Sociedade Educacional Três de Maio - SETREM
Tecnologia em Redes de Computadores

ANDERSON MATTHEUS MALISZEWSKI
WILLIAN BAUM

PERFORMANCE CHARACTERIZATIONS OF IAAS PRIVATE CLOUDS FOR SCIENTIFIC AND ENTERPRISE WORKLOADS

Três de Maio
2017

ANDERSON MATTHEUS MALISZEWSKI

WILLIAN BAUM

PERFORMANCE CHARACTERIZATIONS OF IAAS PRIVATE CLOUDS FOR SCIENTIFIC AND ENTERPRISE WORKLOADS

Undergraduate Thesis

Sociedade Educacional Três de Maio

Faculdade Três de Maio

Computer Networks Technology

Advisor:

Ph.D. Dalvan Griebler

Três de Maio

2017

TERMO DE APROVAÇÃO

ANDERSON MATTHEUS MALISZEWSKI

WILLIAN BAUM

PERFORMANCE CHARACTERIZATIONS ON IAAS PRIVATE CLOUDS FOR SCIENTIFIC AND ENTERPRISE WORKLOADS

Relatório aprovado como requisito parcial para obtenção do título de **Tecnólogo em Redes de Computadores** concedido pela Faculdade de Tecnologia em Redes de Computadores da Sociedade Educacional Três de Maio, pela seguinte Banca examinadora:

Três de Maio, 19 de junho de 2017

Este trabalho é dedicado as nossas famílias por todo o apoio e pela paciência nos momentos em que estivemos ocupados e ausentes.

AGRADECIMENTOS

À faculdade SETREM, seu corpo docente e direção que oportunizaram a realização deste trabalho.

Ao orientador Dalvan Griebler, pela oportunidade de aprendizado e confiança em nossos estudos.

À equipe do LARCC, principalmente aos colegas Carlos Franco Maron e Adriano Wogel, pelo acompanhamento, discussões e suporte.

"This is your last chance. After this, there is no turning back. You take the blue pill — the story ends, you wake up in your bed and believe whatever you want to believe. You take the red pill — you stay in Wonderland, and I show you how deep the rabbit hole goes. Remember: all I'm offering is the truth. Nothing more."

Morpheus, *Matrix*

ABSTRACT

Private IaaS clouds offer an attractive environment to be used in enterprise and scientific fields providing advantages such as scalability, security and avoids dependence on third parties. However, one of the challenges is to port applications to the cloud environment without compromising performance. In response to this, the goal of this text is to characterize the applications performance in private IaaS clouds using scientific and enterprise applications. Therefore, the CloudStack was used to manage clouds and KVM and LXC-based were deployed as virtualization technologies. To represent real-world applications from the scientific and enterprise fields, it was used the NPB-OMP and PARSEC suite. These applications were benchmarked to characterize the high-performance and multi-tenancy environment. Statistical method was used to verify if there were significant differences among the clouds in each proposed environment. The results reveals that scientific and enterprise workloads are statistically different in the majority of the experiments performed in KVM and LXC-based clouds, however there are results with non-significant differences.

Keywords: Computer Network, Cloud Computing, CloudStack, KVM, LXC, IaaS, Multi-Tenancy.

RESUMO

As nuvens privadas IaaS oferecem um ambiente atraente para serem usado nos âmbitos empresariais e científicos, provisionando vantagens como escalabilidade, segurança e evitando a dependência de terceiros. No entanto, um dos desafios é portar as aplicações para o ambiente da nuvem sem comprometer seu desempenho. Em resposta a isso, o objetivo deste trabalho é caracterizar o desempenho das aplicações em nuvens privadas IaaS usando aplicações científicas e empresariais. Para tanto, o Cloudstack foi utilizado para gerenciar as nuvens KVM e LXC foram implantadas como tecnologias de virtualização. Para representar as aplicações do mundo real dos campos científico e empresarial, utilizou-se as suites NPB-OMP e PARSEC. Essas aplicações foram comparadas para caracterizar o ambiente de alto desempenho e multi usuários. Então, o método estatístico foi utilizado para verificar se houveram diferenças significativas entre as nuvens em cada ambiente proposto. Os resultados revelam que as cargas de trabalho das aplicações científicas e empresariais são estatisticamente diferentes na maioria dos experimentos realizados em nuvens com KVM e LXC, porém existem resultados com diferenças não significativas.

Palavras-Chaves: Redes de Computadores, Computação em Nuvem, CloudStack, KVM, LXC, IaaS, Multi-Tenancy.

LIST OF FIGURES

LIST OF TABLES

LISTINGS

LIST OF ABBREVIATIONS AND ACRONYMS

1-D	1-Dimension
3D	3 Dimension
ADI	Alternative Directions Implicit
AMQP	Advanced Message Queuing Protocol
API	Application Programming Interface
ARM	Architecture Review Board
AWD	Amazon Web Services
BIND	Berkeley Internet Name Domain
BLOBs	Binary Large Objects
BoT	Bag of Tasks
BT	Block Tridiagonal
BT-IO	Block Tridiagonal Input/Output
BT-MZ	Block Tridiagonal Multi Zone
CFD	Computational Fluid Dynamics
CG	Conjugate Gradient
CPU	Central Processing Unit
CRM	Customer Relationship Management
DC	Data Cube
DFT	Discrete Fourier transform
DHCP	Dynamic Host Configuration Protocol
DNS	Domain Name System

DSA	Distributed Systems Architecture
DT	Data Traffic
ED	Embarrassingly Distributed
EP	Embarrassingly Parallel
EPT	Extended Page Table
ERP	Enterprise Resource Planning
FFT	Fast Fourier Transform
FIMI	Frequent Itemset Mining
FP	Frequent Path
FT	Fourier Transform
GNU	GNU is Not Unix
HC	Helical Chain
HDTV	High-definition Television
HJM	Heath-Jarrow-Morton
HP	Hewlett-Packard
HPC	High Performance Computing
I/O	Input/Output
IaaS	Infrastructure as a Service
IPMI	Intelligent Plataform Management Interface
IS	Integer Sort
IT	Information Technology
KSM	Kernel Same-page Merging
KVM	Kernel-based Virtual Machine
LAN	Local Area Network
LARCC	Laboratory of Advanced Researches for Cloud Computing
LU	Lower-Upper
LU-MZ	Lower-Upper Multi Zone
LXC	Linux Container
MB	Mixed Bag
MC	Monte Carlo

MG	Multi Grid
MPI	Message Passing Interface
MZ	Multi Zone
NAS	NASA Advanced Supercomputing Division
NASA	National Aeronautics and Space Administration
NPB	NAS Parallel Benchmark
NSD	Name Server Daemon
OMP	Open Multi-Processing
OS	Operational System
PaaS	Platform as a service
PC	Personal Computer
PDE	Partial Differential Equation
POSIX	Portable Operating System Interface
PThread	POSIX Threads
QA	Quality Assurance
REST	Representational State Transfer
SA	Simulated Annealing
SaaS	Software as a Service
SAN	Storage Area Network
SETREM	*Sociedade Educacional Três de Maio*
SLA	Service Layer Agreement
SP	Scalar Pentidiagonal
SPH	Smoothed Particle Hydrodynamics
SPI	SaaS PaaS and IaaS
SP-MZ	Scalar Pentadiagonal Multi Zone
SSH	Secure Shell
SSOR	Symmetric Successive Over-Relaxation
SPI	SaaS PaaS and IaaS
UA	Unstructured Adaptive
UI	User Interface

VIPS	VASIRI Image Processing System
VM	Virtual Machine
VP	Visualization Pipeline
VPN	Virtual Private Network
VT	Virtualization Technology

CONTENTS

INTRODUCTION

Cloud computing is an emergent paradigm which refers to applications and services being distributed over network, using virtualized resources. They are accessed by Internet protocols and network standards, which provide on-demand computational resources, (SOSINSKY, 2010). Thus, this infrastructure has become an attractive environment to scientific and enterprise applications, offering high flexibility in resource allocations and elasticity, at a relatively low cost (STONEBRAKER; PAVLO; TAFT; BRODIE, 2014).

Otherwise, as appointed by Iosup et al. (2011), scientific and enterprise workloads often require High-Performance Computing (HPC) capabilities, which offer a better condition for hosting such applications. Consequently, migrating these applications that are traditionally developed to the HPC environment to the cloud environment using virtualization technologies may cause performance degradation. Therefore, porting applications to the cloud without compromising performance is one of the major challenges of cloud computing.

Several studies about IaaS cloud tools were performed at the Laboratory of Advanced Researches on Cloud Computing (LARCC)[1]. The study of Maron (2014), Maron et al. (2014) and Maron, Griebler and Schepke (2014), performed an evalu-

[1]http://larcc.setrem.com.br

ation and comparison between parallel and scientific applications using OpenStack[2] and OpenNebula[3] cloud tools. Thus, these IaaS tools have been deployed and benchmarked to simulate the scientific applications' behavior in the native and cloud environments, allowing comparison among the tools. Their main contribution was the methodology to evaluate IaaS tools factor through the NAS Parallel Benchmark and intensive workloads over specific properties in the environment such as network, storage, memory, and processing (MARON; GRIEBLER; SCHEPKE; FERNANDES, 2016; VOGEL; GRIEBLER; MARON; SCHEPKE; FERNANDES, 2016). Therefore, the previous research only considered the use of a single virtualization technology (KVM) and high-performance computing environment (one instance per node).

On the other hand, the survey performed in Vogel (2015) and Vogel et al. (2016) had the goal of evaluating these IaaS tools, concerning the features and support for robustness (flexibility and resilience). Also, a survey state of the art for performance on a cloud was conducted to highlight cloud's challenges and potential solutions for reducing its performance overhead. This survey demonstrated that the open source cloud IaaS solutions can provide high robustness levels and are suitable for enterprise applications. Moreover, through the results of the evaluation, it has been determined that CloudStack is the most flexible IaaS tool as well as OpenStack is the most resilient.

This research was also developed in the LARCC and contributed to the previous work. The main difference is that it used the workloads of benchmark suites to simulate the performance of real-world applications. Moreover, different deployments will be evaluated in the IaaS cloud and native environment. The expected contribution is a performance-aware characterization to deploy a cloud environment for scientific and enterprise workloads.

The thesis is organized in 3 chapters. Chapter 1 presents topics about the project, such as the theme, problem, objectives and the methodology. The next one

[2]https://www.openstack.org/
[3]https://opennebula.org/

(Chapter 2) introduces the Literature Review, which describes the main subjects related to the theme (cloud computing, grids, cluster). Finally, Chapter 3 shows the results of the study, focusing on the exposure of tool characterization.

CHAPTER1: RESEARCH PLAN

This chapter introduces and contextualizes the methodology used in the research.

1.1 THEME

Characterizing performance on private IaaS clouds for scientific and enterprise application workloads.

1.1.1 Theme Delimitation

This research intends to investigate performance issues and bottlenecks in private IaaS clouds. The goal is to characterize performance-aware scenarios by testing different virtualization technologies, cloud management tools, multi-tenancy, workloads from scientific and enterprise domains and deployment optimizations. Cloud deployments will be configured using private IaaS management tools (CloudStack, OpenStack, and OpenNebula) with KVM and LXC-based intances, which were chosen based on previous studies presented by Vogel et al. (2017), Vogel et al. (2016) and Adriano Vogel Carlos A. F. Maron (2015). To represent the real world applications on Scientific and Enterprise domains, workloads will be used provided by representative benchmark suites (*i.e.*, NAS-NPB and PARSEC). The performance result should provide enough empirical data to characterize the best deployment scenarios for each workloads.

The project is authored by Anderson Mattheus Maliszewski and Willian Baum as a requirement for the final undergraduate thesis in the Computer Network course, which is advised by the Ph.D. Dalvan Jair Griebler at Sociedade Educacional Três de Maio (SETREM). The period of the research is between October of 2016 and July of 2017.

