, and remained stagnant in these values until the end of the experiment. During these periods, response time and throughput were sharply degraded. So, although availability reached 100%, the performance of non-virtualized cluster was lower than expected, due the stagnation of maximum RSS.

With maximum RSS stagnation, the response times for the non-virtualized cluster grew boundlessly, and throughput was below 100 responses per second after the 38^{th} hour of the experiment.

The bandwidth savings for the non-virtualized cluster were also impacted. Table 4.16

summarizes the total bandwidth savings for virtualized cluster, which reached 1.571TB, while Table 4.17 is related to non-virtualized cluster, which reached 801TB. Virtualization cluster was 93.95% better than non-virtualized.

Table 4.16: Bandwidth savings for 9CTs on 3PMs with 1.8TB of total cache storage

Bandwidth savings (GB)				
	CT1	CT2	CT3	**Total**
PM1	178	168	170	507
PM2	176	177	177	530
PM3	177	179	178	534
Total Cluster Savings (TB)				**1.571**

Table 4.17: Bandwidth savings for 3PMs with 1.8TB of total cache storage

Bandwidth savings (GB)	
PM1	252
PM2	277
PM3	281
Total Cluster Savings	**810**

4.7.1 Profiling web cache server clusters with storage capacity of 1.8TB

This Section presents the analysis of CPU_CLK_UNHALTED and L2_CACHE_MISSES events for web cache server clusters with storage capacity of 1.8TB.

- **CPU_CLK_UNHALTED**

Table 4.18 shows a comparison with CPU_CLK_UNHALTED results from Case Study 3. Such comparison aims to describe how RAM stagnation on non-virtualized cluster with storage capacity of 1.8TB affects the behavior of cycles outside of halt state. As can be noted, whereas CPU_CLK_UNHALTED for CTs cluster decreased 54.97% on average, for non-virtualized (PM) cluster, the decrease was 256.57%. Such outstanding difference also happens due to RAM stagnation on 1.8TB non-virtualized cluster.

Table 4.18: CPU_CLK_UNHALTED comparison: 900GB x 1.8TB of spool storage capacity. Values on columns 900GB and 1.8TB are multiplied by 10^{12}

CPU_CLK_UNHALTED Comparison						
	900GB		1.8TB		**Decrease**	
	PM	CTs	PM	CTs	PM	CTs
Node 1	636.4652	998.088	187.9870	641.2819	238.57%	55.64%
Node 2	619.7525	947.704	153.4187	618.8516	303.96%	53.14%
Node 3	615.0589	942.447	187.9869	603.668	227.18%	56.12%
				Average Decrease	**256.57%**	**54.97%**

Figure 4.29 shows the comparison of CPU_CLK_UNHALTED between web cache server processes in containers and physical machines with cache capacity of 1.8TB, node by node.

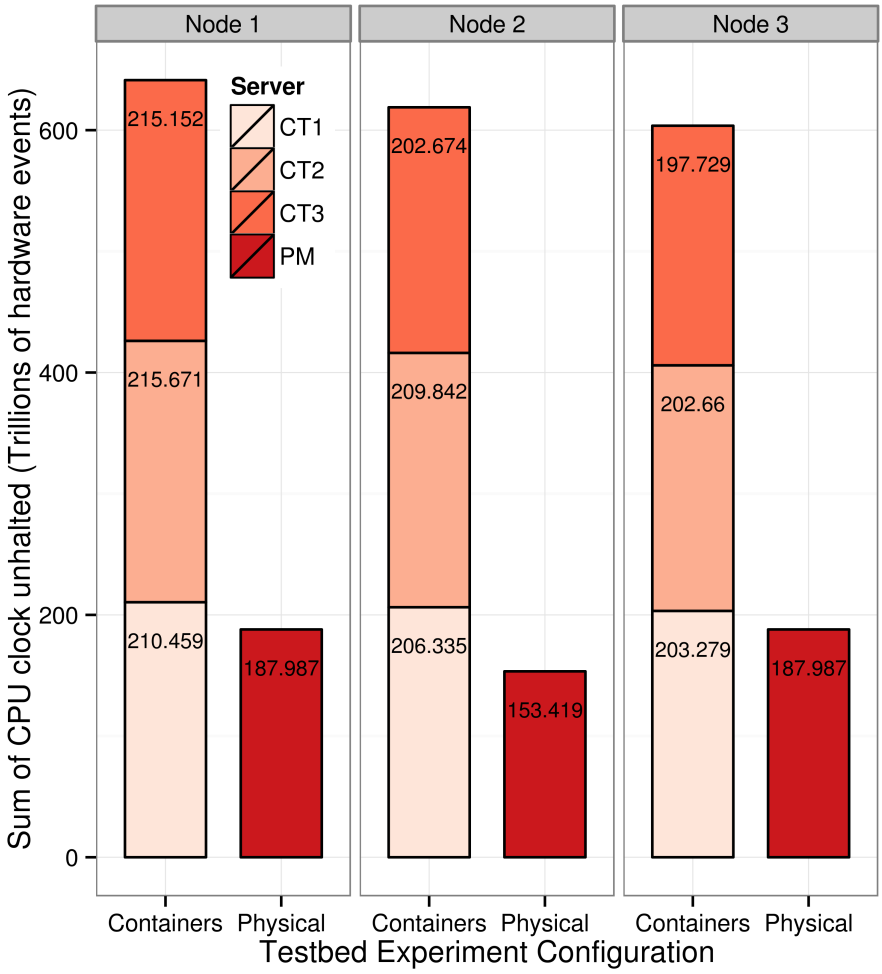


Figure 4.29: CPU_CLK_UNHALTED sum for 9CTs x 3PMs with 1.8TB of cache capacity

With 900GB of cache capacity, CPU_CLK_UNHALTED presented better performance for each of the individual virtualized servers than non-virtualized. However, for 1.8TB of cache capacity, non-virtualized cluster marked smaller summed values for cycles outside of halt state.

Aiming to explain reversal behavior of non-virtualized cluster for 1.8TB, the following evidences are presented:

- the difference in decreasing CPU_CLK_UNHALTED behavior between 900GB and 1.8TB experiments, presented in Table 4.18;
- strictly increasing response times for the 1.8TB non-virtualized scenario;
- low throughput in 1.8TB non-virtualized scenario;
- 93.95% worst behavior in bandwidth savings in the 1.8TB non-virtualized scenario.

Indeed, all these arguments are side effects of RAM stagnation.

Still in Figure 4.29, one can see that PM of Node1 presented 187.987 x 10^{12} CPU_CLK_UNHALTED, whereas CT2 (that marked lower summed value) reached 215.671 x 10^{12}; PM outperformed CT in 11.95%. Similar results were reached for Node1 and Node2, with better behavior for PM: 36.77% and 8.13%, respectively.

As expected, cumulative analysis of CPU_CLK_UNHALTED is more favorable for virtualized cluster. For Node1, summed values of the 3 containers reached 641.2819 x 10^{12} . For Node2, 618.8516 x 10^{12} and, for Node3, 603.668 x 10^{12} . So, comparing nodes web cache servers in a virtualized cluster with non-virtualized, CPU_CLK_UNHALTED of virtualized performs better for 1.8TB scenario.

So, per node web cache servers' joint behavior of CPU_CLK_UNHALTED that compound a virtualized cluster, in comparison with individual non-virtualized nodes, performs better for 1.8TB scenario, with a favorable behavior of 241.17%, 303.37%, and 221.11% for Nodes 01, 02, and 03, respectively.

- **L2_CACHE_MISSES**

Concerning to L2_CACHE_MISSES, the behavior was similar in both 900GB and 1.8TB clusters: the summed values by each container were smaller than the summed values of each physical machine.

A decreasing comparison between these two scenarios was also conducted , aiming to observe how large was the difference in L2 cache miss behavior. Table 4.19 presents this comparison with decreasing percentage values expressed as average. As can be noted, percentage decreasing values of L2_CACHE_MISSES are similar to ones presented by CPU_CLK_UNHALTED in Table 4.18: the decrease for non-virtualized cluster was about 5 times greater than containers cluster.

Table 4.19: L2_CACHE_MISSES summed values comparison: 900GB x 1.8TB of spool storage capacity. Values on columns 900GB and 1.8TB are multiplied by 10^{12}

L2_CACHE_MISSES Summed Values Comparison										
	900GB				1.8TB				Decrease	
	PM	CT1	CT2	CT3	PM	CT1	CT2	CT3	PM	$\overline{\text{CTs}}$
Node 1	4.022	1.608	1.624	1.625	1.345	1.042	1.069	1.088	**199.03%**	51.86%
Node 2	4.003	1.596	1.638	1.516	1.092	1.044	1.055	1.024	**266.57%**	52.06%
Node 3	4.022	1.577	1.610	1.542	1.087	1.037	1.035	1.033	**270.01%**	52.30%
								Average Decrease	**245.20%**	**54.97%**

Figure 4.30 shows the sum of L2_CACHE_MISSES by nodes. Each virtualized web cache server presented total of L2_CACHE_MISSES lower than each physical machine.