1.2 PROBLEM

As emphasized by Buyya, Vecchiola and Selvi (2013), the paradigm of cloud computing is changing the way how services are delivered to costumers and end users, such as water, electricity, gas, and telephony (on-demand). In addition, there is a growing demand for accessing information anywhere and anytime. Based on these needs, cloud computing is an emerging technology to provide a new way to organize and use the hardware resources through virtualization, which allows the provision of services on-demand (BUYYA; BROBERG; GOSCINSKI, 2010).

Currently, there are three major factors that make most organizations migrate their solutions to the cloud, they are: Costs; deployment; and organizational changes. This can be done because, cloud computing focuses on maximizing the effectiveness of shared resources (BADGER; CHAPMAN; PATT-CORNER; VOAS, 2012). However, virtualized environments may experience performance degradations due to the overhead of virtualization software (ROSSO, 2015). There is a risk involved in deploying the cloud environment as well, impacting considerably on the application performance. Therefore, scientific and enterprise workloads must be characterized in private cloud environments to identify the best deployment scenario. This will be useful to stakeholders to improve their environment according to the type of application and requirements, avoiding a significant performance degradation and providing efficient resource usage.

Finally, how should the cloud be deployed for enterprise and scientific workloads to achieve the best performance?

1.3 HYPOTHESES

1. The performance characteristics of scientific and enterprise workloads are statistically different among deployed cloud scenarios.

2. The performance characteristics of the scientific workloads on the deployed cloud scenarios are statistically different with respect to the native computing environment.

3. The performance characteristics of the enterprise workloads on the deployed cloud scenarios are statistically different with respect to the native computing environment.

1.4 VARIABLES

- Performance

- Cloud Platform

- Virtualization Technologies

- Scientific and Enterprise Applications

1.5 OBJECTIVES

This section presents the study goals.

1.5.1 General Objective

The goal of the research is to investigate performance issues and bottlenecks in private IaaS clouds for scientific and enterprise workloads, using different virtualization technologies, cloud management tools, deployment optimizations and important application features (for instance, scientific workloads that target distributed and shared memory architectures as well as enterprise workloads, which represents concurrency and client-server environments).

1.5.2 Specific Objectives

- Review the literature of the following subjects: Private IaaS; Virtualization technologies; Cloud management tools; Deployments.
- Define a set of benchmark suites that represents scientific and enterprise application workloads.
- Create and deploy a set of deployment scenarios.
- Run the defined scientific and enterprise workloads.
- Compare the results of the benchmarks.
- Characterize the best deployment scenarios.

1.6 JUSTIFICATION

The fast growth in demand for computing power, along with the new possibilities brought by the Web 2.0, open their way to the full adoption of the cloud computing (CHANG, 2015). The utilization of the cloud computing, whether public, private or hybrid, can work with green IT, as long as it has proven to be effective and financially sustainable, as IT costs in staff, hardware, and equipments decreases (SOSINSKY, 2010).

The cloud computing utilizes a stack architecture, whereas the virtualization stays directly above hardware resources, offering support to the high level layers, such as IaaS, PaaS and SaaS (CHANG; WALTERS; WILLS, 2013).

However, although the technologies that are used by cloud computing have been improved over the past three decades (TANENBAUM; AUSTIN, 2013), the abstraction layer provided by hypervisors, comes with a cost, performance. In comparison to the native environment, they do not have the same abstraction layer as cloud computing.

One of the technical risks involved in utilization of cloud is whether the performance will be worse than expected for the native en. Therefore, the use of representative benchmark tools to simulate workloads of real world applications will help the service provider to choose a better option according to their needs, allowing them to compare the performance of the cloud environment before the decision making. Furthermore, it is necessary to use scientific methods in order to characterize and compare the performance for each deployment.

The combination of application features, drivers and virtualization technologies in the OpenStack, CloudStack and OpenNebula deployments methods, provides a specific environment for each technology, resulting in a different performance when utilizes scientific and enterprise workloads. However, the benefits and risks should be clear to the organization's strategic goals. A survey on this area will guide the stakeholders needs, allowing them to provide better planing to deploy the cloud in their production environment, and provides the ability to choose the best ways to deploy for specific use.

This research comes from the need for a clear understanding of questions that are little discussed in the literature and allow to aggregate knowledge, in order to provide the frameworks in cloud computing area. When benchmarks of real applications are performed, and makes a comparison among them, it is possible to find the best environment for an application type. Making it possible to the stakeholder, take full advantage of the hardware resources, accomplish the service layer agreement (SLA), and bring the best of cloud computing.

1.7 METHODOLOGY

This section describes the methods that will be used to reach and treat the results of the research.

1.7.1 Methods

In this research, it will be used the quantitative method because the goal is to analyse the benchmarks results statistically. To achieve this goal, a study method will be used to highlight the performance to each kind of workload and each cloud platform. In addition, a deductive method will be used because of the experimental research.

Before presenting how the hypotheses are evaluated, observe that their validation are specifically applied to the achieved results and specific scenarios tested. Consequently, the experiment plan and methodology will limit the achievement of each one of the hypotheses. The general assumption should be avoided as performance experiments are always specifically designed to each particular computing environment targeted. Therefore, each hypothesis and its way to validate are presented on the following topics:

1. **The performance characteristics of the scientific and enterprise workloads are statistically different among deployed cloud scenarios.** For instance, it is possible to have a deployed cloud with a KVM-based environment and another one with an LXC-based environment. The hypothesis requires deploying at least two cloud environment types that can be using the same or different cloud plat-

form and choose representative benchmarks that mimic/simulate scientific and enterprise workload. Then, the basic performance metrics (*e.g.,* execution time) defined in the experiments later will be measured when running the benchmarks. This is because each kind of workload has its own requirements. The statistic significance will be analyzed by using the appropriate statistical hypothesis test, which will be known only after the tests were run and analyzed.

2. **The performance characteristics of the scientific workloads on the deployed cloud scenarios are statistically different with respect to the native computing environment.** Supposing that the cloud scenarios were already deployed for the second hypothesis, it will be necessary only to run the same benchmarks in a equivalent native computing environment, measuring the same basic performance metrics. After, the statistic significance will be measured by using the appropriate statistical hypothesis test and compare both native and cloud scenarios running scientific workloads.

3. **The performance characteristics of the enterprise workloads on the deployed cloud scenarios are statistically different with respect to the native computing environment.** Supposing that the cloud scenarios were already deployed for the second hypothesis, it will be necessary only to run the same benchmarks in a equivalent native computing environment, measuring the same basic performance metrics. After, the statistic significance will be measured by using the appropriate statistical hypothesis test and compare both native and cloud scenarios running enterprise workloads.

1.7.2 Procedures

Statistical research: This procedure will be used because it is needed to show the significant differences between the benchmark results.

Exploratory research: It will be used because it is necessary to review the literature in searching of solutions for cloud and virtualization already explored by other researchers.

1.7.3 Research Techniques

Experimental Technique: It will be used because the results of this research are not known, and it needed to be conducted by experiments. Thus, it will be conducted creating the specific scenario to simulate with benchmarks tests.

1.8 RESOURCES

This section describes the resources that will be used during the development of the research.

1.8.1 Human Resources

The human resources are the frequent contact with the advisor, teachers and fellow students.

1.8.2 Material Resources

The material resources are the books, articles, equipments, papers, digital libraries, computers and IT devices.

1.8.3 Institutional Resources

The institutional resources used are the library, IT labs, network lab, coordination of the Computer Networks Technology course and the LARCC infrastructure.

1.9 SCHEDULE

The Table 1.1 shows the proposed activities and the period they were performed during the research. The table is divided in columns, counting the activities and sh the months in which they will be executed. The gray cells show the expected period settled for each activity and the cells with the character "X" representing the objectives already achieved.

1.10 ESTIMATED COST

The Table 1.2 shows the financial resources needed to complete the project. The first column describes the activity, the second describes the amount of this activities, the third describes the unit value in national currency and the fourth shows the sum of the amounts in national currency.

Activity	**2016**			**2017**					
	Oct	**Nov**	**Dec**	**Jan**	**Feb**	**Mar**	**Apr**	**May**	**Jun**
Write a research project	X	X	X						
Deliver the research project			▨						
Study private IaaS cloud tools		X	▨	▨					
Study of related works		X	▨	▨					
Present and deliver the research progress			X						
Study of benchmarks				X	X	X			
Study of scientif and enterprise workloads		▨	X	X	X	X			
Perform the benchamark tests				X	X	X	X		
Analysis and validation of the hypothesis						▨	▨	▨	X
Write the final thesis							▨	X	X
Present the final thesis									X

Source: Baum, Maliszewski, Griebler, 2017.

Table 1.1: Schedule

Activity	**Amount**	**Unit Value**	**Total Value**
Number of printouts	1500	R$ 0.25	R$ 375.00
Spiral binding	3	R$ 5.00	R$ 15.00
Hardcover binding	1	R$ 70.00	R$ 70.00
Gasoline	1000	R$ 3.89	R$ 3.890.00
Writing and publish articles	3	R$ 150.00	R$ 450.00
English correction	3	R$ 120.00	R$ 360.00
Openstack course	1	R$ 1.500.00	R$ 1.500.00
Work hours	1600	R$ 60.00	R$ 96.000.00
Total			R$ 102.660.00

Source: Baum, Maliszewski, Griebler, 2017.

Table 1.2: Estimated Costs.

CHAPTER2: LITERATURE REVIEW

This chapter defines the terms with their characteristics and topics, following the literature.

2.1 DEFINITION OF TERMS

This section defines the terms related to the research. It explains the main characteristics and details of the topics.

2.1.1 Benchmarks

As the number of computing architectures has increased, the diversity of platforms makes it difficult to predict the behavior of an application in certain architectures. The computing community faced the problem, by using a practical approach to provide mathematically the comparison of computational performance using benchmarks (GUSTAFSON; SNELL, 1995).

Benchmark is essentially an algorithmic structure used on computational system with the characteristics of a real-world application, allowing to mimic the behavior in those system. Thus, as appointed by Zhu (2014), benchmark is a test method to evaluate and identify the performance of the computer characteristic. Therefore, the benchmarks results can be compared across different platforms (BUYYA; BROBERG;

GOSCINSKI, 2010). It allows to measure scientifically the performance implementation, helping the engineering to quantify and improve standards.

2.1.2 Process

A process is essentially a container that stores all the information necessary to run a program. It is associated with each process is its address space, a list of memory locations, ranging from 0 to a maximum, which the process can read and write. Thus, the address space contains the executable program, the program data, and its stack. Also associated with each process is a set of features, usually including registers, a list of open files, pending alarms, related process lists, and all other information needed to run a program (TANENBAUM; BOS, 2014).

The authors Silberschatz, Galvin and Gagne (2012) emphasized, that a program by itself is not a process. A Program is a passive entity, such as a file containing a list of instructions stored on the disk (executable files). Thus, in contrast, a process is an active entity, with a program counter specifying the next statement to execute and a set of associated resources. It means that a traditional process has a single thread of control.

Also, as emphasized by Tanenbaum and Bos (2014), the operating system periodically interrupts a running process and starts another because the former would have exceeded its CPU sharing time. When a process is temporarily suspended this way, it should be restarted later, from exactly the same point it was when it was interrupted. This means that all process-related information must be explicitly saved somewhere during the suspension.

Moreover, on many operating systems, all information related to a process other than the content of its own address space is stored in an operating system table called a process table, which is an arrangement of structures, one for each existing process.

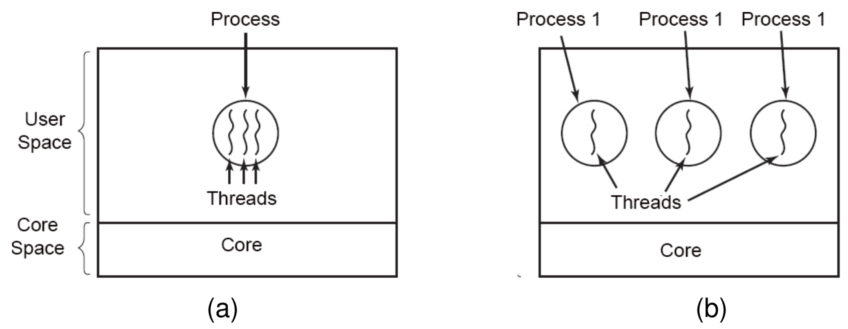
2.1.3 Threads

A Thread is basically a process inside another process. A process typically has one address space and a single control flow. Often there is a situation where it is desirable to have multiple control flows in the same address space, running almost in parallel, as if they were separate processes. These control flows are known as Threads (TANENBAUM; BOS, 2014).

In addition, as emphasized by the same author, a thread has a program counter that controls which statement is going to be executed. It has registers, which contain their current working variables. It has a stack, which contains the execution history, with a block for each called function, but for which there has not yet been a return.

However, although a thread must be run in some process, the thread and its process have different concepts. Processes are used to group resources and threads are the entities programmed to run on the CPU. What is enhanced by Threads in the process model is that it allows multiple executions to occur in the same process environment quite independently of each other. In Figure 2.1(a), a single process with three control threads is shown. In contrast, in the Figure 2.1(b), three traditional processes are shown. Each process has its own address space and a single control thread.

Although in both cases there are three threads, in Figure 2.1(a) all three share the same address space, while in Figure 2.1(b) each of them operates in a different address space.



Source: Extracted from Tanenbaum and Bos (2014).

Figure 2.1: A process with three threads (a). Three processes, each one with a thread (b).

2.1.4 Performance Evaluation

One of the main goals of computing, is the potential to provide to their owners the benefits of economy allied to performance efficiency, so this will impact on the overall cost of the business, and on the production environment (IOSUP; OSTERMANN; YIGITBASI; PRODAN; FAHRINGER; EPEMA, 2011). However, in order to quantify the performance among the diverse computational architectures, we need to evaluate some key features of the system. They are described below.

2.1.4.1 Response Time (Latency)

One of the characteristics of computer devices is that they are constantly generating and transferring data between them (NEUHAUS; FEINBUBE; JANUSZ; POLZE, 2015). Thus, an important way to evaluate the performance is to measure the latency.

As defined by Badger et al. (2012) and Marinescu (2013), latency is the elapsed time (delay) from the instant an operation is initiated until it is effected is sensed. Thus, the result of a low latency environment will have a positive impact on the overall response time. This can be represented by the formula:

$$Latency = \frac{TransferredDate}{Time} \tag{2.1}$$

2.1.4.2 Overhead

Overhead is a common issue of virtual machines, because a new layer is introduced between the hardware resources and the operating system. In addition, as mentioned by Xu, Liu and Vasilakos (2014), despite resource isolation techniques, resources shared by the virtual machines, such as CPU cache and I/O bandwidth, introduces an overhead. So, communication overhead is the time needed for processors to communicate and possibly exchange data while executing tasks (EL-REWINI; ABD-EL-BARR, 2005).

Moreover, it is an activity that do not contribute with the execution itself, but the time to instructions becomes actions. Thus, as emphasized by Janssen and Nielsen (2008), highs rates of communication overhead causes degrading performance.

2.1.4.3 Throughput

Throughput is the maximum bandwidth rate achieved in a computer environment in a given data stream. As higher the throughput is, higher is the performance. However, as emphasized by Blokland, Mengerink and Pol (2013), an insufficient throughput, can produce a performance risk to the cloud environment, creating a bottleneck due to the lack of capacity, increasing the latency.

As emphasized by Chandrasekaran (2014), the ideal throughput should coexist in a cloud environment and obtain from the system the resources that best match the application requirements. Therefore, it is appointed by Baer (2010), as a metric that represents the amount of work per unit of time:

$$Throughput = \frac{Instructions(Number)}{Latency} \tag{2.2}$$

2.1.4.4 Speedup

Speedup is the performance gain from original task compared to the improved task. Thus, the speedup measures the effectiveness of parallelization (MARINESCU, 2013).

In other words, as initially presented by Amdahl's Law (AMDAHL, 1967), speedup shows how a task will run faster by using the computer with enhancement as opposed to the original computer.

The theoretical speedup equation in latency of a certain task is represented by Hennessy and Patterson (2012) of the following formula:

$$Speedup = \frac{Performance(Improved)}{Performance(Normal)} \tag{2.3}$$

2.1.4.5 Efficiency

As emphasized by Baer (2010), a smaller execution run-time implies better performance, improving better resource usage. Thus, efficiency is an important metric that reveal the resources utilization, calculating the speedup (performance improvement obtained by adding resources) per number of processors (n), the result will be the efficiency.

$$Efficiency = \frac{Speedup}{Number(Processors)} \tag{2.4}$$

In addition, El-Rewini and Abd-El-Barr (2005) define the efficiency as the ratio between the speedup factor and the number of processors (n). Is proposed in the parallel computation with serial sections mode, to add the communication overhead to the equation, and a fraction of serial computation (f), allowing to compare them.

$$\xi(\text{No Communication Overhead}) = \frac{1}{1 + (n-1)f} \tag{2.5}$$

$$\xi(\text{With Communication Overhead}) = \frac{1}{f(n-1) + 1 + n(t_c + t_s)} \tag{2.6}$$

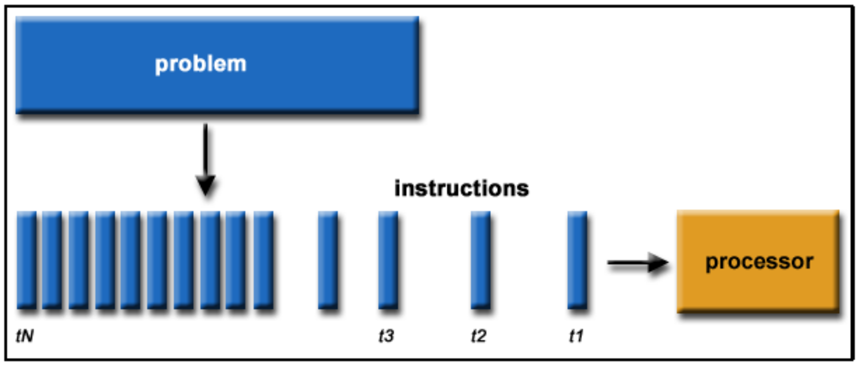
2.1.4.6 Resource Isolation

The cloud computing environment provides resource virtualization by sharing a common hardware between instances. Sakr and Gaber (2014) appointed that resource isolation is managed by the operating system in order to efficiently control and distribute the tasks and system calls.

However, one of the key challenges is to provide an efficient use of these resources, avoiding the overhead. As mentioned by Barker and Shenoy (2010), multiple virtual machines running disparate applications may share the same physical server, thus inducing penalties among processes, especially for running latency-sensitive applications. Therefore, low level of resource isolation will cause considerable overhead between systems (BESERRA; MORENO; ENDO; BARRETO; SADOK; FERNANDES, 2015a), hence a low performance.

2.1.5 Parallel Computing

Parallel computing is a combination of multiple processing elements in a single large system. It is highly used to solve huge problems and perform different tasks in the shortest possible time. According to Mattson, Sanders and Massingill (2004), parallel computing uses the main key to exploit concurrency between problems. It decomposes the problems into sub-problems and then executes them at the same time. In the Figure 2.2, a description of a traditional serial computation is shown, which is the opposite of parallel computing. The serial way breaks the problem into a discrete series of instructions, and then the instructions are executed by a master processor.



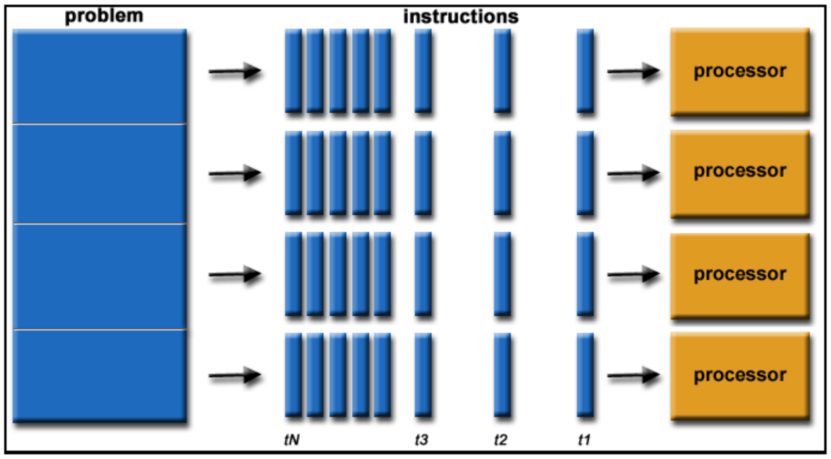
Source: Extracted from Silberschatz, Galvin and Gagne (2012).

Figure 2.2: Serial problem execution.

Unlike the serial, parallel computing is the simultaneous use of multiple compute resources to solve a computational problem (2.3). It first decomposes a main problem into distinct parts that can be solved concurrency. Then, each part is broken into instructions and after that the instructions of each part execute simultaneously on different processors (BARNEY, 2012).

The author Barney (2012) justifies the use of parallel computing for the following reasons:

- Save time and money: The greater use of resources in a task will shorten its time



Source: Extracted from Barney (2012).

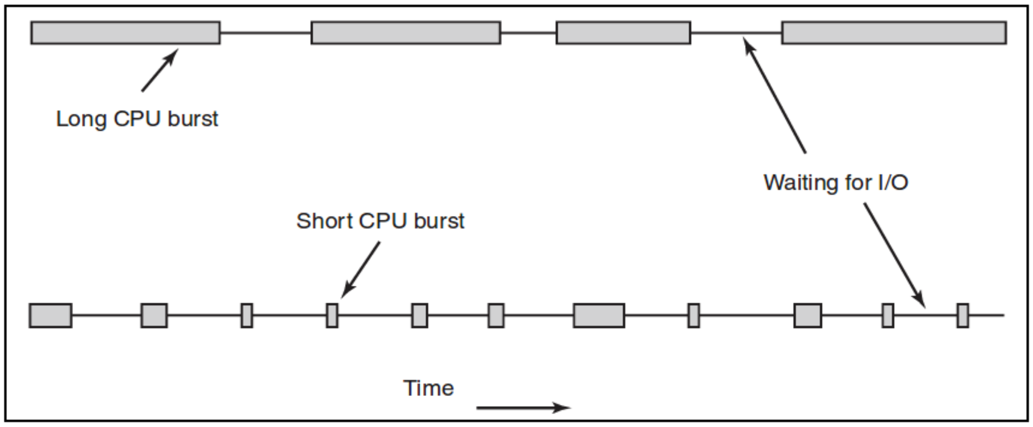
Figure 2.3: Parallel problem execution.

to completion, potentially cost-saving.

- Solve large and complex problems: Many problems could be so large and complex to solve that it becomes impossible to solve that on a single computer. Generally the biggest limitation is related to the amount of memory available, which does not happen in a parallel environment.
- Provide Concurrency: In a single computer environment one task could be solved at a time. In a parallel environment, many tasks could be solved simultaneously.
- Take advantage of non-local resources: Allows to use resources on a wide area network, or even the Internet.

2.1.6 I/O and CPU Bound

According to Tanenbaum and Bos (2014), some processes spend most of their time computing, while others spend most of their time waiting for I/O. The first are called bounded by the CPU (compute-bound or CPU-bound); The latter are those limited by I/O (I/O-bound). Processes limited by the CPU usually have long CPU bursts and sporadic I/O waits; Whereas I/O-bound processes have small peaks in CPU usage and frequent I/O waits. The Figure 2.4 shows the difference between the I/O and CPU bound.



Source: Extracted from Tanenbaum and Bos (2014).

Figure 2.4: I/O and CPU bound comparison.

In addition, the same author emphasized that I/O-oriented processes are so called because between one request and another for I/O, they do not perform much computation, not because they have especially long I/O requests. The time to read a disk block is always the same regardless of the time it takes to process the data that arrives later. As CPUs become faster, processes tend to be more limited by I/O. This effect occurs because CPUs are getting much faster than disks. The basic idea is that if an I/O-oriented process wants to run, that opportunity should be quickly given to it as it will execute its disk requests, keeping the disk busy.