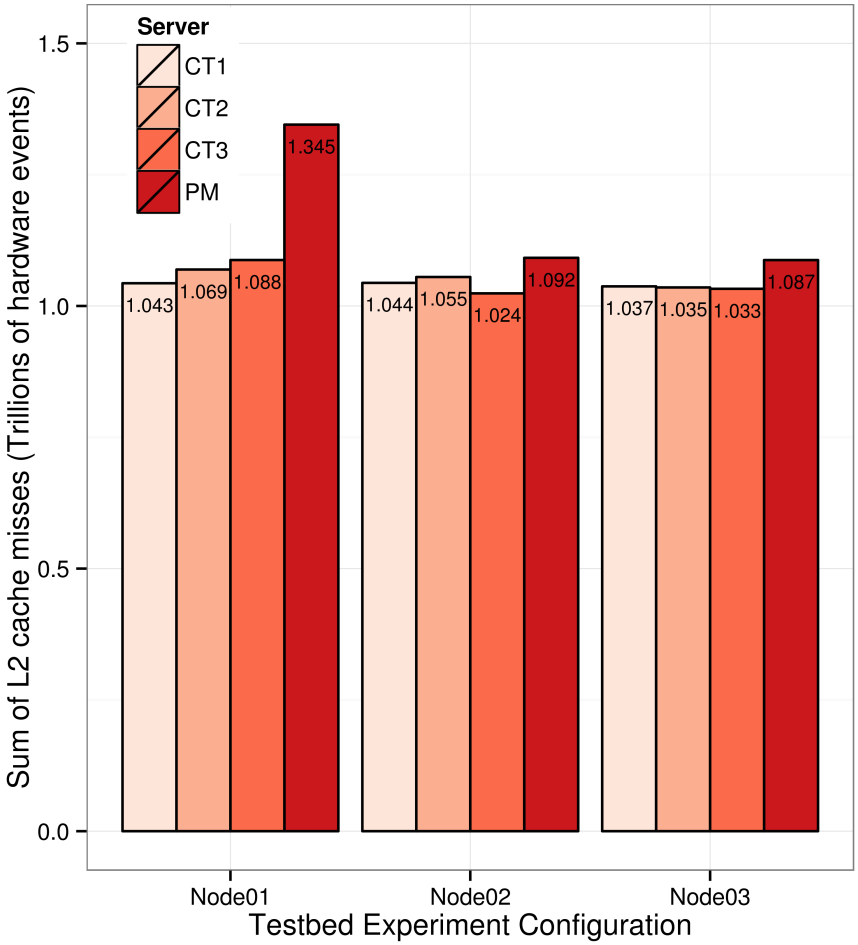
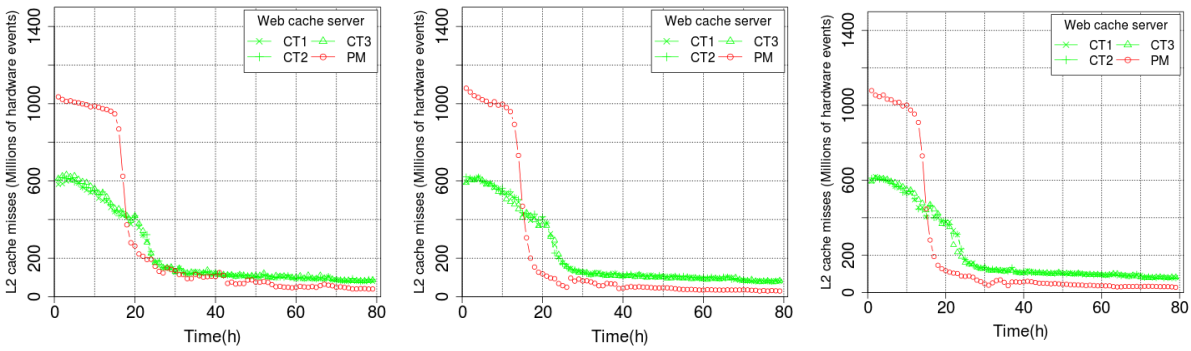


Figure 4.30: L2_CACHE_MISSES sum for 9CTs x 3PMs with 1.8TB of cache capacity

In Node 01, the higher summed value of L2_CACHE_MISSES for the virtual cluster was reached by CT3, with 1.088 x 10^{12} events, while PM marked 1.345 x 10^{12}. So, PM presented

a L2 cache misses value 23.62% higher. Nodes 02 and 03 presented a similar behavior, with CT3 of Node 02 reaching the smallest summed values of L2 CACHE MISSES. Concerning the L2 cache misses, the PM behaviors of Node02 and Node03 were 3.51% and 4.82% higher than CT3s servers.

As depicted in Figure 4.31, the behavior of the L2_CACHE_MISSES,during the transient state, was better for virtualized servers. L2_CACHE_MISSES, after the transient state, decreased sharply for non-virtualized cluster. Apart from reducing performance due to replacement policy operation, L2_CACHE_MISSES also decreased due to limited resources, specifically primary memory.



(a) Node 01 **(b)** Node 02 **(c)** Node 03

Figure 4.31: L2 cache miss hardware events behaviour for 1.8TB of total cache capacity

During the transient state, when spool capacity was not yet completely filled, the behavior of the L2_CACHE_MISSES was better for virtualized servers. After the transitory state, L2_CACHE_MISSES decreases sharply for non-virtualized cluster. Apart from reducing performance due to replacement policy operation, L2_CACHE_MISSES also decreased due to limited resources, specifically primary memory.

4.8 Summary of recommendations

The two main reasons for adoption of a web cache server system are:

- to reduce response times of requested objects, improving user experience;
- to save external bandwidth.

Regarding these two reasons and based on experimental results presented in the current chapter, the following list of recommendations about operational behavior of studied I/O bound web cache server clusters are made:

1. Use of server virtualization favors faster response times on I/O bound web cache server clusters, since increasing the number of nodes in a cluster results in the

decrease of response time for the provided service. Through the application of server virtualization, it was possible to reach 18 virtualized servers, in contrast to the limited number of 3 physical servers. After initial comparative experiments, described in Section 4.3.1, all others resulted in better response times for virtualized clusters.

2. As Internet Cache Protocol (ICP) scales accordingly to growth of virtualized servers, it is a suitable option of intercommunication between web cache servers that forms the cluster. Even with the increasing number of web cache servers that form the cluster, the response times perform better for the studied scenarios of 9 web cache servers provided on containers versus 3 web cache servers provided on physical machines.

3. Regarding bandwidth savings, container-based virtualized web cache clusters perform similarly and even better, in comparison with non-virtualized clusters. For the two studied large scale storage web cache clusters, with 900GB and 1.8TB of cache space, a virtualized cluster saves:

    - 2.609% less bandwidth than non-virtualized with the former (900GB);
    - 93.95% more bandwidth than non-virtualized with the latter (1.8TB). This is due to RAM stagnation problem of non-virtualized cluster that had not been arising with virtualized cluster.

Additionally for the main reasons of web cache server usage, the following implications are also relevant:

- In comparison with full virtualization technology, the use of container-based virtualization is preferred to I/O bound clustering due to its lower overhead.
- It was not possible to reject profiling hardware events as an effective investigative technique to justify better performance of virtualized environments. In contrast, the concerned hardware events were useful as a way to evidence reasons of better performance in virtualized environments when compared to non-virtualized ones.

5

Related Works

This chapter presents a set of works whose subjects are related to this dissertation. Section 5.1 presents works that applies measurements aiming at evaluating the performance in environments of server virtualization. To demonstrate the infrastructure on which each work was performed, the physical components of the testbeds are presented with as much detail as provided. Next, Section 5.2 presents works related to experiments automation in virtualized environments. Finally, a discussion comparing this dissertation and related works is conducted in Section 5.3.

5.1 Performance Evaluation of Server Virtualization through Measurements

In (DUAN; GU, 2010), the authors proposed a hybrid Web proxy cache cluster. The nodes that form the cluster were divided in two groups: local and cloud nodes. When the use of resources in the local nodes exceeds the established threshold, the proposed system adds cloud nodes to the Web proxy cache cluster. A performance comparison of response times between 8 local only nodes and 16 hybrid nodes (8 local and 8 from cloud) clusters was achieved, with favorable response times for the hybrid cluster. Even arguing that the proposed solution has high availability, the authors did not present clear results about availability.

In the Xavier et al. (XAVIER et al., 2013) work, they argued that container-based virtualization can be a powerful technology for HPC environments and proposed a performance and isolation evaluation of recent container-based implementations. Moreover, the use of virtualization technologies in high performance computing (HPC) environments had traditionally been avoided due to their inherent performance overhead. However, with the rise of container-based virtualization implementations, such as Linux V-Server, OpenVZ and Linux Containers (LXC), it is possible to obtain a very low overhead leading to near-native performance. They conducted a number of experiments in order to perform an in-depth performance evaluation of container-based virtualization for HPC, analyzing CPU, memory, disk and network resources. The setup consisted of four identical Dell PowerEdge R610 with two 2.27GHz Intel Xeon E5520

processors (with 8 cores each), 8M of L3 cache per core, 16GB of RAM and one NetXtreme II BCM5709 Gigabit Ethernet adapter. Some depicted scenarios presented better performance for a container-based virtualization solution, such as disk write operations for Linux V-Server. As future work, they planned to study the performance and isolation of container-based systems for other kinds of workloads, including I/O bound applications that were performed in this dissertation.

Ye et. al (YE et al., 2010) performed an evaluation of the virtualization cost aiming at finding the performance bottlenecks when running HPC applications in a virtual cluster. Their work applied hardware profiling analysis through the use of oprofile and xenoprof. They used the HPC Challenge Benchmark suite to investigate the performance of several aspects of a cluster. The experiments were conducted on Dell 2900 PowerEdge servers, with 2 Quad-core 64-bit Xeon processors at 1.86 GHz. No information about the total RAM amount was provided. They investigated the virtualization overhead comparing the performance of Xen virtual machines, running both in para-virtualized and full virtualized modes, with a physical machine. The first results showed that paravirtualization caused a little performance loss in comparison with physical machine and that the L2 cache miss rate is one of the major reasons for degrading the performance of the virtualized environment. Furthermore, 5 repetitions of experiments in 16 nodes cluster was performed, aimed at investigating possible bottlenecks in a scalability study: the number of VMs scales from 1 to 16. Both full virtualization and paravirtualization had an increase in the L2 cache misses rate. The performance penalties, considering L2 cache misses, were 13.97% and 16.40% for paravirtualization and full virtualization, respectively. They concluded that virtualization does bring performance overheads for HPC applications, but within an acceptable range. Their work was entirely CPU-bound, applying the Xen hypervisor, and no observation about disk workloads was cited. No information was provided about the duration of the experiment. No analysis about failures or system downtimes was reported.