2.2 BACKGROUND

Cloud computing is an emerging paradigm that provides computing resources through service levels, (VOGEL; MARON; GRIEBLER; SCHEPKE, 2016). Therefore, it is built on several technologies and models. In this approach, to understand the development of this work, it is necessary to contextualize these technologies and related techniques, presented in the next sections.

2.2.1 Statistical Hypotheses Testing

In approaching a scientific research, it is necessary to prove or not the hypotheses created on the obtained results. Following this line, statistical methods are used to analyze and compare all data. Therefore, the following sections will describe the items related to the statistical hypothesis test.

2.2.1.1 Hypotheses Test

According the authors Ron Larson (2010) a hypothesis test is a process that uses statistical samples to test an affirmation about a value of a population parameter.

In addition Freund (2009) argues that a statistical hypothesis is a statement or a conjugation about a parameter (or parameters), about a population (or populations); It may also refer to the type or nature of the population.

To develop the processes of testing statistical hypotheses, one must first know what to expect when a hypothesis is true, and it is for this reason that the hypothesis is often formulated opposite to what is expected to be proved.

Once the hypothesis has been made that the affirmations of this are equivalent, we enter the null hypothesis.

2.2.1.2 Null Hypotheses

According to Field (2013), a null hypothesis[1] usually indicates that an element is missing. It is denoted by H_0. A null hypothesis is used because it is impossible to prove the experimental hypothesis using statistics, but a null affirmation can be rejected.

[1]The term "null hypothesis" was introduced by Ronald Fischer (FISHER, 1925). If the assertion in the null hypothesis is not true, then the alternative hypothesis must be true.

In addition Ron Larson (2010) states that a null hypothesis H_0 is a statistical hypothesis that contains an equality affirmation, such as $\leq$, = or $\geq$.

2.2.1.3 Alternative Hypotheses

According to Field (2013), this type is the opposite of the null hypothesis, always being a present effect. It is denoted by H_1. In addition, the author Freund (2009) mentioned that the alternative hypothesis should always be formulated together with the null hypothesis.

Another author, Ron Larson (2010) defines the alternative hypothesis as a complement to the null hypothesis. It is an affirmation that must be true if the H_0 is false and contain a strict inequality affirmation, such as $>$, neq or $<$. Taking an example, one can easily understand the differences between these two types of hypotheses generated bellow:

- **Null Hypothesis:** A 5% increase in the price of a particular product will not adversely affect sales.
- **Alternative Hypothesis:** A 5% increase in the price of a particular product will affect sales.

2.2.1.4 Type I Error

According to Field (2009), a type I error occurs when one believes that there is a genuine "effect" in the population, when in fact it does not exist. Following Fisher criteria, the probability of occurrence of this error is 0.05 (5%). This error can also be named as α. Thus, if a given data is replicated 100 times, it can be expected that on five occasions, the statistical test suggests that there is an "effect" on the population, even when it does not exist.

2.2.1.5 Type II Error

The opposite of type I error, is known as type II error, and its occurrence representation is given by β. According to the authors Field (2009), this error occurs when it is believed that there is no "effect" on a population when it actually exists. Typically, this error occurs when you have obtained a small statistic test. In the Table 2.1, it summarized the typical situation of both errors.

	Accept H_0	**Reject H_0**
H_0 **is true**	Right decision	Type I Error
H_0 **is false**	Type II Error	Right decision

Source: Extracted from Freund (2009).

Table 2.1: Error I and Error II.

If the null hypothesis H_0 is true and accepted, or false and rejected, the decision is correct in both cases. In addition, if it is true and rejected, or false and accepted, the decision is wrong in both cases. In practical, the first error is known as Type I error and the second is Type II error (FREUND, 2009).

2.2.1.6 Significance Test

According to Freund (2009) the main role of statistical analysis is to establish if the results obtained are statistically significant according to predetermined limits. In the next sessions, the items related to the significance of the results will be described.

2.2.1.7 Statistically Non-Significant / Statistically Significant

According to Freund (2009) when the difference between what is expected and what is observed is so small that it is reasonably attributable to chance, a result is statistically non-significant or not significant. On the other hand, if the difference between what is expected under the null hypothesis and what is observed in a sampling is too large for it to be reasonably inaccurate, the null hypothesis is rejected and therefore have a statistically significant result.

It is important to note that these expressions are always employed in view of previously chosen 'levels of significance'.

2.2.1.8 Significance Level

According to Field (2009) it is the limit that is taken as the basis to affirm that a certain deviation is due to the chance or not. The levels P = 0.05 and P = 0.01, i.e., 5% and 1% respectively, are accepted as statistically significant. From a conventionalized level of significance (α) deviations are due to the law of chance and the result is considered non-significant. Thus, if α = 5%, is represented in the example image 2.5:

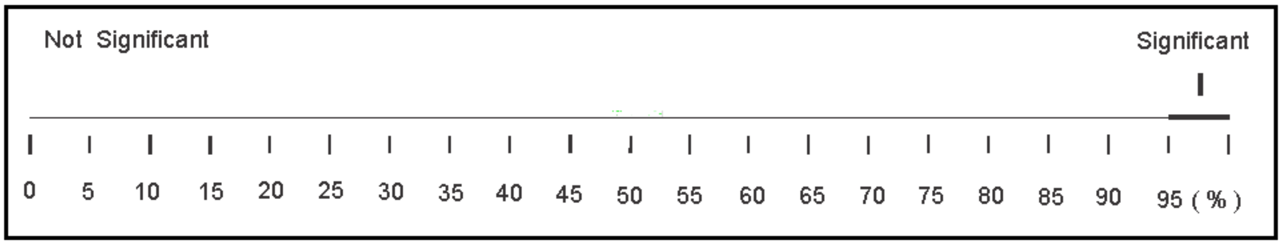


Figure 2.5: Significance Level Example.

Source: Extracted from Field (2009).

In practice, the 5% error probability limit is considered to be satisfactory, differences that have a probability above this limit are not significant.

The level of significance must be established before the experiment is realized and corresponds to the risk of rejecting a true hypothesis or accepting a hypothesis. The significance of a result is also called the $p - value$.

2.2.1.9 Z Test

Used when distributed Sigma population is known. Z is a random value that has a standard distribution population and can be represented by the formula 2.7. Thus, according to Freund (2009) by using standard units, it can formulate hypothesis and criteria to test the hypothesis as represented in the Table 2.2.

$$z = \frac{\bar{x} - \mu_0}{\sigma/\sqrt{n}} \tag{2.7}$$

Alternative Hypothesis	**Reject the null hypothesis if**	**Accept the null hypothesis or reserve the judgment**
$\mu < \mu_0$	$z \leq -z_a$	$z > -z_\sigma$
$\mu > \mu_0$	$z \geq z_a$	$z < z_\sigma$
$\mu \not\equiv \mu_0$	$z \leq -z_\sigma/2 or z \geq z_\sigma/2$	$-z_\sigma/2 < z < z\sigma/2$

Source: Extracted from Freund (2009).

Table 2.2: Formulate hypothesis and criteria to test hypothesis.

2.2.1.10 T Test

It is very similar to the Z test, but is most often used when the population of standard deviation is unknown and can be represented by the formula 2.8. However, according to Freund (2009), to perform this test it is necessary to estimate where the population samples comes from to correctly represent the normal distribution.

$$t = \frac{\bar{x} - \mu_0}{s/\sqrt{n}} \tag{2.8}$$

2.2.1.11 Differences between means

According to Freund (2009) there are many problems in deciding whether an observed difference between two samples means can be attributed to chance or whether it is an indication of the fact that the two samples come from populations with different means. Two processes used to test this difference are the standard error and the standard deviation described in the following sections.

2.2.1.12 Standard Deviation

According to Field (2009) the standard deviation aims to demonstrate the regularity of a set of data in order to indicate the degree of oscillation of these in comparison with the average of the values of the set.

To determine the standard deviation, it is first necessary to calculate the mean, made by the formula in the equation 2.9, where "μ" is the mean, "Σ X" is the sum of the values and X represents each of the numbers of the sum, finally "N" indicates the total size of the population.

$$\mu = \frac{\Sigma X}{N} \tag{2.9}$$

The standard deviation is calculated with the formula in the equation 2.10, where "σ" is the standard deviation representation, "Σ" is the sum, "X" is the sum number, "μ" is the mean and finally the "N" is the total number of the population..

$$\sigma = \sqrt{\frac{(\Sigma(X-\mu)^2}{N}} \tag{2.10}$$

2.2.1.13 Standard Error

According to Field (2009) the standard error (SE) estimates the variability among the means of the sample that would be obtained if several samples of the same population were collected.

In addition, the authors Ron Larson (2010) emphasized that the standard error is used to determine the accuracy with which the sample mean estimates the population mean. Lower values of the standard error of the mean indicate more accurate estimates of the population mean. In the equation 2.11, the formula for calculating the standard error is displayed. The Greek letter "σ" is the standard deviation and the "n" is the sample size.

$$SE = \frac{\sigma}{\sqrt{n}} \tag{2.11}$$

2.2.1.14 Analysis of variance - ANOVA

According to Field (2009) a $ttest$ tests the hypothesis with two samples that have the same mean. Similarly, the ANOVA determines whether two or more means are equal by testing the null hypothesis. An ANOVA produces a F statistic or F ratio, which is similar to the t statistic, in that it compares the amount of systematic variance in the data with the amount of non-systematic variance. In other words, F is the model's reason for its error (FIELD, 2009).

ANOVA is an omnibus test, which means that it tests a global experimental effect, although it says whether experimental manipulation was generally successful, it does not provide specific information about which groups were affected. Assuming that one experiment was conducted with three different groups, the ratio F says that the means of these three samples are not the same. However, there are several ways in which the means may differ. The first possibility is that the three sample means are significantly different. A second possibility is that the means of groups 1 and 2 are the same, but group 3 has a significantly different mean. Another possibility is that groups 2 and 3 have similar means, but group 1 has a significantly different mean. Finally, groups 1 and 3 may have similar means, but group 2 has a significantly different mean. So in one experiment, the F ratio only tells us that experimental manipulation has had some effect, but it does not specifically tell us what the effect was (FIELD, 2009).

2.2.1.15 Homogeneity of Variance

According to Field (2009) homogeneity of variance means that as it moves between levels of one variable, the variance of the other should not change. If data groups were collected, this means that the variance of the output variable or variable should be the same in each of these groups. If continuous data were collected, this assumption means that the variance of one variable must be stable at all levels of the other variable.

2.2.2 Test of Normality

According to Field (2009) normality tests are used to verify if the probability distribution associated with a dataset can be approximated by the normal distribution. In the next sections is described the Kolmogorov-Smirnov and Shapiro Wilk tests.

2.2.2.1 Test of Kolmogorov-Smirnov / Shapiro Wilk

According to Field (2009), both tests compare "scores" from a sample to a normal distribution model of the same mean and variance of the values found in the sample. If the test is non-significant ($p > 0.05$), it reports that the sample data does not differ significantly from a normal distribution (that is, they may be normal). On the other hand, if the test is significant ($p < 0.05$), the distribution in question is significantly different from a normal distribution (that is, it is non-normal).

However, both tests have limitations because with large samples it is very easy to obtain significant values from small deviations from normality and, therefore, a significant result does not necessarily inform us if the deviation from normality is sufficient to undermine the statistical procedures that will be applied to the data.

2.2.3 High Performance Computing

High Performance Computing (HPC) is characterized to be a massive number of computers installed on facilities (clusters and supercomputers) for performing large scale experiments (VECCHIOLA; PANDEY; BUYYA, 2009).

HPC focuses on performance, and it is used in many scientific and enterprise fields, such as airplane projects, drug development, global climatology, military applications, financial analysis, space research and other applications that requires high degrees of accuracy. In addition, as defined by Yang and Guo (2005), supercomputers from HPC domain are also known to use state-of-the-art technology, providing the highest levels of performance for certain applications, but also with an expansive cost.

According to Cisco (2008) the HPC has three main types in the enterprise environment, which are:

- **Tightly Coupled:** Applications run on all compute nodes simultaneously in parallel, having one master node to determinate the input processing for each compute node.

- **Distributed I/O processing:** Computing is balanced between master nodes, then divided among the compute nodes for parallel processing. It is commonly used in search engines.