Huber et. al (HUBER et al., 2010) summarized the performance-influencing factors of several virtualization platforms. They presented an interesting classification of the major performance-influencing factors of virtualization platforms, dividing them into 3 classes: virtualization type, resource management configuration and workload profile. Aiming at evaluating such factors, two different scenarios were used during the conducted experiments, with Windows Server 2003 as both Host and Guest OSs: (i) to evaluate the overhead of the virtualization layer, an HP Compaq dc5750 machine with an Athlon64 dual-core 4600+, 2.4 GHz, 4 GB DDR2-5300 of main memory, a 250 GB SATA HDD and a 10/100/1000-BaseT-Ethernet connection; (ii) to evaluate the performance when scaling the number of VMs, a SunFire X4440 x64 Server, with 4*2.4 GHz AMD Opteron 6 core processors with 3MB L2, 6MB L3 cache each, 128 GB DDR2-667 main memory, 8*300 GB of serial attached SCSI storage and 4*10/100/1000-BaseT-Ethernet connections. For the first scenario (directly related to this dissertation), the performance degradation when switching from a native to a virtualized system remains around 4%. Even arguing that I/O virtualization suffers from significant performance overheads, they did not

conduct specific experiments with workloads for disk I/O evaluation, just for network I/O.

Che et al. (CHE et al., 2010) measured and analyzed the performance of three open source virtualization solutions: OpenVZ, Xen and KVM, which adopt the container-based virtualization, paravirtualization and full-virtualization, respectively. Measures regarding macro-performance and micro-performance were realized. Macro-performance considers the overall performance of a subsystem or whole system, whereas micro-performance mainly means the fine performance of some specific operations or instructions. All experiments were conducted on a Dell OPTIPLEX 755, with a 2.33GHz E6550 Intel Core2 DUO processor, 2GB DDRII 667 RAM, and 250GB 7200 RPM SATA II disk. They adopted the Gentoo Linux 2008.0 AMD64 with kernel 2.6.18 as both host operating system and guest. To investigate the macro-performance of disk virtualization, they ran IOzone to read and write operations of 2MB files. The results were expressed as bandwidth, in Mbps. OpenVZ held a partial read and write bandwidth compared with native Gentoo. Xen followed OpenVZ with a slight degradation in most experiments, while KVM had apparently lower performance than OpenVZ and Xen. No information was provided about the duration of the experiment, as well as any information about failures or system downtimes.

Padala et al. (PADALA et al., 2007) consolidated multi-tiered systems in several nodes using hypervisor based virtualization (Xen) and container-based virtualization (OpenVZ). Furthermore, both technologies were compared with a non-virtualized base system, in terms of application performance and low-level system metrics (like cache misses). An HP Proliant DL385 G1 was used for servers and client machines. Every server has two 2.6 GHz processors, each with 1MB of L2 cache, 8 GB of RAM, and two Gigabit network interfaces. No information about the hard disk technology was provided. The performance hardware events were collected through oprofile. Their experiments were conducted during a 15 minute period. The results depict that the throughput presented a similar behavior in the virtualized (hypervisor and container based) and non-virtualized environments, whereas for response times, the virtualized environment based on a hypervisor solution (Xen) presented an increase of over 600%, from 18ms to 130ms, when the number of threads was increased from 500 to 800. For the analysis of hardware counters in the OpenVZ-based system, each of the counter values was assigned for the whole system, including the shared kernel and all the virtual containers. This work differs on this point: it was able to isolate the hardware event counter for the application being provided. The counter values for CPU_CLOCK_UNHALTED and L2_CACHE_MISSES in the OpenVZ environment were less than twice the corresponding values for the base case. Compared to our work, the time intervals applied for the experiments were small and no considerations regarding availability were performed.

In (SOLTESZ et al., 2007), the authors discuss the features of Container-based Operating System virtualization, performing a comparison of the isolation and efficiency among hypervisor-based virtualization solution (Xen), a COS solution (Linux V-Server) and non-virtualized native Linux. For this purpose, a series of experiments was executed on an HP Proliant DL360 G4p,

assembled with 4GB RAM and two 160GB 7.2k RPM SATA disks, formatted on an ext3 file system with the journal recovery mechanism enabled. In the proposed experiments, each guest operating system was assigned to partitions of 2GB. For the DBench macro-benchmark that is disk I/O intensive and represents the throughput experienced by a single client performing around 90,000 file system operations, the Linux V-Server uniprocessor slightly exceeds the Linux native performance. With partitions of 2GB of size and journal recovery mechanism enabled, the unresponsiveness issue may have been hidden. Moreover, no considerations regarding availability were made.

Pu et. al (PU et al., 2010) presented their experimental study on the performance interference in parallel processing of CPU and network intensive workloads in a Xen VMM. The experiments were performed in IBM ThinkCentre A52 Workstations with two 3.2GHz Intel Pentium 4 CPUs (both have 16KB L1 caches and 2MB L2 caches), 2 GB 400MHz DDR RAM, a 250 GB 7200 RPM SATA2 disk, and an Intel e100 PRO/100 network interface. Their work is exclusively on network I/O bound workloads: no disk reading was involved. Additionally, even stating that multiplexing/demultiplexing of bridge and I/O channel may incur memory page management interferences, such as packets lost for high latency, fragmentations and increased data coping overhead, there were no narratives about such known failures sources. The workload generator tool used, httperf, is able to report errors on its output results. It was not elucidated if failures were not tracked or did not emerge. Neither of the assumptions can be refuted.

5.2 Experiments Automation

In the context of automatic monitoring, Ganglia (MASSIE; CHUN; CULLER, 2004) has a large set of qualities. It is a distributed monitoring system for high performance computing systems and is based on a multicast listen/announce protocol. The most influential consideration shaping Ganglia's design is scale (MASSIE et al., 2012). It provides traditional parameters, such as CPU, memory, disk and network. Nevertheless, Ganglia does not detect software hangs of applications that are being provided in the monitored nodes. It focuses on nodes and network failures. Moreover, it is composed of a series of configuration files and daemons, resulting in a sharp learning curve.

For automatic configuration of experiments, CloudBench (SILVA et al., 2013) is an existing alternative. It enables the automation of cloud evaluation through the running of controlled experiments. The applications used in the experiments are predefined benchmarks. CloudBench developers argue that, to port a new application, it typically requires the writing of approximately 150 lines of Bash script. The monitoring of resources is carried out with Ganglia. In our testbed, managing an experiment through CloudBench will make it a competitor, once our environment is sensitive to overloads. Moreover, it would be necessary to port Web Polygraph as a new application, since it is not an available benchmark tool. So, even presenting a sophisticated and extensible framework, the time cost to integrate CloudBench, as well as the risks associate

with its processing requirements, were judged as prohibitive.

5.3 Discussion

From the features and results of the aforementioned related works, Table 5.1 presents a summary of most relevant points aiming at comparing them. The goal is to highlight the strengths and limitations of this work compared to the others that have been published.

A remarkable point of this work was to monitor the availability of the provided service during the execution of experiments. As can be observed in the second column in Table 5.1, this feature was not contemplated by any other related work.

Regarding the use of hardware events counting, (PADALA et al., 2007) and (YE et al., 2010) applied this technique aiming at justifying why virtualized scenarios presented bottlenecks and overheads in comparison with non-virtualized environments. However, they did not obtain favorable results in virtualized environments, whereas the other five works presented them, as can be noticed in the last column of the Table 5.1.

The work that most closely matches the goals of this dissertation is the one conducted by Padala et. al (PADALA et al., 2007), since they evaluated I/O bound workloads in OpenVZ container-based virtualized cluster while counting hardware events. Nevertheless, they did not perform any procedures of overhead reduction in the conducted experiments on virtualized environments. This fact is pointed to here as the main cause of difference in the presented results in comparison with those presented in this dissertation. A point in favor of the work of these authors, and which appears as a limitation of this dissertation, was to monitor the execution of experiments through profiling. Some preliminary experiments performed by us that applied profiling yielded more than 80GB of resulting data, when they were interrupted due to the lack of storage space. Moreover, as the use of measurements to evaluate performability in virtualization environments is a highly time consuming activity, it presents risks to meeting deadlines. The experiments performed by Padala et. al were 15 minutes long, in contrast to the dozens of hours performed by us. As counting hardware events requires a volume of storage that is compatible with our testbed capacity, it was adopted.

A singularity of this work can be noted from Table 5.1: it was unique in that it joined favorable performance metric results in a virtualized environment with the technique of hardware event counting as a mechanism to investigate favorable performance in the virtualized environments.

Table 5.1: Related Works

Work	**Monitoring availability during the performance evaluations?**	**COS Virtualization?**	**I/O bound workload?**	**Monitor hardware events?**	**Favorable performance metric in virtualized environment?**
(SOLTESZ et al., 2007)	No	Yes	Yes	No	Yes, for network throughput and for dbench file system benchmark
(CHE et al., 2010)	No	Yes	Yes	No	Yes, network TCP tx for 4k packages
(HUBER et al., 2010)	No	No	No	No	No
(PADALA et al., 2007)	No	Yes	Yes	Yes, through counting and profiling	No
(DUAN; GU, 2010)	No	N/A	Yes	No	Yes, response times
(XAVIER et al., 2013)	No	Yes	No	No	Yes, disk write operation
(YE et al., 2010)	No	No	No	Yes, through counting	No
(PU et al., 2010)	No	No	Yes, network exclusively	No	Yes
This dissertation	Yes	Yes	Yes	Yes, through counting	Yes, hit ratio, response times, throughput and Byte Hit Ratio

6

Conclusions

Is it possible to execute a service with similar or even better performance, in a virtualized environment in comparison with a non-virtualized one? That was the central point of this conduced research. During the survey of state of the art studies, it was found that most of the related research was focused on CPU-bound applications analysis. Moreover, the performance evaluations based on collected measurements have highlighted the overheads of virtualized infrastructures, but without regarding their implications on availability.

Conducting performance evaluations of virtualized servers without regarding availability can lead to results that hide failures and, thereafter, compromise the results of evaluations concerned. Furthermore, the current work focuses on an I/O bound application, aiming at following an alternative path to most studies.

Through an extensive collection of experiments, the applied workload generation tool was able to identify occurrences of failures directly. Such failures were characterized as software hangs due to the unresponsiveness phenomenon and to tackle them was the main challenge faced. So, before analyzing performance, it was necessary to take care of system availability, featuring the research as a performability study. During all executed experiments, a goal of five 9's of availability was stated as a condition to proceed with the performance evaluation.

After overcoming the issue of availability below five 9's, the screening fractional factorial experiment proved itself to be a successful technique to tackle the issue of discovering favorable performance metrics in virtualized server clusters, in comparison with non-virtualized ones. After applying it, two of three performance evaluated metrics behaved favorably for a small storage capacity web cache server cluster based on container-based virtualization: hit ratio and response time.

The larger the storage space of the web cache server clusters became, greater was the unresponsiveness issue. For all the conducted experiments, one common factor was most often presented in the results whose availability was below five 9's: journal consistency mechanism.

The journal consistency mechanism of the ext3 and ext4 file systems was detected as the main source of overhead during the activity of tackling the issue of unresponsiveness. For the evaluated scenarios, when the journal mechanism was enabled, only environments with very

small cache storage capacity were able to provide high available I/O bound virtualized web cache server clusters, with a total storage capacity of 3GB. Any other tested virtualized scenarios were able to reach five 9's of availability with journal mechanism enabled, regardless of the applied tunings. So, the disabling of the consistency mechanism was shown to be a requirement of reaching five 9's of availability.

The discussed case studies presented recommendations on the using of I/O bound web cache server cluster solutions, both for non-virtualized and virtualized scenarios. The profiling of hardware events was successfully applied as a technique to provide evidence of possible reasons for a better performance in the virtualized clusters, even with the known intrinsic overhead. For full spool web cache servers, the virtualized cluster is recommended as the most stable solution, that is not impacted by the RAM stagnation issue due to the granularity of primary memory provided by the applied virtualization solution.

As a result of the work presented in this dissertation, the following contributions can be highlighted:

- the innovative approach in the performability analysis, that applied measurements to evaluate performance when failures were considered;
- the design and implementation of the TISVEP protocol, that applied a unique approach to establish communication with physical machines, virtual machines, containers, benchmarking machines, and storage machines during the execution of experiments;
- a methodology that leads to favorable performance metrics results on web cache server clusters provided in container-based virtualized environments, in a fair comparison with the non-virtualized ones.

In addition to the mentioned contribution, the following paper presenting findings of this dissertation was produced:

- Erico Guedes, Luis Silva and Paulo Maciel. Performability Analysis of I/O Bound Application on Container-based Server Virtualization Cluster. In: Proceedings of the IEEE 19th Symposium on Computers and Communications (IEEE-ISCC 2014). Madeira, Portugal.

6.1 Future Works

A series of further activities was identified as future works of the current research results.

The activity perceived as more challenging and with greater innovation is the development of a higher performance solution for the consistency mechanism of the ext3 and ext4 file systems. For virtualized environments, when the journal mechanism was enabled, the provided web cache server cluster was invariably affected by the unresponsiveness issue.

Disabling the consistency mechanism of the file systems prevents them to recovery in an automatic way and may lead to high downtimes: after some possible system failure, the time spent checking the consistency of a file system depends mainly on the number of files and directories that must be verified. Therefore, it also depends on the disk size. With terabytes of capacity, a single consistency check may take hours or even days. The involved downtime is unacceptable for any production environment of a high availability server.

Furthermore, to increase the number of I/O bound applications that could benefit from the use of container-based virtualization, it is of paramount importance to deploy solutions with consistency file system mechanism enabled to provide high availability for I/O bound services.

The evaluation of additional storage techniques would stretch the set of recommendations regarding the providing of I/O bound applications. The investigated underlying storage infrastructure was based on SATA Direct-Attached Storage (DAS). The adoption of different underlying storage technologies, such as SCSI, Solid State Drive (SSD), or even Storage Area Network (SAN), would yield benefits regarding the unresponsiveness issue? With such an outperform technologies, would be possible to provide the storage service with journal enabled? These are relevant questions that should be investigated aiming at tackling the overheads of virtualized infrastructures.

Additionally, according to 2014 ISOC Global Internet Report KENDE (2014), the proportion of fixed Internet traffic originating from video applications reached 50% in June 2012 and can reach 67% in 2017. Then, caching of Video on Demand (VoD) traffic should be better understood. All the proposed and applied methodology regarding web objects caching can be applied, and improved whether necessary, to investigated VoD caching.

Faced with these aforementioned challenges, a summary of the future works was performed below, ranked in order of priority:

1. to research solutions about file system consistency mechanisms that do not impact the responsiveness of I/O bound applications. As it is an activity with a high degree of sophistication, since it involves changing the operating system kernel on which the file system is used, it can be high time consuming.

2. to use alternative underlying storage technologies, such as SCSI, SSD, and Storage Area Network (SAN), and to state whether similar issues over the performability, as unresponsiveness, will be presented.

3. to evaluate the behavior of Video on Demand (VoD) workloads in cache systems. As they present greater file sizes, even more intense data flows, and represent the largest percentage of Internet traffic, larger storage devices will be required. With larger storage devices, new challenges regarding the management of resources will take place.

4. to increase the number of monitoring factors supported by the TISVEP protocol,

aiming at supporting the additional tuning elements that can lead to performance improvements in the virtualized clusters.

References

ADAMIC, L. A.; HUBERMAN, B. A. Zipf's Law and the Internet. **Glottometrics**, [S.l.], v.3, p.143–150, 2002.

AGUILAR, J.; LEISS, E. L. A coherence-replacement protocol for web proxy cache systems. **Int. J. Comput. Appl.**, Anaheim, CA, USA, v.28, n.1, p.12–18, Jan. 2006.

AHMED, W. **Mastering Proxmox**. [S.l.]: Packt Publishing, 2014.

ALMEIDA, V.; ALMEIDA, J. Internet Workloads: measurement, characterization, and modeling. **Internet Computing, IEEE**, [S.l.], v.15, n.2, p.15–18, March 2011.

APPARAO, P. et al. Characterization & Analysis of a Server Consolidation Benchmark. In: FOURTH ACM SIGPLAN/SIGOPS INTERNATIONAL CONFERENCE ON VIRTUAL EXECUTION ENVIRONMENTS, New York, NY, USA. **Proceedings...** ACM, 2008. p.21–30. (VEE '08).

APPARAO, P.; MAKINENI, S.; NEWELL, D. Characterization of network processing overheads in Xen. In: VIRTUALIZATION TECHNOLOGY IN DISTRIBUTED COMPUTING, 2006. VTDC 2006. FIRST INTERNATIONAL WORKSHOP ON. **Anais...** [S.l.: s.n.], 2006. p.2–2.

ARLITT, M. F.; WILLIAMSON, C. L. Web Server Workload Characterization: the search for invariants. **SIGMETRICS Perform. Eval. Rev.**, New York, NY, USA, v.24, n.1, p.126–137, May 1996.

ARLITT M.F.; WILLIAMSON, C. Internet Web servers: workload characterization and performance implications. **IEEE/ACM Transactions on Networking**, [S.l.], v.5, p.631–645, 1997.

AVIZIENIS, A.; LAPRIE, J.-c.; RANDELL, B. **Fundamental Concepts of Dependability**. 2001.

BAKIRAS, S.; LOUKOPOULOS, T.; AHMAD, I. Dynamic organization schemes for cooperative proxy caching. In: PARALLEL AND DISTRIBUTED PROCESSING SYMPOSIUM, 2003. PROCEEDINGS. INTERNATIONAL. **Anais...** [S.l.: s.n.], 2003. p.8 pp.–.

BAUER, E.; ADAMS, R. **Reliability and Availability of Cloud Computing**. [S.l.]: Wiley-IEEE Press, 2012.

BEERMANN, C.; POPHAL, M. **Calamaris Home Page**. URL: http://cord.de/calamaris-homepage.

BERNIER, R. **Network Your Shell Scripts with Netpipes**. URL: http://www.onlamp.com/pub/a/onlamp/2004/05/27/netpipes.html.

BLOOM, B. H. Space/Time Trade-offs in Hash Coding with Allowable Errors. **Commun. ACM**, New York, NY, USA, v.13, n.7, p.422–426, July 1970.

BRESLAU, L. et al. Web caching and Zipf-like distributions: evidence and implications. In: INFOCOM '99. EIGHTEENTH ANNUAL JOINT CONFERENCE OF THE IEEE COMPUTER AND COMMUNICATIONS SOCIETIES. PROCEEDINGS. IEEE. **Anais...** [S.l.: s.n.], 1999. v.1, p.126–134 vol.1.

CACHE.ORG squid. **Squid configuration directives**. URL: http://www.squid-cache.org/Doc/config/.

CHE, J. et al. A Synthetical Performance Evaluation of OpenVZ, Xen and KVM. In: SERVICES COMPUTING CONFERENCE (APSCC), 2010 IEEE ASIA-PACIFIC. **Anais...** [S.l.: s.n.], 2010. p.587–594.

CHLEBUS, E.; OHRI, R. Estimating parameters of the Pareto distribution by means of Zipf's law: application to internet research. In: GLOBAL TELECOMMUNICATIONS CONFERENCE, 2005. GLOBECOM '05. IEEE. **Anais...** [S.l.: s.n.], 2005. v.2, p.5 pp.–.

COOPER, I.; DILLER, J. **Known HTTP Proxy/Caching Problems**. URL: https://www.ietf.org/rfc/rfc3143.txt.

COOPER, I.; DILLEY, J. **RFC 3143 - Known HTTP Proxy/Caching Problems**. URL: http://tools.ietf.org/html/rfc3143.

DELGADO, J. et al. Paravirtualization for Scientific Computing: performance analysis and prediction. In: HIGH PERFORMANCE COMPUTING AND COMMUNICATIONS (HPCC), 2011 IEEE 13TH INTERNATIONAL CONFERENCE ON. **Anais...** [S.l.: s.n.], 2011. p.536–543.