- **Loosely Coupled:** The middleware controls the file processing by dividing and distributing through the compute pool to be computed in parallel. The processed components are then reorganized and stored.

The HPC are the main tool for running scientific and enterprise applications because the massive performance that can achieve by its given hardware. Thus, the applications used on HPC domain requires very high performance cores and floating-point processors, scalable memory, and fast I/O (GOLDWORM; SKAMAROCK, 2007).

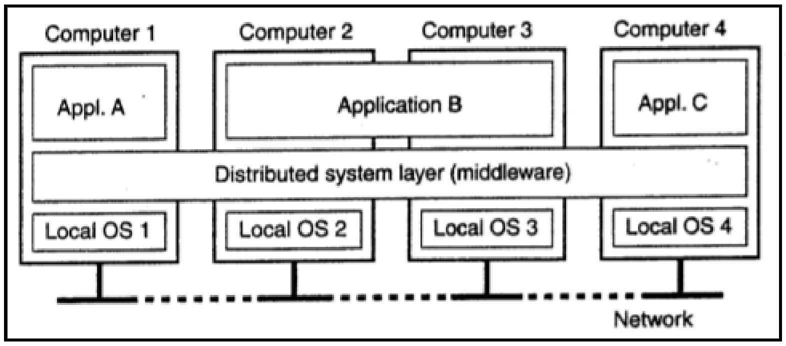
However, despite its efficiency, alternatives such as workstation clusters and cloud computing, can offer a relatively good performance at low cost, and it is easy to scale, becoming an alternative to HPC (JUVE; DEELMAN; VAHI; MEHTA; BERRIMAN; BERMAN; MAECHLING, 2009).

2.2.4 Distributed Systems

Distributed system has emerged as one of the key research areas driving innovations in business, engineering, and science, (CHANG, 2015). It was characterized as a set of independent computers which are presented to the users as a single and consistent system (TANENBAUM; STEEN, 2007).

Also, Tanenbaum and Steen (2007) emphasize that to achieve homogeneity from the users point of view, the distributed system has some important aspects that must be considered. The first, consists of the independent components (computers). The second refers to the users, being people or programs who think they are working with an individual system. The way it will be established this collaboration is the main point of distributed systems.

In cloud computing environment, the distributed system plays an important role according to Buyya et al. (2009). This is because one of the key feature of the cloud is a system that consists in a collection of interconnected computers that are dynamically provisioned, presented as one or more unified computing resources and designed to maintain availability.



Source: Extracted from Tanenbaum and Steen (2007).

Figure 2.6: Distributed System example.

In the Figure 2.6 an example of a distributed system is presented. It shows four computers and three applications. It can be seen that application B is distributed between computer 2 and 3.

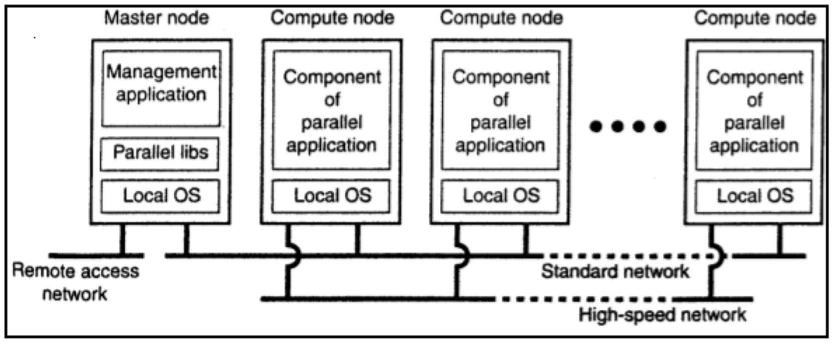
According to Marinescu (2013), the distributed system coordinates its activities by a software called middleware, which should support a set of characteristics that composes a distributed system, they are:

- **Access transparency:** Identical operations to access local or remote information.
- **Location transparency:** The information is accessed without the knowledge of their location.
- **Concurrency transparency:** Processes are executed simultaneously by sharing information without interference between them.
- **Replication transparency:** Multiples instances can replicate resources among them without user knowledge.
- **Failure transparency:** continuously available, even through a component failure.
- **Migration transparency:** The information object can be moved in real time without affect the access.

- **Performance transparency:** The performance can be determined by the quality of service requirements.
- **Scaling transparency:** System and applications can scale without changing the structure and also affect the applications.

2.2.4.1 Cluster

Cluster is a group of computer servers that are interconnected by network. It provides an economical way to achieve high performance. Thus, the performance is increased by the aggregation of more nodes, as a consequence, computational power will be maximized. It is an economical way for departments that could not afford an expensive supercomputer, have the alternative with this technology (EL-REWINI; ABD-EL-BARR, 2005).



Source: Extracted from Tanenbaum and Steen (2007).

Figure 2.7: Cluster example.

In the Figure 2.7 a beowulf cluster is shown. It consists of a group of computers nodes that are controlled by a master node.

In addition, Tanenbaum and Steen (2007) emphasize that clusters have become financially attractive to building a supercomputer using off-the-shelf technology, by connecting computers to a high-speed network. In almost all cases, clustering is used for parallel programming in which a single program (intensive computing) runs in parallel on multiple machines.

According to Buyya, Broberg and Goscinski (2010), cluster technology allowed access to large amounts of computing power, aggregating resources. The improvement of this technology, in conjunction with grid computing is one of the roots of cloud computing.

2.2.4.2 Node/Workstation

Each individual computer, is called a node. So, a node is a computer device that is interconnected and hence, forms a larger data structure. Each individual node can be a workstation, a personal computer or a multiprocessor system.

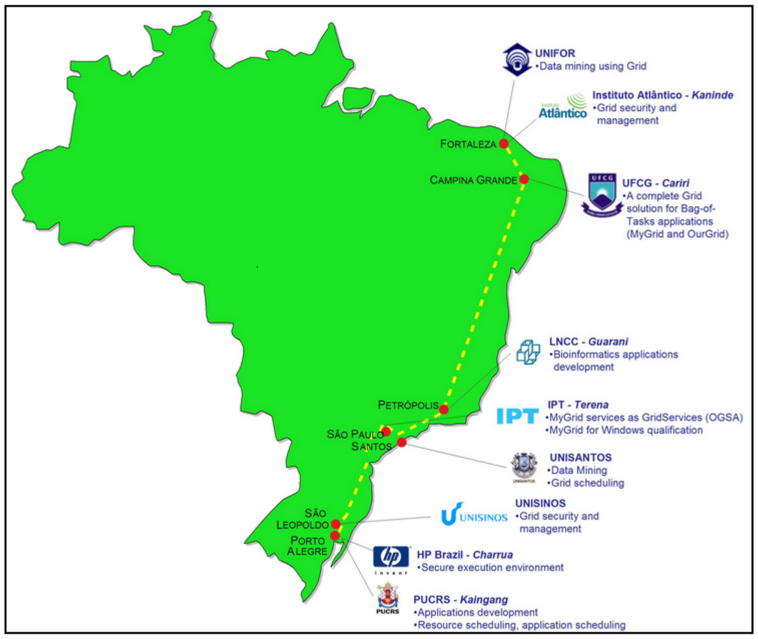
They are usually connected by a high-speed LAN and execute a software that allows them to work together. Therefore, they can engage their own activities while at the same time, can cooperate with others computational tasks, in order to solve a problem, such as scientific or engineering (EL-REWINI; ABD-EL-BARR, 2005), (TANENBAUM; AUSTIN, 2013).

According to Chang (2015), nodes can help to provide a cloud. There are several scenarios to build a desktop cloud, such as a group of universities. Therefore, they can benefit from sum of the computational resources provided by PCs, laptops and server nodes.

2.2.4.3 Grid Computing

The term grid computing emerged in the 1990s with the definition of a distributed computing infrastructure that focuses on large-scale resource sharing, innovative applications and high-performance orientation (FOSTER; KESSELMAN; TUECKE, 2001).

In addition, authors El-Rewini and Abd-El-Barr (2005) emphasized that while clusters are collections of computers linked together as a single system, a grid consists of multiple systems that work together, keeping their identities distinct. Grid resources can use distributed systems to achieve and aggregate performance as they balance the workload.


UNIFOR
•Data mining using Grid
Instituto Atlântico - *Kaninde*
•Grid security and management
FORTALEZA
CAMPINA GRANDE
UFCG - *Cariri*
•A complete Grid solution for Bag-of-Tasks applications (MyGrid and OurGrid)
LNCC - *Guarani*
•Bioinformatics applications development
PETRÓPOLIS
IPT - *Terena*
•MyGrid services as GridServices (OGSA)
•MyGrid for Windows qualification
SÃO PAULO
SANTOS
UNISANTOS
•Data Mining
•Grid scheduling
UNISINOS
•Grid security and management
SÃO LEOPOLDO
PORTO ALEGRE
HP Brazil - *Charrua*
•Secure execution environment
PUCRS - *Kaingang*
•Applications development
•Resource scheduling, application scheduling

Source: Extracted from HP (2005).

Figure 2.8: Pauá Grid Project.

An example of grid is the Pauá Grid Project (Figure 2.8). According to Wilter et al. (2005), Pauá is an initiative created by HP Brazil R & D to build a Brazilian national grid. PAUÁ currently involves 11 different universities and research centers collaborating with HP Brazil R & D in HP's "ecological research system". PAUÁ is a 250-node grid that supports the execution of Bag-of-Tasks applications whose tasks are independent. Bag-of-Tasks (BoT) applications are parallel applications whose tasks are independent of one another. Because of the independence of their tasks, BoT applications can run successfully on widely distributed computing networks.

However, in accordance to Cafaro and Aloisio (2010), cloud and grid computing are different from the customers point of view. In the cloud, you pay for the use, and in the grid, you schedule your work, regardless where you run it. They are not exclusive to each other, but you can use a grid on a cloud.

2.2.5 Virtualization

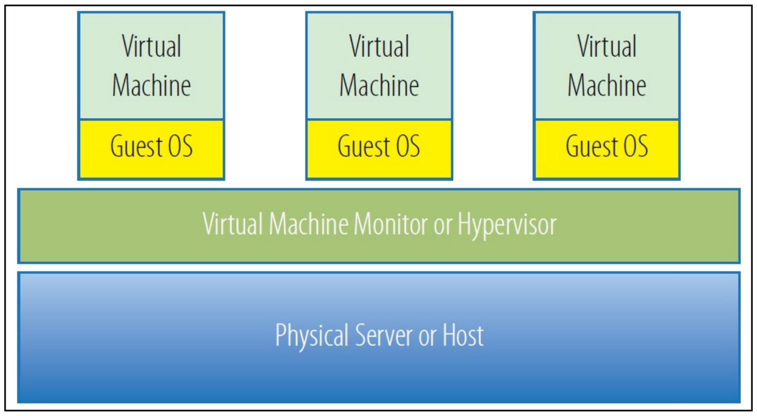
Virtualization is a technology that has changed the way of how services are provided. It makes possible to have a better use and efficiency of hardware resources, by installing multiple operating systems on different virtual machines on the same hardware. According to Portnoy (2016), it is widely used because it provides an abstraction of hardware resources (memory, disk, processors) to a virtual machine.

Pujolle (2015) emphasizes that the main idea of a virtual environments is to abstract from the user and work like a normal setup. In a common situation the OS is installed directly over the hardware. But in the most cases the system does not need all the available resources and wastes them. The virtualization works by distributing these resources to the VMs (Virtual Machines) and thus promoting a better use of the available hardware.

In Figure 2.9, the virtualization layers are shown. First, is the physical layer, shortly thereafter the hypervisor layer, guest OS layer and finally, the virtual machine layer.

Virtualization has many benefits, such as reduce power consumption, cost efficiency, isolate applications and provide a better infrastructure management. However, Pujolle (2015) highlights that the basic virtualization problem is the significant reduction of performance. In order to recover this performance, it needs a computer or even server a much more powerfull.

There are several virtualization techniques, the main ones are described in the



Source: Extracted from Portnoy (2016).

Figure 2.9: Virtualization Layers.

following sections.