DEVICES, A. A. M. **BIOS and Kernel Developer's Guide (BKDG) For AMD Family 10h Processors**. [S.l.]: AMD Advanded Micro Devices, 2010.

DILLEY, J.; ARLITT, M.; PERRET, S. **Enhancement and Validation of Squid's Cache Replacement Policy**. [S.l.]: Internet Systems and Applications Laboratory HP Laboratories Palo Alto, 1999.

DOUGLAS C. MONTGOMERY, G. C. R. **Applied Statistics and Probability for Engineers**. 5th.ed. [S.l.]: John Wiley and Sons, 2011. 784p.

DU, J.; SEHRAWAT, N.; ZWAENEPOEL, W. Performance Profiling of Virtual Machines. In: ACM SIGPLAN/SIGOPS INTERNATIONAL CONFERENCE ON VIRTUAL EXECUTION ENVIRONMENTS, 7., New York, NY, USA. **Proceedings...** ACM, 2011. p.3–14. (VEE '11).

DUAN, Z.; GU, Z. EWPCC: an elastic web proxy cache cluster basing on cloud computing. In: COMPUTER SCIENCE AND INFORMATION TECHNOLOGY (ICCSIT), 2010 3RD IEEE INTERNATIONAL CONFERENCE ON. **Anais...** [S.l.: s.n.], 2010. v.1, p.85–88.

DYKES, S.; JEFFERY, C.; DAS, S. Taxonomy and design analysis for distributed Web caching. In: SYSTEMS SCIENCES, 1999. HICSS-32. PROCEEDINGS OF THE 32ND ANNUAL HAWAII INTERNATIONAL CONFERENCE ON. **Anais...** [S.l.: s.n.], 1999. v.Track8, p.10 pp.–.

FENLASON, J. et al. **sed - a stream editor**. URL: https://www.gnu.org/software/sed/manual/sed.html.

FOONG, A.; FUNG, J.; NEWELL, D. An in-depth analysis of the impact of processor affinity on network performance. In: NETWORKS, 2004. (ICON 2004). PROCEEDINGS. 12TH IEEE INTERNATIONAL CONFERENCE ON. **Anais...** [S.l.: s.n.], 2004. v.1, p.244–250 vol.1.

FORSMAN, R. **Netpipes software package**. URL: http://freecode.com/projects/netpipes.

FUNDATION, A. S. **Apache HTTP server benchmarking tool**. URL: http://httpd.apache.org/docs/2.2/programs/ab.html.

FUNDATION, T. R. **The R Project for Statistical Computing**. URL: http://www.r-project.org/.

G., V. A. . P. A study on Web Caching Architectures and Performance. **Multi-Conference on Systemics, Cybernetics and Informatics**, [S.l.], v.1, p.1–5, 2001.

GOLDBERG, R. P. **Architectural Principles for Virtual Computer Systems**. 1973. Tese (Doutorado em Ciência da Computação) — Harvard University - Cambridge, MA US.

GOLDEN, B. **Virtualization for Dummies**. New York, NY, USA: John Wiley & Sons, Inc., 2007.

GONZALEZ-CANETE, F.; CASILARI, E.; TRIVINO-CABRERA, A. Two new metrics to evaluate the performance of a Web cache with admission control. In: ELECTROTECHNICAL CONFERENCE, 2006. MELECON 2006. IEEE MEDITERRANEAN. **Anais...** [S.l.: s.n.], 2006. p.696–699.

GORMAN, M. **Understanding the Linux Virtual Memory Manager**. Upper Saddle River, NJ, USA: Prentice Hall PTR, 2004.

GREGG, B. **Systems Performance**: enterprise and the cloud. 1st.ed. Upper Saddle River, NJ, USA: Prentice Hall Press, 2013.

HAGEN, W. von. **Professional Xen Virtualization**. [S.l.]: Wrox Press Ltd., 2008.

HAMILTON, M.; ROUSSKOV, A.; WESSELS, D. **Cache Digest specification - version 5**. URL: http://www.squid-cache.org/CacheDigest/cache-digest-v5.txt.

HINTEMANN, R.; FICHTER, K.; SCHLITT, D. Adaptive computing and server virtualization in German data centers - Potentials for increasing energy efficiency today and in 2020. In: BIS-VERLAG. **Anais...** BIS-Verlag, 2014. p.477–484. ISBN 978-3-8142-2317-9.

HUBER, N. et al. Analysis of the Performance-Influencing Factors of Virtualization Platforms. In: MEERSMAN, R.; DILLON, T.; HERRERO, P. (Ed.). **On the Move to Meaningful Internet Systems, OTM 2010**. [S.l.]: Springer Berlin Heidelberg, 2010. p.811–828. (Lecture Notes in Computer Science, v.6427).

IAKOBASHVILI, R.; MOSER, M. **curl-loader**. URL: http://curl-loader.sourceforge.net.

JAIN, R. **The art of computer systems performance analysis - techniques for experimental design, measurement, simulation, and modeling.** [S.l.]: Wiley, 1991. I-XXVII, 1-685p. (Wiley professional computing).

JEFFRIES, A. **Features/CacheManager - Squid Web Proxy Wiki**. URL: http://wiki.squid-cache.org/Features/CacheManager.

KANDALINTSEV, A. et al. Profiling cloud applications with hardware performance counters. In: INFORMATION NETWORKING (ICOIN), 2014 INTERNATIONAL CONFERENCE ON. **Anais...** [S.l.: s.n.], 2014. p.52–57.

KENDE, M. **Internet Society Global Internet Report 2014**. [S.l.]: ISOC, 2014.

KIM, D. S.; MACHIDA, F.; TRIVEDI, K. S. Availability Modeling and Analysis of a Virtualized System. **2009 15th IEEE Pacific Rim International Symposium on Dependable Computing**, [S.l.], p.365–371, Nov. 2009.

KOLYSHKIN, K. **Virtualization in Linux**. URL: http://download.openvz.org/doc/openvz-intro.pdf.

KOLYSHKIN, K. **UBC systemwide configuration**. URL: http://wiki.openvz.org/UBC_systemwide_configuration.

KOLYSHKIN, K. **User Guide - OpenVZ Linux ContainersGuide**. URL: http://openvz.org/User_Guide/Print_version.

KOLYSHKIN, K. **Bind mounts - OpenVZ Linux Containers Wiki**. URL: http://openvz.org/Bind_mounts.

KUJAU, C. **ext4 Frequently Asked Questions**. URL: https://ext4.wiki.kernel.org/index.php/Frequently_Asked_Questions.

KVM. **KVM**. URL: http://www.linux-kvm.org/page/Main_Page.

LE, D.; HUANG, H.; WANG, H. Understanding Performance Implications of Nested File Systems in a Virtualized Environment. In: USENIX CONFERENCE ON FILE AND STORAGE TECHNOLOGIES, 10., Berkeley, CA, USA. **Proceedings...** USENIX Association, 2012. p.8–8. (FAST'12).

LILJA, D. J. **Measuring Computer Performance**: a practitioner's guide. [S.l.]: Cambridge University Press, 2005.

LXC. **LXC - Linux Containers**. URL: https://linuxcontainers.org/.

MASSIE, M. et al. **Monitoring with Ganglia**. 1st.ed. [S.l.]: O'Reilly Media, Inc., 2012.

MASSIE, M. L.; CHUN, B. N.; CULLER, D. E. The ganglia distributed monitoring system: design, implementation, and experience. **Parallel Computing**, [S.l.], v.30, n.7, p.817 – 840, 2004.

MATOS, R. et al. Sensitivity Analysis of Server Virtualized System Availability. **Reliability, IEEE Transactions on**, [S.l.], v.61, n.4, p.994–1006, Dec 2012.

MCDOUGALL, R.; ANDERSON, J. Virtualization Performance: perspectives and challenges ahead. **SIGOPS Oper. Syst. Rev.**, New York, NY, USA, v.44, n.4, p.40–56, Dec. 2010.

MEDINA, V.; GARCÍA, J. M. A Survey of Migration Mechanisms of Virtual Machines. **ACM Comput. Surv.**, New York, NY, USA, v.46, n.3, p.30:1–30:33, Jan. 2014.

MELL, P. M.; GRANCE, T. **SP 800-145. The NIST Definition of Cloud Computing**. Gaithersburg, MD, United States: NIST, 2011.

MEYER, J. F. On Evaluating the Performability of Degradable Computing Systems. **IEEE Trans. Comput.**, Washington, DC, USA, v.29, n.8, p.720–731, Aug. 1980.

MORREALE, P. W. **Documentation for /proc/sys/vm/***. URL: https://www.kernel.org/doc/Documentation/sysctl/vm.txt.

MOUSA, H. et al. VrtProf: vertical profiling for system virtualization. In: SYSTEM SCIENCES (HICSS), 2010 43RD HAWAII INTERNATIONAL CONFERENCE ON. **Anais...** [S.l.: s.n.], 2010. p.1–10.

OPENNEBULA. **OpenNebula**. URL: http://opennebula.org/.

OPENVZ. **OpenVZ Linux Containers**. URL: http://openvz.org.

ORACLE. **Oracle Virtual Box**. URL: https://www.virtualbox.org.

ORGERIE, A.-C.; ASSUNCAO, M. D. d.; LEFEVRE, L. A Survey on Techniques for Improving the Energy Efficiency of Large-scale Distributed Systems. **ACM Comput. Surv.**, New York, NY, USA, v.46, n.4, p.47:1–47:31, Mar. 2014.

OUYANG, J.; LANGE, J. R. Preemptable Ticket Spinlocks: improving consolidated performance in the cloud. **SIGPLAN Not.**, New York, NY, USA, v.48, n.7, p.191–200, Mar. 2013.

PADALA, P. et al. **Performance evaluation of virtualization technologies for server consolidation**. [S.l.]: HP Laboratories, 2007.

PADALA, P. et al. Adaptive control of virtualized resources in utility computing environments. **SIGOPS Oper. Syst. Rev.**, New York, NY, USA, v.41, n.3, p.289–302, Mar. 2007.

PITKOW, J. E. Summary of WWW Characterizations. **World Wide Web**, [S.l.], v.2, 1998.

POLYGRAPH, W. **Web Polygraph**: a benchmarking tool for caching proxies and other web intermediaries. URL: http://www.web-polygraph.org/.

POTZL, H. et al. **Linux-VServer**. URL: http://linux-vserver.org/.

PROXMOX. **Proxmox**. URL: https://www.proxmox.com/.

PU, X. et al. Understanding Performance Interference of I/O Workload in Virtualized Cloud Environments. In: CLOUD COMPUTING (CLOUD), 2010 IEEE 3RD INTERNATIONAL CONFERENCE ON. **Anais...** [S.l.: s.n.], 2010. p.51–58.

QEMU. **Qemu - Open Source Processor Emulator**. URL: http://www.qemu.org.

REGOLA, N.; DUCOM, J.-C. Recommendations for Virtualization Technologies in High Performance Computing. In: CLOUD COMPUTING TECHNOLOGY AND SCIENCE (CLOUDCOM), 2010 IEEE SECOND INTERNATIONAL CONFERENCE ON. **Anais...** [S.l.: s.n.], 2010. p.409–416.

RELIASOFT. **Reliability Basics**: availability and the different ways to calculate it. URL: http://www.weibull.com/hotwire/issue79/relbasics79.htm.

RUBINO, G.; SERICOLA, B. Interval availability distribution computation. In: FAULT-TOLERANT COMPUTING, 1993. FTCS-23. DIGEST OF PAPERS., THE TWENTY-THIRD INTERNATIONAL SYMPOSIUM ON. **Anais...** [S.l.: s.n.], 1993.

SHAN, Z.; CHIUEH, T.-c.; WANG, X. Virtualizing system and ordinary services in Windows-based OS-level virtual machines. In: ACM SYMPOSIUM ON APPLIED COMPUTING, 2011., New York, NY, USA. **Proceedings...** ACM, 2011. p.579–583. (SAC '11).

SHAO, L. et al. User-Perceived Service Availability: a metric and an estimation approach. In: WEB SERVICES, 2009. ICWS 2009. IEEE INTERNATIONAL CONFERENCE ON. **Anais...** [S.l.: s.n.], 2009. p.647–654.

SILVA, M. et al. CloudBench: experiment automation for cloud environments. In: CLOUD ENGINEERING (IC2E), 2013 IEEE INTERNATIONAL CONFERENCE ON. **Anais...** [S.l.: s.n.], 2013. p.302–311.

SMITH, J.; NAIR, R. The architecture of virtual machines. **Computer**, [S.l.], v.38, n.5, p.32–38, May 2005.

SOLTESZ, S. et al. Container-based Operating System Virtualization: a scalable, high-performance alternative to hypervisors. **SIGOPS Oper. Syst. Rev.**, New York, NY, USA, v.41, n.3, p.275–287, Mar. 2007.

SONG, X.; CHEN, H.; ZANG, B. Why software hangs and what can be done with it. In: DEPENDABLE SYSTEMS AND NETWORKS (DSN), 2010 IEEE/IFIP INTERNATIONAL CONFERENCE ON. **Anais...** [S.l.: s.n.], 2010. p.311–316.

SOUZA, D. et al. EucaBomber: experimental evaluation of availability in eucalyptus private clouds. In: SYSTEMS, MAN, AND CYBERNETICS (SMC), 2013 IEEE INTERNATIONAL CONFERENCE ON. **Anais...** [S.l.: s.n.], 2013. p.4080–4085.

STAPELBERG, R. F. **Handbook of reliability, availability, maintainability and safety in engineering design**. [S.l.]: Springer, 2009.

TEETOR, P. **R Cookbook**. 1st.ed. [S.l.]: O'Reilly Media, Inc., 2011.

TOPE, I. et al. Performance Evaluation of Oracle VM Server Virtualization Software 64 Bit Linux Environment. In: SECURITY MEASUREMENTS AND METRICS (METRISEC), 2011 THIRD INTERNATIONAL WORKSHOP ON. **Anais...** [S.l.: s.n.], 2011. p.51–57.

TOTTY, B. et al. **Http**: the definitive guide. Sebastopol, CA, USA: O'Reilly & Associates, Inc., 2002.

VALLOPPILLIL, V.; ROSS, K. **Cache Array Routing Protocol v1.0**. URL: https://tools.ietf.org/html/draft-vinod-carp-v1-03.

VARGAS, E.; BIANCO, J.; DEETHS, D. **Sun Cluster environment**. [S.l.]: Sun Microsystems Press, 2001.

VIXIE, P.; WESSLES, D. **Hyper Text Caching Protocol (HTCP/0.0)**. URL: https://tools.ietf.org/html/rfc2756.

VMWARE. **VMware Virtualization Software for Desktops, Servers and Virtual Machines for Public and Private Cloud Solutions**. URL: http://www.vmware.com.

WALDSPURGER, C.; ROSENBLUM, M. I/O Virtualization. **Commun. ACM**, New York, NY, USA, v.55, n.1, p.66–73, Jan. 2012.

WANG, X. et al. Hang Analysis: fighting responsiveness bugs. **SIGOPS Oper. Syst. Rev.**, New York, NY, USA, v.42, n.4, p.177–190, Apr. 2008.

WESSELS, D. **Web Caching**. Sebastopol, CA, USA: O'Reilly & Associates, Inc., 2001.

WESSELS, D. **Squid**: the definitive guide. [S.l.]: O'Reilly, 2004.

WESSELS, D.; CLAFFY, K. **Internet Cache Protocol (ICP)**. URL: https://www.ietf.org/rfc/rfc2186.txt.

WESSELS, D.; CLAFFY, K. **Application of Internet Cache Protocol (ICP)**. URL: https://www.ietf.org/rfc/rfc2187.txt.

XAVIER, M. et al. Performance Evaluation of Container-Based Virtualization for High Performance Computing Environments. In: PARALLEL, DISTRIBUTED AND NETWORK-BASED PROCESSING (PDP), 2013 21ST EUROMICRO INTERNATIONAL CONFERENCE ON. **Anais...** [S.l.: s.n.], 2013. p.233–240.

XEN. **Xen Hypervisor**. URL: http://xen.org.

YE, K. et al. Analyzing and Modeling the Performance in Xen-Based Virtual Cluster Environment. In: HIGH PERFORMANCE COMPUTING AND COMMUNICATIONS (HPCC), 2010 12TH IEEE INTERNATIONAL CONFERENCE ON. **Anais...** [S.l.: s.n.], 2010. p.273–280.

Appendix

A

Codes

A.1 Web-polygraph Pareto Distribution Implementation

```
// File polygraph-4.3.1/src/base/opts.cc
if (type == "pareto") {
   if (splitVal(params, s[0], s[1], ',') &&
   isRTX(s[0], p[0]) && isRTX(s[1], p[1]))
   distr = new ParetoDistr(gen, p[0], p[1]);
} else

// File polygraph-4.3.1/src/xstd/rndDistrs.h
class ParetoDistr: public RndDistr
{
 public:
    ParetoDistr(RndGen *aGen, double aShape, double aScale):
       RndDistr(aGen), theShape(aShape), theScale(aScale) {}
    virtual const char *pdfName() const { return "pareto"; }
    virtual double mean() const { return -1; }
    virtual double sdev() const { return 0; }
    virtual double shape() const { return theShape; }
    virtual double scale() const { return theScale; }
    virtual double trial();
    virtual ostream &print(ostream &os, ArgPrinter p =
       &RndDistr_DefArgPrinter) const;

 protected:
    double theShape;
    double theScale;
};
```

```
// File polygraph-4.3.1/src/xstd/rndDistrs.cc
double ParetoDistr::trial() {
  return theScale * pow(1 - theGen->trial(), (-1/theShape));
}

ostream &ParetoDistr::print(ostream &os, ArgPrinter p) const {
  os <pdfName() <'(';
  p(os, theShape, 0);
  p(os <", ", theScale, 1);
  return os <')';
}

// File polygraph-4.3.1/src/pgl/PglSemx.cc

if (cname == "pareto") {
  checkArgs(cname, 2, args);
  return new DistrSym(dType, new ParetoDistr(gen, dargs[0],
     dargs[1]));
} else
```

A.2 TISVEP config.properties file instance

PHYSICAL MACHINES SECTION

```
PMs=(192.168.15.23 192.168.15.19 192.168.15.18)
HD_LABELS="/dev/sdb /dev/sdc /dev/sdd"
PARTITIONS_NUMBER=1
PARTITION_SIZE=+2G
FS_CONSISTENCY=0
FS_TYPE="ext4"
NO_CONSISTENCY_MOUNT_OPTIONS="defaults,noatime,nodiratime,barrier=0"
SPOOL_LABEL="/spool"
```

SERVICE SECTION

```
SQUID_CONF_FILE="/usr/local/squid/etc/squid.conf"
SQUID_BIN="/usr/local/squid/sbin/squid"
CACHE_LOG_DIR="/usr/local/squid/var/logs"
ENABLE_CACHE_LOG=1
DEFAULT_PORT=3128
CACHE_DIR_PREFIX="/var"
```

```
CACHE_DIR_SUFFIX="/squid3"
CACHE_SIZE=1024
L1_CACHE_LEVEL=16
L2_CACHE_LEVEL=256
MI_AMOUNT=3
```

BENCHMARKING TOOL SECTION

```
WP_SERVER_IP=192.168.15.10
WP_CLIENT_IP=192.168.15.25
EXPERIMENT_DURATION=600sec
WP_CONFIG_FILE=/opt/scripts/WebWorkload.pg
```