2.2.5.1 Para-Virtualization

Also known as OS-assisted, para-virtualization allows the OS to have direct access to the hardware. Refers to communication between the OS and the hypervisor to improve the performance and efficiency. According to Sosinsky (2010) an abstraction occurs to place the I/O operations outside the virtual environment, thus allowing a more efficient execution of I/O devices. In a simple way it is a host operating system performing the I/O devices through a para-API.

2.2.5.2 Container Virtualization

The container-based virtualization allows a physical server to run multiple isolated operating system instances. Thus, the host operational system does not need to emulate guest system call, because they share the same kernel (MUKHEDKAR; VETTATHU; CHIRAMMAL, 2016). This is provided by an isolation of the resources.

Containers make their abstraction layer at the operating system level, rather than the hardware level. Thus, the main benefit of this approach is that it can minimize the overhead caused by the virtualization layer. Therefore, the technology that orchestrates container virtualization is promising and the difference in performance compared to other virtualization technologies, could reveal a better way to deliver and deploy applications in cloud computing.

However, the container model do have some limitations. As emphasized by Portnoy (2016), all workloads must run the same operating system or kernel because it abstracts the operating system level. Furthermore, the isolation between workloads is not as robust as what hypervisors and virtual machines provide.

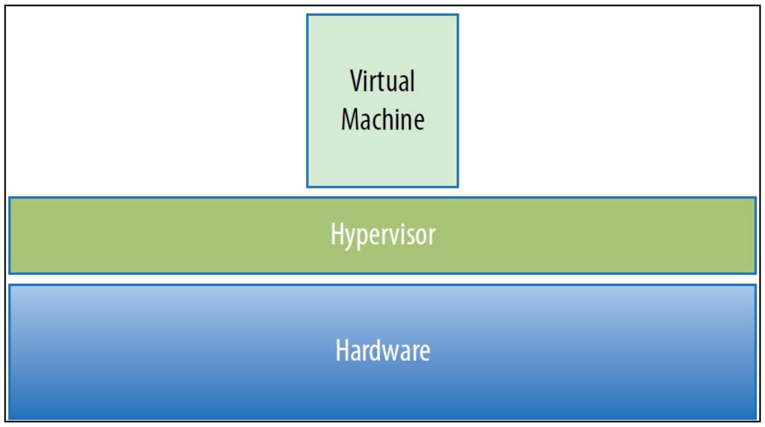
2.2.5.3 Emulated

According to Matthews et al. (2008), in this kind of platform the hardware is totally divergent from the physical architecture, the platform is entirely simulated by the hypervisor, so that the guest, where commonly executes a specific application, performs all its functions without the need for changes.

This type of virtualization is used when there is a need to test certain software under development on a specific hardware, the emulator in this case will cause an environment to be reproduced taking into account the requirements that determines the application, this environment is often completely distinct from the actual computer architecture.

2.2.6 Hypervisors

According to Portnoy (2016) a hypervisor is a layer of software located between hardware resources and a virtual machine. It provides the virtual environment to these workloads, communication between themselves and provides system clustering for high availability. On 2.10 is shown where the hypervisor layer is located.



Source: Extracted from Portnoy (2016).

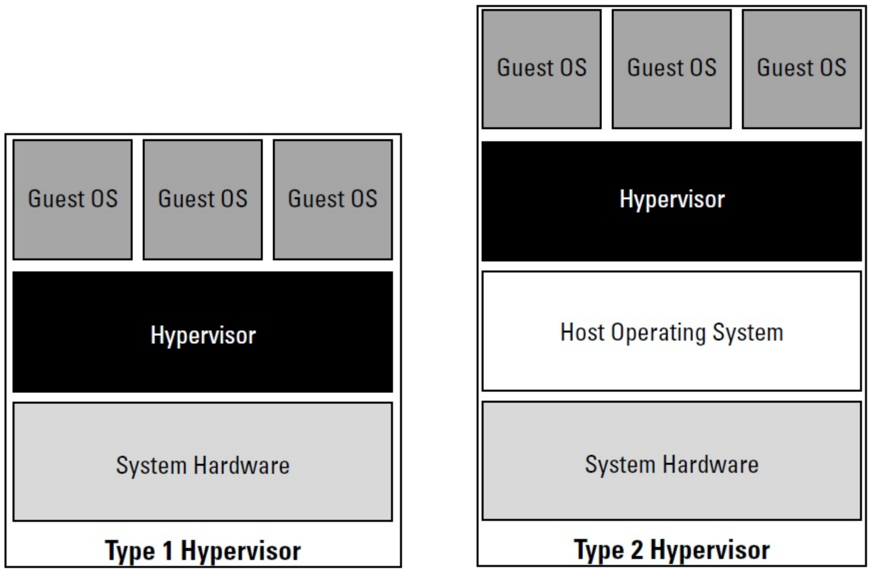
Figure 2.10: Where hypervisor resides.

In addition, as emphasized by Sosinsky (2010), hypervisors are classified into two types: type 1 VM or native VM (2.11(a)), those that have no host operating system between the hypervisor and the hardware, is a bare system implementation. Without and intermediary OS, hypervisors type 1 are much more efficient and safer. The type 2 or hosted VM (2.11(b)), is the one that has an OS between the hypervisor and the hardware, respectively. It creates a software interface to emulate the devices with which the system normally interact. On 2.11 is presented a comparison between the layers of type 1 and type 2 hypervisors.

Regarding to hypervisors, there is a large number of them. The main ones are described below.

2.2.6.1 KVM

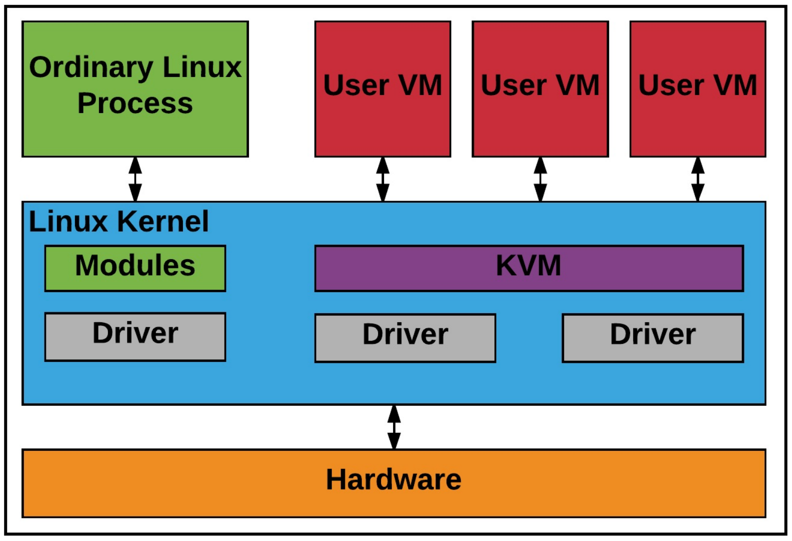
According to Mukhedkar, Vettathu and Chirammal (2016) Kernal-based Virtual Machine (KVM) is a next generation open source hypervisor integrated in Linux. Built on experience with previous generation of technologies, KVM works with modern hardware available today. When KVM kernel module is installed, it transforms the Linux kernel into a hypervisor.



(a) (b)

Source: Extracted from Sosinsky (2010).

Figure 2.11: Comparison between type 1 and typer 2 hypervisors.

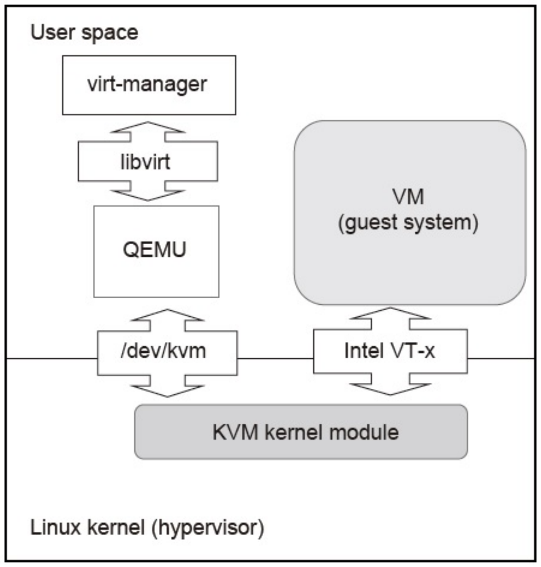


Source: Extracted from LinuxPlanet (2017).

Figure 2.12: Kernel-based Virtual Machine Structure.

In addition, the KVM has a virtualization method, which resembles a non-virtualized system as presented in Figure 2.12. Furthermore, a perceptual difference in comparison to other hypervisors is the capacity to use a little disk space, because it has a great reuse of the current hardware's resources.

In the KVM, there are components that work together to provide operations to the virtualized operating system. In the Figure 2.13 the components are shown and described below.

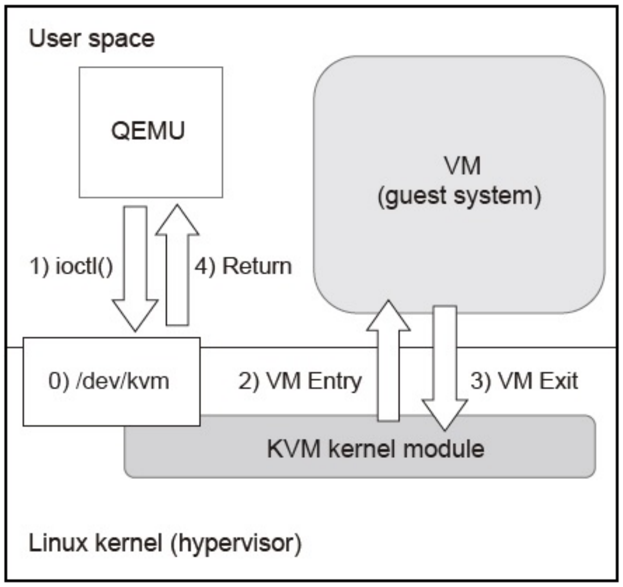


Source: Extracted from Goto (2011).

Figure 2.13: KVM Components.

- **Virt-Manager:** It is a desktop user interface for managing virtual machines through the libvirt component. Virt-Manager is used combined with the KVM. It provides a brief overview of running domains, their live performance and the uses of static resource utilizations (VIRT-MANAGER, 2017).
- **Libvirt:** Is a collection of software that provides management of virtual machines in a pleasant way and other virtualization features such as storage and network interface management. It includes an API library, a daemon (libvirtd) and a command-line utility (virsh). Its primary purpose is to provide a way to manage multiple provider or different virtualization hypervisors (LIBVIRT, 2017a).
- **QEMU:** The KVM kernel, by itself, can not create a VM. To do this, it must use QEMU, a hardware emulation and virtualization process. It can be used as a

machine emulator, running operating systems and programs made for a system built on a different machine. When used as a virtualizer, QEMU achieves near-native performance by running guest code directly on the CPU host. QEMU supports virtualization when running on Xen hypervisor or KVM (QEMU, 2017).



Source: Extracted from Goto (2011).

Figure 2.14: QEMU/KVM execution flow.

The QEMU/KVM execution flow is shown in the Figure 2.14. According to Goto (2011), a file named /dev/kvm is created by the KVM module. This file allows QEMU to transmit some requests to the KVM to perform hypervisor functions. When QEMU starts a guest system, it makes a system call called ioctl(). When it is time to start running the OS, QEMU again calls ioctl() to order the KVM to start a guest system. Kernel mode performs a VM entry and starts running the guest system. In addition, when the guest system performs a sensitive process, an VM Exit runs and KVM identifies the reason for this output. If a QEMU intervention is required, the control is transferred to the QEMU process and executes the tasks. When the execution is finished, QEMU re-runs an ioctl() request and KVM returns the guest processing.

- **KVM Kernel Module:** It is a Linux kernel module, it works with the VM exits of the guest operating systems and performs the VM input instructions (GOTO, 2011).