VIRTUAL SERVERS SECTION

```
declare -A PMsCT_IDs=(
 ["${PMs[0]}"]="/spool1_100+/spool2_101+/spool3_102"
 ["${PMs[1]}"]="/spool1_106+/spool2_107+/spool3_108"
 ["${PMs[2]}"]="/spool1_112+/spool2_113+/spool3_114"
)

declare -A VMsSPOOLs=(
 ["192.168.15.113"]="/spool1_118"
 ["192.168.15.114"]="/spool2_119"
 ["192.168.15.115"]="/spool3_120"
 ["192.168.15.119"]="/spool1_124"
 ["192.168.15.120"]="/spool2_125"
 ["192.168.15.121"]="/spool3_126"
 ["192.168.15.125"]="/spool1_130"
 ["192.168.15.126"]="/spool2_131"
 ["192.168.15.127"]="/spool3_132"
)

declare -A PMsServerIPs=(
     ["${PMs[0]}"]="192.168.15.113 192.168.15.114 192.168.15.115"
     ["${PMs[1]}"]="192.168.15.119 192.168.15.120 192.168.15.121"
     ["${PMs[2]}"]="192.168.15.125 192.168.15.126 192.168.15.127"
)
```

MANAGEMENT SECTION

```
SIMULATION_TYPE="CTS"
MOUNTPOINT_NAMES="spool_squid"
```

```
NMS_IP="192.168.15.22"
CACHE_MONITORING_STEP=300
LOCAL_STORAGE="/experiments/"
REMOTE_STORAGE="/experiments/"
ITERATIONS=30
```

PROFILING SECTION

```
declare -A PROFILING_EXP=(
      ["CPU_CLK_UNHALTED"]=50000
      ["L2_CACHE_MISS"]=500
      ["L3_CACHE_MISSES"]=500
)
OPERF="/usr/local/bin/operf"
OCOUNT="/usr/local/bin/ocount"
PROFILE_COD=2
OCOUNT_INTERVAL_LENGTH=60000 # miliseconds
```

B

Specification of TISVEP Messages

TISVEP messages, that comprise the message code class, with their respective goals and textual descriptions of a message instance, are presented int the following Sections. As described below, each transmitted message has the MSG_ID parameter in the first field. Each message is easily identified by a bold Message Code and name.

B.1 Infrastructure Configuration Messages

1: hdsConfig: the processing of this message results in the formatting and configuration of the partitions and its respective file systems, and also mounts these partitions on the directories that will be used as Directed-Attached Storage spools by web cache servers.

Example:

```
MSG_ID=1:HD_LABELS="/dev/sdb/dev/sdc/dev/sdd":PARTITION_NU
MBER=2:PARTITION_SIZE=+115G:FS_CONSISTENCY=0:FS_TYPE=ext4:NO_CO
NSISTENCY_MOUNT_OPTIONS="defaults,noatime,nodiratime,barrier=0":
SPOOL_LABEL="/spool":ITERATION=2
```

In the above example, three hard disks (`/dev/sdb/dev/sdc/dev/sdd`) will be used per destination physical machine during the execution of the configured experiment. Each of them will contain two partitions of 115GB, configured without the journal consistency mechanism of the ext4 file system. Moreover, each of the partitions will be mounted in the destination directory with a `spool` prefix. There will be 2 replications of this configured experiment. The formatting routine will be performed only in the first replication. In the subsequent ones, the partition will only have its file system rebuilt. This will reduce the time spent between replications.

2: mountVirtualSpool: it is the message responsible for mounting partitions in ex-

periments whose web cache spools were provided by virtual servers. It uses a bind-mounted technique when container-based virtualization is applied, and external block partitions, for KVM virtual machines. Web cache objects are stored on these partitions.

Example:
`MSG_ID=2:SIMULATION_TYPE=CTS:SPOOL_VID_SET=/spool1_100+ /spool2_101`

In the above example, an experiment with container-based virtualization was conducted (CTS). It involves containers with numerical identifiers of 100 and 101. In the container 100, the spool directory used was `/spool1`, whereas the `/spool2` directory was used by container 101.

3: mountCacheLog: it is the message responsible for mounting web cache log partition. It is used to store Squid log files during the execution of experiments. It uses a bind-mounted technique when container-based virtualization is applied, and external block partitions for KVM virtual machines as well as for PMs.
Example:
`MSG_ID=3:CACHE_LOG_DIR=/usr/local/squid/var/logs:CT_IDS:100+101+ 102`

The example above exhibit a **mountCacheLog** message sent during an experiment that applies containers. The directory `/usr/local/squid/var/logs` was applied as the storage point of the Squid log files.

4: getMountedPartitions: it returns a list of mounted partitions, according to a transmitted directory key.

Example:
`4:spool`

In the above example, the keyword spool was used to search for mounted partitions at a remote destination. It returns the output of the command as it would be executed locally.

5: umountSpool: between the execution of experiments, the partitions must be unmounted before formatting, enabling this routine. A mounted partition cannot be formatted.

Example:
`5:/spool/squid1`

The example above will result in the unmounting of the partition referenced by the `/spool/`
`squid1` directory.

B.2 Service Configuration Messages

102: configureService: it is the message whose processing results in a tuned configuration file of the Squid web cache server. Through the use of the sed FENLASON et al. (2004) stream editor, it was possible to perform automatic modifications to the required parameters in the configuration file of each Squid web server that forms the cluster.

Example:
`MSG_ID=102:SIMULATION_TYPE=CTS:ACCESS_LOG_FILE=/usr/local/squid/var/logs/accessC1.log:CACHE_LOG_FILE=/usr/local/squid/var/logs/accessC1.log:CACHE_DIR_SPEC(CACHE_DIR=/spool/squid3,CACHE_SIZE=1024+L1_CACHE_LEVEL=16+L2_CACHE_LEVEL=256):PEERS_LIST=192.168.15.126_192.168.15.127_192.168.15.128:SQUID_CONF_FILE=/usr/local/squid/etc/squid.conf`

The above message example is sent during a container-based virtualization (`CTS`) experiment. The Squid access log file (`/usr/local/squid/var/logs/accessC1.log`), used to store the transactions performed by the web cache server (it is through the processing of this file that a Byte Hit Ratio value is obtained), as well as the cache log file (`/usr/local/squid/var/logs/cacheC1.log`), that stores the behavior of the server itself, form the list of required parameters aimed at executing an instance of a web cache server. Similarly, the directory where the web objects are `/spool/squid3`, the capacity of the cache storage, and the number of directories that form L1 (`16`) and L2 (`256`) cache levels of Squid server are sent as well. The peers list (`192.168.15.126_192.168.15.127_192.168.15.128`) contains the IP addresses of servers that form the cluster. Finally, the full pathname of the Squid configuration file (`/usr/local/squid/etc/squid.conf`) is transmitted to inform the exact location of the file that is tuned on the destination server.

103: startupService: it performs the required tunings on the web cache server configuration that are external to the configuration file and starts the servers.

Example: `MSG_ID=103:CACHE_DIR=/spool1/squid3:SQUID_CONF_FILE=/usr/local/squid/etc/squid.conf:SQUID_BIN=/usr/local/squid/sbin/squid`

It is necessary to perform three external configurations before the starting of the Squid process: (i) create the spool directories (`/spool1/squid3`); (ii) assign the correct permissions to these spool directories; and (iii) create the Squid L1 and L2 cache subdirectories inside the spool directories. All these routines are performed during the processing of message `103`, enabling the initialization of the Squid process: it is the last step of the processing of this message.

104: stopService: it is responsible for gracefully stopping the web cache server, aimed at avoiding data corruption of the saved log files.

Example: `MSG_ID=104:SQUID_BIN=/usr/local/squid/sbin/squid`

The processing of the above message, at its destinations, results in gathering all numerical identifications of Squid processes, subsequently halting them.

105: cleanUpServiceLog: the processing of this message cleans up the cache log files between experiments.

Example: `MSG_ID=105:CACHE_LOG_DIR:/usr/local/squid/var/logs`

Messages such as those in the above example are transmitted after the finalization of a replication of an experiment. It drives the destination to delete the log files in the `/usr/local/squid/var/logs` directory as soon as they are copied to the external storage. Moreover, it recreates empty instances of these files for the next experiment that will be conducted.

B.3 Benchmarking Configuration Messages

201: startWPServer: it aims at starting a Web Polygraph Server-side, with specified configuration parameters.

Example
`MSG_ID=201:EXPERIMENT_DURATION=30min:HTTP_PROXIES=192.168.15.113_3128+192.168.15.114_3128+192.168.15.115_3128:WP_CONFIG_FILE=/opt/scripts/WebWorkload.pg:STORAGE_PREFIX=CTS_2014-09-10_05h36m_I10`

The above message example results in the configuration of an experiment with 30 minutes of duration (`30min`). The Web Polygraph server-side process will answer to the three web cache servers (`192.168.15.113_3128+192.168.15.114_3128+192.168.15.115_3128`) when they do not have the requested objects. when they do not have the requested

objects. Besides, the full pathname configuration file of the Web Polygraph is sent as well. Similarly to the Squid configuration file, the `sed` stream editor is used to perform all necessary modifications in the Web Polygraph configuration file (`/opt/scripts/WebWorkload.pg`), such as to adjust the duration time of the experiment. Web Polygraph log files, that are used to obtain the numerical values of the hit ratio, response time, and throughput, are stored in the `CTS_2014-09-10_05h36m_I10` directory. The last step produced by the processing of the `startWPServer` message is the launching of `polygraph-server` process, that is initialized in background, releasing start up TISVEP flow to the next message.

202: startWPClient: it aims at starting the Web Polygraph client-side, with specified configuration parameters. The processing of this message results in the initialization of the experiment.

Example:
`MSG_ID=202:EXPERIMENT_DURATION=60min:HTTP_PROXIES=192.168.15.113_3128+192.168.15.114_3128+192.168.15.115_3128:WP_CONFIG_FILE=/opt/scripts/WebWorkload.pg:STORAGE_PREFIX=CTS_2014-09-10_05h36m_I20`

Except for the value of `MSG_ID`, this message is similar to `startWPServer` message. Web Polygraph requires that both the server and client sides have an identical configuration file. However, the client-side of Web Polygraph is not started in the background. The experiment will stay in this state as long as the `polygraph-client` process remains alive. The Web Polygraph client-side process will send requests to the three web cache servers (`192.168.15.113_3128+192.168.15.114_3128+192.168.15.115_3128`), filling its spools.