- **Extended Page Table (EPT):** It extends the address translation engine provided by the CPU. Prior to its development, it was necessary to perform address translation processing in software using a technique called "shadow paging".Thus, it allows the "physical addresses" used in a virtual machine to be converted by the CPU, making software conversion unnecessary. EPT plays a key role in KVM operations, resulting in a significantly improved virtual machine performance (GOTO, 2011).

- **VT-d:** According to Intel (2017a), Intel® Managed I/O Virtualization Technology (VT-d) extends Intel Virtualization Technology (VT) by providing hardware assists for virtualization solution. Intel VT-d improves the security and reliability of systems and also improves the performance of I/O devices in a virtualized environment. In addition, the author Goto (2011) emphasizes that VT-d is an address conversion mechanism for I/O devices. Therefore, it must be supported by firmware.

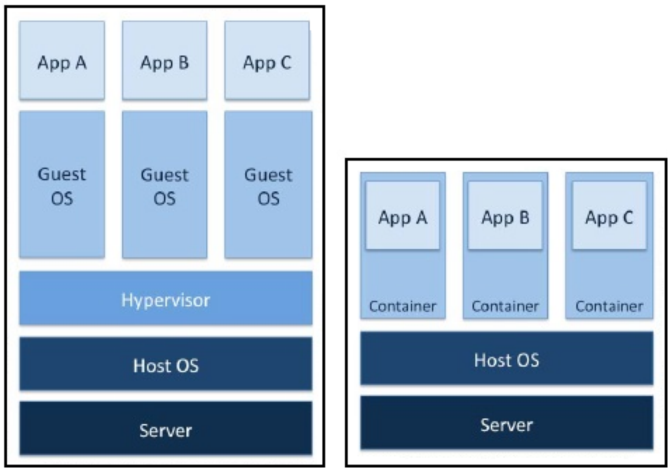
- **Virtio:** According to Goto (2011), it is a mechanism introduced to reduce the overhead associated with QEMU and improves high-speed processing. Libvirt (2017b) introduces Virtio is an virtualization standard that runs in the virtual environment and works in conjunction with the hypervisor. Thus, it provides a high performance network and disk operations.

- **Kernel Same-page Merging (KSM):** According to Goto (2011), its functionality is to monitor the memory usage of processes and then merge duplicate pages into a common page. Linux (2017) argues that KSM is a memory-saving feature and can be useful for any application that generates same data.

2.2.6.2 LXC (Linux Containers)

LXC is a container technology for Linux, which uses cgroups to control system resources and namespaces to create and isolate the objects within the container.(FELTER; FERREIRA; RAJAMONY; RUBIO, 2015). According to LXC (2017), it is a user-space interface for the linux kernel. Using powerful APIs and some tools, it provides to Linux users the creation of management systems and applications containers. Therefore, it is defined as being a lightweight virtualization, which does not require physical hardware emulation (BESERRA; MORENO; ENDO; BARRETO; SADOK; FERNANDES, 2015b).

When full-virtualization is used, it causes an overhead using a hypervisor, decreasing performance. Therefore, the main purpose of using Linux Containers is to provide resources or the same environment such as a VM, and to avoid the overhead caused by the hypervisor. LXC uses the same linux kernel, avoiding the utilization of a second kernel for virtualization and through a combination of kernel security features makes it possible to create a virtual environment on the same machine without a hypervisor (LXC, 2017).



(a) KVM. (b) LXC.

Source: Extracted from Beserra et al. (2015b).

Figure 2.15: Comparing the structure of KVM and LXC.

The comparison shown on Figure 2.15, between KVM and LXC demonstrates the layers of each one. LXC takes full advantage of the Linux kernel by isolating OS resources.

According to IBM (2017), containers can divide resources that are managed by an operating system, into isolated groups to balance demands on resource usage. Furthermore, LXC can execute native instructions to the CPU core without any special interpretation mechanism. Thus, through the use of containers, the OS gives the applications the illusion of running on separate machines while sharing many of the underlying resources.

2.2.7 Cloud Computing

Cloud computing is a model that allows a pool of resources such as, network, servers, storage, applications, and services, to be delivered ubiquitously. Cafaro and Aloisio (2010), argues that the cloud refers to moving files from the local storage to store them in secure scalable environments. Likewise, according to ISO/IEC-17788 (2014), cloud computing is a paradigm that enables network access to a scalable and elastic pool of shareable physical or virtual resources with self-service provisioning and administration on demand.

As defined by Badger et al. (2012), the cloud model has five essential characteristics, three service models, and four deployment models.

There are some characteristic to define a cloud computing, they are:

- **On-demand self-service**

  A costumer can, as it needs, configure computing capabilities and features without human interaction. Giving the option whenever necessary (BADGER; CHAPMAN; PATT-CORNER; VOAS, 2012). This on-demand self-service is usually done through an online control panel, at the hosting provider.

- **Broad network access**

  Capabilities are available over the network and are accessed through standard mechanisms that promote use by heterogeneous thin or thick client platforms (*e.g.* smart phones, tablets, laptops, and workstations) (BADGER; CHAPMAN; PATT-CORNER; VOAS, 2012). This characteristic, is the essential way of connection between the servers and clients.

- **Resource pooling**

  According to Badger et al. (2012), resources are pooled to serve multiple costumers using a multi-tenant model. The different types of physical and virtual resources are dynamically assigned as the costumer demand. The resource pooling must be able to serve the costumer with high level of abstraction, regardless of the physical location of the resources (country, state or datacenter).

- **Rapid elasticity**

  Cloud computing must provide the capacity for the costumer to expand they computational power, whether automatically or not. This technical approach should be rapid and without human interaction. Therefore, it avoid the lack of resources. In according with Badger et al. (2012), the capabilities available for provisioning often appear to be unlimited and can be appropriated in any quantity at any time.

- **Measured service**

  Cloud system allow the resources to be monitored, controlled, and reported, providing transparency for both the provider and consumer of the utilized service (BADGER; CHAPMAN; PATT-CORNER; VOAS, 2012). These characteristics, are essential in the model as a service, providing both the user and the provider, to administrate the contracted plan and the resources used.

2.2.7.1 Public Cloud

According to Ruparelia (2016a), public cloud is a deployment model and, as the name suggests, it is available to the public as a whole or also to organizations.

The company's provider of this service is responsible to share its infrastructure for consumers use. Typically, in the public cloud resources are provided such as; infrastructure; platform; software; information; and the most know, storage. Examples of community cloud include Google Docs, Microsoft Office 365 and Amazon EC2. In according with Sosinsky (2010), public cloud are services sold by large organizations that have cloud computing services, for ordinary users and industries.

2.2.7.2 Hybrid Cloud

The hybrid cloud is a composition of several types of other clouds, including private, public and community clouds. The author Sosinsky (2010) emphasizes that a hybrid cloud is an amount of clouds with their unique identities, working together as a unite.

In addition, as mentioned by the author Ruparelia (2016a), hybrid cloud can also be known by cloud bursting. This name is used because services bursts out of its cloud for use resources from other clouds.

2.2.7.3 Private Cloud

A private cloud refers to a model of deployment of the cloud done in-house. In contrast to the public cloud, the private cloud is typically used by companies that want storage data on their internal infrastructure. Thus, as defined by Kavis (2014), the cloud infrastructure is provisioned for exclusive use by a single organization that can be owned, managed, and operated by the organization.

The private cloud infrastructure is operated exclusively for an organization. In this case, the private cloud is an emulation of the public cloud, usually on a private network, and exist to support the goals of an organization. Therefore, it gives more control over the security architecture, and has less exposure (SMOOT; TAN, 2011).

2.2.7.4 Community Cloud

This cloud infrastructure is a hybrid cloud for a group of specific individuals or organizations that shares a common goal, and work together on joint projects. It can be managed by the organizations or a third parties in a central cloud computing (SMOOT; TAN, 2011).

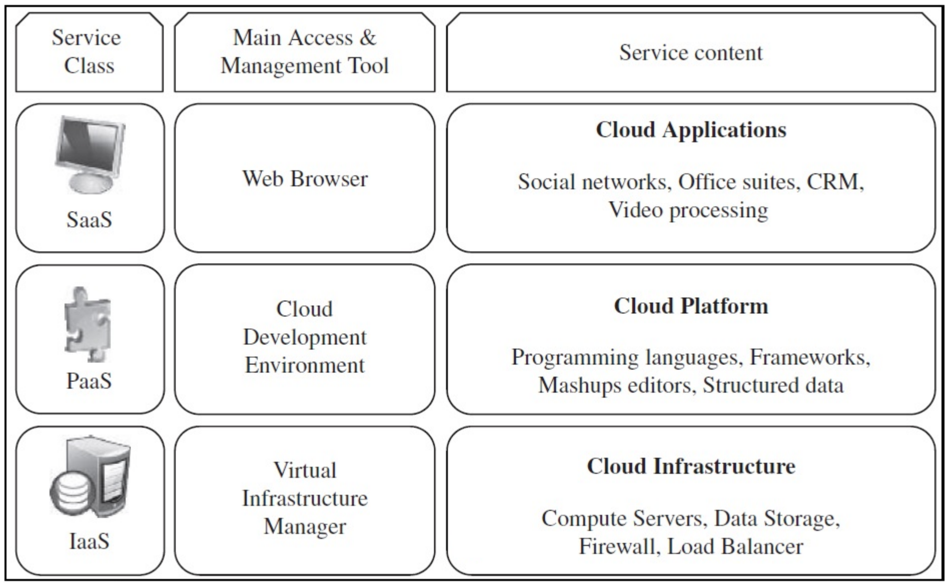
2.2.8 Service Models

Acordding to Badger et al. (2012) cloud computing is composed by three main service models. Also known by SPI (SaaS, PaaS and IaaS), they are described in the next sections.

2.2.8.1 IaaS (Infrastructure as a Service)

Infrastructure as a service, as defined by Badger et al. (2012), is a capability provided to the consumer to alocate processing, storage, networking, and other computing resources, allowing the deployment of operating systems and applications. Thus, the service consumer, does not manage the underlying cloud infrastructure, only the service provider. Therefore, it provides an abstraction layer of the infrastructure.

According to Kavis (2014), each cloud service model provides abstraction and automation levels for these tasks, providing more agility for cloud service consumers, so they can focus more time on their business problems and less time to manage infrastructure. Thus, the provider can provide the resources that the consumer needs with


Service Class
Main Access & Management Tool
Service content
SaaS
Web Browser
Cloud Applications
Social networks, Office suites, CRM, Video processing
PaaS
Cloud Development Environment
Cloud Platform
Programming languages, Frameworks, Mashups editors, Structured data
IaaS
Virtual Infrastructure Manager
Cloud Infrastructure
Compute Servers, Data Storage, Firewall, Load Balancer

Source: Extracted from Buyya, Broberg and Goscinski (2010).

Figure 2.16: Service Models.

limited control of network components, such as firewalls and load-balance (SMOOT; TAN, 2011).

2.2.8.2 PaaS (Platform as a Service)

Badger et al. (2012) defines Platform as a Service, as a capability provided to consumers, to deploy supported applications on the cloud infrastructure. This consumer does not manage the resources of the cloud infrastructure, but has the control of its application deployed there. The author Buyya, Broberg and Goscinski (2010), defines PaaS, as an environment where developers can create and deploy their application, without knowing the required use of the hardware.

In addition, Sosinsky (2010) emphasizes, that PaaS provides VMs, OSs, applications, services, deployment frameworks, transactions and control structures. Furthermore, the service provider is responsible for managing the cloud infrastructure and the client for the applications that is deployed.

2.2.8.3 SaaS (Sofware as a Service)

Badger et al. (2012) defines Software as a Service, as a capability provided to the consumers, to use an application, running from a cloud infrastructure. This application is available on several client devices. Also is defined as a new model of delivering applications. The author Ruparelia (2016a) emphasized some important advantages of SaaS; use only the application or applications needed; use only when necessary; and avoid to pay the installing and maintaining the application and its supporting hardware infrastructure.

In addition, according to Buyya, Broberg and Goscinski (2010), the utilization of SaaS reduces considerably the maintenance of software, and also, simplifies the development and testing for providers.