203: stopWPServer: it is responsible for gracefully stopping the Web Polygraph server-side.

Example: `MSG_ID=203`

The processing of the above message by the Web Polygraph server-side machine identifies the PID of the `polygraph-server` process and kills it gracefully, with a `-TERM` signal. A process that is killed through a `-TERM` signal can catch it and perform natively implemented shutdown routines before exit.

B.4 Experiment Monitoring and Management Messages

301: startCacheMonitoring: it launches the process that monitors web cache server performance metrics used for performability analysis. This message is asynchronous: it is

executed in the background during the same time interval as the experiment, as defined in the `EXPERIMENT_DURATION` parameter.

Example: `MSG_ID=301:STORAGE_PREFIX=CTS_2014-09-10_05h36m_I20:SIMULATION_TYPE=CTS:EXPERIMENT_DURATION:1800:CACHE_MONITORING_STEP=300`

The above example of the message `startCacheMonitoring` will monitor a `CTS` experiment, storing its resulting data in the `CTS_2014-09-10_05h36m_I20` directory. Such an experiment will have `1800` seconds and at every `300` seconds a sample of the performance metrics will be captured.

302: exportBenchmarkingData: as soon as the experiment is over, several routines of data exporting are performed. `exportBenchmarkingData` is one of them. Log files from the Web Polygraph client side machine are exported to the Cloud NMS machine, where they are stored for further analysis.

Example: `MSG_ID=302:SOURCE_DIR=/opt/scripts/CTS_2014-07-23_12h00m/:DESTINATION_DIR=/experiments/CTS_2014-07-23_12h00m:NMS_IP=192.168.15.22`

The above `exportBenchmarkingData` message will export all files of the `/opt/scripts/CTS_2014-07-23_12h00m/` directory to the `/experiments/CTS_2014-07-23_12h00m/` directory that lies on Cloud NMS whose IP address is `192.168.15.22`.

303: updateFunctionalities: this message aims at updating the `autSimulation.sh` file to virtual destinations (containers or virtual machines), physical destinations, and Web Polygraph servers.Cloud NMS itself sends and receives this message, triggering the upload process of `autSimulation.sh`. The upload is performed through scp command, that uses SSH protocol to upload `autSimulation.sh` file.

304: restartFunctionalities: this message is responsible for restarting the socket that is bound to the `autSimulation.sh` script file after an update. So, the new version of TISVEP protocol will be automatically available, without manual interaction of the experimenter.

Example: `MSG\_ID=304`

305: exportCacheLogs: this message is responsible for exporting the log files of the

web cache servers. The Cloud NMS machine must receive the resulting cache log data files, for further analysis.

Example:
`MSG_ID=305:SOURCE_DIR=/opt/scripts/CTS_2014-07-23_12h00m/:DESTINATION_DIR=/experiments/CTS_2014-07-23_12h00m:NMS_IP=192.168.15.22`

The above `exportCacheLogs` produces a similar result to the `exportBenchmarkingData` message.

306: startProfiling: to perform profiling of hardware events, `startProfiling` messages are sent to Physical Machines. The processing of `startProfiling` messages results in the initialization of profiling over web cache server processes.

Example:
`PROFILE_COD=2:SIMULATION_TYPE=CTS:EXPRESSIONS=CPU_CLK_UNHALTED~L2_CACHE_MISS:SESSION_DIR=/experiments/CTS_2014-07-23_12h00m/192.168.15.23/:EXTENSION=60000`

The above `startProfiling` message of code `2` will launch an `ocount` process that will report, in time intervals of 60000ms (per minute), how many instances of CPU_CLK_UNHALTED and L2_CACHE_MISSES hardware events were accounted. The results will be stored in the `/experiments/CTS_2014-07-23_12h00m/192.168.15.23/` directory.

307: stopProfiling: this message is responsible for gracefully halting the process of the `oprofile`.

Example: `MSG\_ID=307:PROFILE\_COD=2`

The receiver PMs will kill the `oprofile` process gracefully, through `-SIGINT` signal, as recommended in the oprofile manual.

308: exportProfileData: this message is responsible for exporting the files containing profiling results. The Cloud NMS machine must receive such files and store them for further analysis.

Example:
`MSG_ID=308:SOURCE_DIR=/opt/scripts/CTS_2014-07-23_12h00m/:DESTI`

NATION_DIR=/experiments/CTS_2014-07-23_12h00m:NMS_IP=192.168.15.22

The above exportProfileData produces a similar result to exportBenchmarkingData and exportCacheLogs messages.

309: waitServiceStartup: this message is responsible for synchronizing the profiler process with the Squid processes. The initialization of Squid processes are non blocking: as soon as the message is received, the TISVEP start up flow goes ahead. As can be noted in Figure 3.2, after startupService, waitServiceStartup is sent and its corresponding functionality is performed. It will prevent the non-profiling of any Squid process that has not completed its initialization.

Example: MSG_ID=309:SERVERS_AMOUNT=3

In the above example of waitServiceStartup message, 3 web cache server processes must be executed on each physical machine. The receiver PM will monitor the number of initialized servers and will return as soon as the 3 server processes are running.

B.5 TISVEP Operation

Aiming at detailing the netpipes software package, the first used TISVEP message, hdsConfig, will be used as an example. The faucet server end command is executed during bootstrap, both in the physical machine and in the virtual machine destinations, as described below:

```
faucet 9997 --in --out bash -c "/opt/scripts/autSimulation.sh; cat /opt/scripts/status.dat"&
```

The above command creates a two-way data flow. The traditional piping inter-process communication mechanism feeds the output of one process as input to the other, in a simplex way. So, it lacks the ability to run bidirectional communications. Using --in and --out switches, a two-way network data flow becomes possible. Specifically, each parameter of the faucet server end command has the following objectives:

- 9997: TCP listen port;
- --in: assigned functionality to input data flow;
- --out assigned functionality to output data flow;
- bash: default shell script language used to execute the following commands;

- -c: specifies a list of commands that will be assigned to each data flow;
- `/opt/scripts/autSimulation.sh`: shell script that will receive input data flow;
- `cat /opt/scripts/status.dat`: command that will be executed after the end of the shell script associated with received input data flow. The `/opt/scripts/status.dat` file content will be returned through output data flow;
- `&`: starts server end `/opt/scripts/autSimulation.sh` script file, that contains TISVEP message processing functionalities, in background.

The `hose` client end command is executed during message forward on NMS, as the following describes:

```
hose $PM 9997 --out --in bash -c "(echo $MESSAGE1;sockdown 1 1); cat » status.dat"
```

The above command will send the `$MESSAGE1` message to the `$PM` destination server. Each parameter of the `hose` client end command has a similar objective to the server end one. The command associated with `--out` output data flow is `echo $MESSAGE1;sockdown`. Netpipes permits the user to choose whether it's the client or server that sends or receives data upon connection by using the `--in` and `--out` switches in a specific order. For the two aforementioned and described commands, which define the `hdsConfig` message transmission and reception, the following data flow is performed:

1. NMS sends an `hdsConfig $MESSAGE1` message through the output data flow and closes the output half of the network pipe (`sockdown`);
2. the destination server gathers the input data flow, processing the `hdsConfig $MESSAGE1` message through `/opt/scripts/autSimulation.sh` script;
3. the destination server sends the standard output of the `cat /opt/scripts/status.dat` command through the output data flow;
4. NMS receives the output of the destination server through the input data flow associated with the command `cat » status.dat`.

Each destination server answers with a `STATUS` message. It contains the outputs of the executed commands from each destination. The `STATUS` messages transmit the outputs of the commands to the NMS administration terminal, enabling easy and centralized monitoring of the configuration procedures that were performed at the destinations.

```
root@nms:~/master# tailf status.dat | nl
     1 Command (m for help): Partition type:
     2    p   primary (1 primary, 0 extended, 3 free)
     3    e   extended
     4 Select (default p): Partition number (1-4, default 2): Using default value 2
     5 First sector (31459328-488397167, default 31459328): Using default value 31459328
     6 Last sector, +sectors or +size{K,M,G} (31459328-488397167, default 488397167):
     7 Command (m for help): The partition table has been altered!

     8 Calling ioctl() to re-read partition table.
     9 Syncing disks.
    10 /dev/sdb1 on /spool1 type ext4 (rw,noatime,nodiratime,barrier=0,data=writeback)
    11 /dev/sdb2 on /spool2 type ext4 (rw,noatime,nodiratime,barrier=0,data=writeback)
    12 /dev/sdc1 on /spool3 type ext4 (rw,noatime,nodiratime,barrier=0,data=writeback)
    13 /dev/sdc2 on /spool4 type ext4 (rw,noatime,nodiratime,barrier=0,data=writeback)
    14 /dev/sdd1 on /spool5 type ext4 (rw,noatime,nodiratime,barrier=0,data=writeback)
    15 /dev/sdd2 on /spool6 type ext4 (rw,noatime,nodiratime,barrier=0,data=writeback)
```

Figure B.1: NMS Monitoring Terminal: gathering real time commands results from remote servers. Specifically, the result of execution of the partition formatting command `fdisk` is depicted. It is used on the remote servers to configure the web cache server spools.

To illustrate such centralized monitoring, refer to Figure B.1. It depicts the output of the above discussed `hdsConfig` message as well as the output of `mountVirtualSpool` message on the NMS administrator terminal. As can be noticed, the file `status.dat` stores the contents of the `STATUS` messages. The output file lines were enumerated to favor the following elucidations:

Lines 1 to 9 list the output of the `fdisk` command, responsible for formatting partitions that will be used by a web cache server as spools. As shown in line 4, the primary partition type was selected (`default p`). The end of line 5 shows the numerical value of the first sector in the current partition under configuration (`31459328`). The numerical value of final partition sector can be observed as last value on line 6: `488397167`. Lines 7, 8, and 9, depict the finalization process of the `fdisk` command. Lines 10 to 15 show the output of the execution of mount `mount` commands. As can be noticed, six particular partitions were mounted as spools.