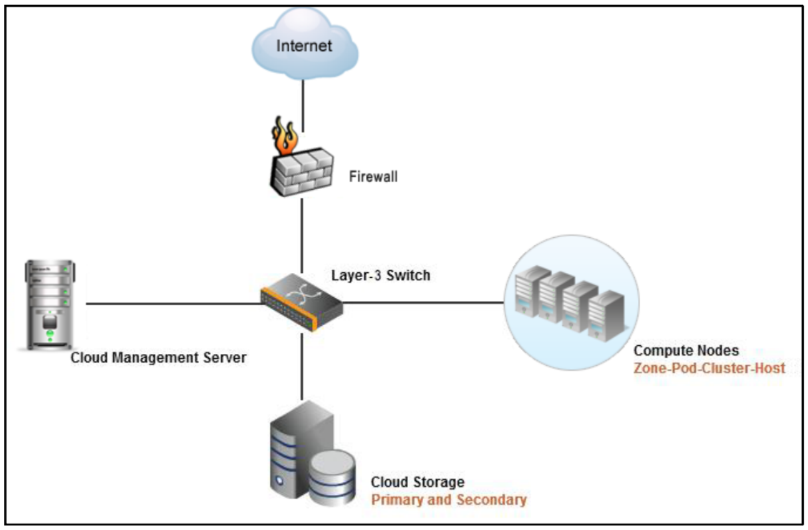
2.2.9 Open Source IaaS Tools

The open source IaaS Tools, is a platform that allows the deliver of infrastructure service by the provider. Additionally, according to Shrivastwa and Sarat (2015), in the cloud environment, the most important component is the orchestrator, who is responsible for delivering the hardware and software resources in aggregate manner, automating the work-flows required to deliver a service.

Some of the most important open-source orchestrator in the cloud community are: Apache CloudStack; OpenStack; Open Nebula. These open source projects, have a growing community and advantage of sharing knowledge. Therefore, many events are performed around the world and supported by a large numbers of sponsors.

2.2.9.1 *Apache CloudStack*

According to Chandrasekaran (2014), Apache CloudStack[2] is an open source software for building public and private clouds, developed and supported by a huge community around the globe as well as backed by some of the leading companies interested in cloud computing area. The main goal of the project is offer scaling out computational resources through an environment capable to offer IT infrastructure by managing the available computational resources. Thus, the CloudStack can provide a high availability cloud computing infrastructure.



Source: Extracted from Sabharwal (2013).

Figure 2.17: CloudStack modules.

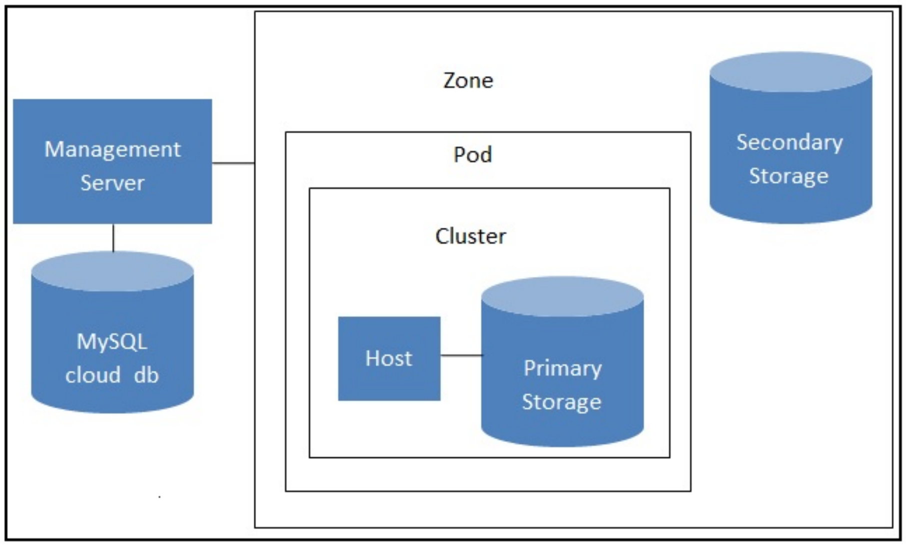
The Figure 2.17 provides an overview of the architecture used by CloudStack. These modules have an important role by managing the distribution of resources and segregation of them, providing the resources needed to build a public, private or hybrid cloud which are capable to manage thousands of physical servers (CLOUDSTACK, 2017a).

The CloudStack main architecture is based on layers which provides the com-

[2]https://cloudstack.apache.org/

ponents necessary to deploy and manage the cloud. This infrastructure layer comprises all the hardware resources available to provide storage, network and security. Thus, these resources are utilized by the hypervisor layer enabling the resource pooling and multi-tenancy between multiple machines (SABHARWAL, 2013).

On the other hand, the management layer provides capabilities to administrate the cloud environment. This layer, as emphasized by Sabharwal (2013), implements APIs, which allows automation, orchestration, task execution and service management helping the integration and interaction among various pieces of software's, which are a critical point between the IT organization and its infrastructure. Consequently, the deployment of the CloudStack consist on a hierarchical structure (Figure 2.18) that will manage the underlying infrastructure.



Source: Extracted from Sabharwal (2013).

Figure 2.18: CloudStack Architecture.

In accordance with CloudStack (2017b), these components showed on Figure 2.18 provide the following features:

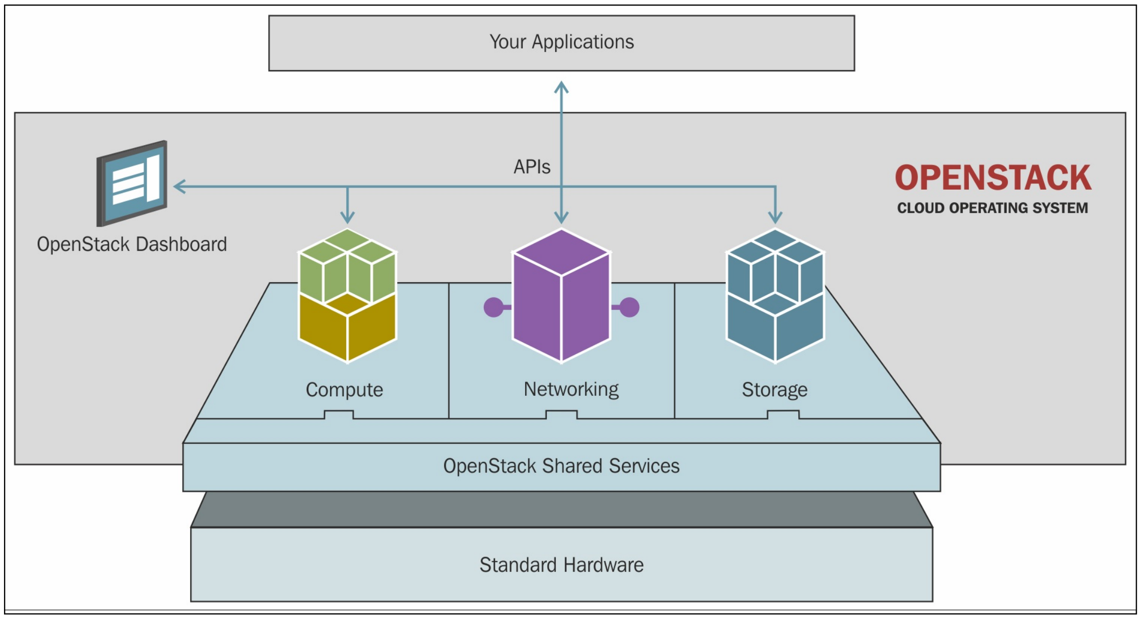
- Zone: Logical division that can represent an entire datacenter of an organization, providing physical isolation and redundancy.

- Pod: Can be compared to a physical rack in a datacenter. Contains a single or various clusters where resides the hosts. CloudStack uses the Pod to manage the subnet of the system VMs logically for administrative purposes.
- Clusters: It's a logical group of hosts contained within pods. They all also have the same hypervisor, hardware, and share the same primary storage.
- Hosts: Provides the computing resources to run guest virtual machines. Each host has hypervisor software installed on it to manage the guest VMs.
- Primary Storage: Provides virtual disks for the VMs on running on the hosts.
- Secondary Storage: Responsible to store templates, ISO images and disk snapshots. The secondary storage can share the data to all hosts throughout the cloud if configured to do that.
- Management Server: The central point of administration which provides functionalities to manage the server. It comprehends the web user interface; APIs; creation of VMs; LAN configuration; storage; orchestration.
- CloudDB: Stores all the configuration information, such as, computer offer, hosts profiles, accounts credential, network information.
- Network Offering: Comprehends a group of network services, such as, DHCP, DNS, VPN, Firewall, and port forwarding.

Apache CloudStack can aggregate computational resources to the cloud including new nodes by installing the CloudStack agent on these nodes. These agents are responsible to communicate with the CloudStack management service in order to control all the instances on the host (CLOUDSTACK, 2017c). Additionally, CloudStack also enables integration with AWD public cloud, by using his APIs services, providing an environment able to support a hybrid cloud. Therefore, according to Chandrasekaran (2014), CloudStack includes almost all the features that most organizations expect from an IaaS cloud.

2.2.9.2 OpenStack

OpenStack is an open source software for creating private and public clouds [3], designed to deliver scale-out cloud environments, deployed in datacenters around the world (JACKSON; BUNCH; SIGLER, 2015). Thus, it controls a large set of computational resources, providing a heterogeneous infrastructure allowing the IT administrator manage the cloud (Figure 2.19).



Source: Extracted from OpenStack (2017a).

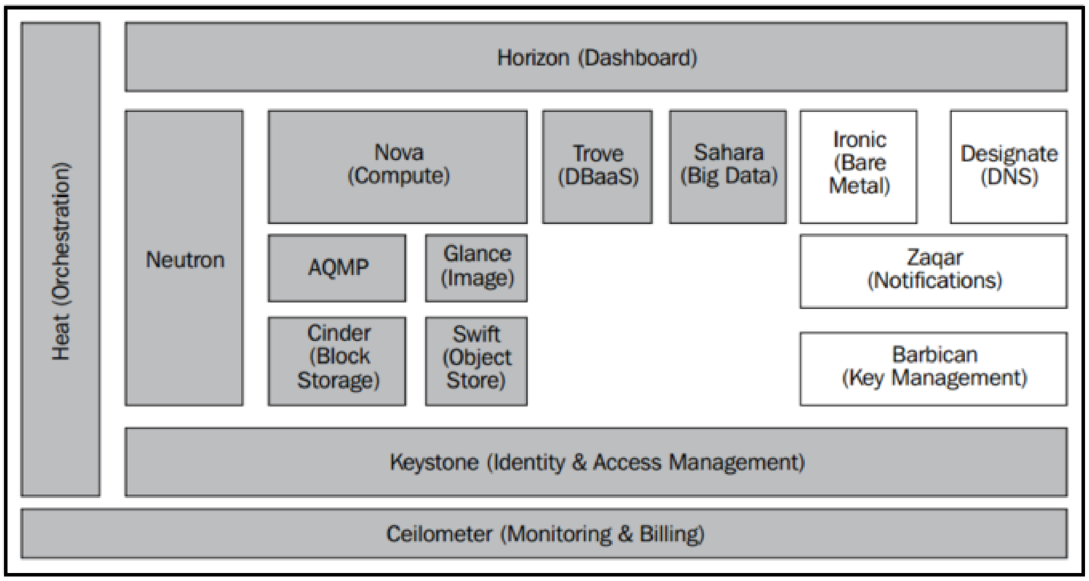
Figure 2.19: Openstack Software Diagram.

The OpenStack project are currently supported by 643 companies (including NASA and Rackspace), in more than 187 countries and has global developer collaboration, becoming one of the biggest names in cloud computing. Part of this global success, is that OpenStack is designed for ownership and control a huge infrastructure that can comprehend the public, private or hybrid cloud. Delivering service models such as IaaS, SaaS and PaaS. In Addition, OpenStack supports the ARM and x86 architecture, which gives more comprehensiveness in terms of platform support.

[3]https://www.openstack.org/

According to Shrivastwa and Sarat (2015), one of the main features of OpenStack is that it is not a single product, but a collection of multiple open source projects, which consists of several independent parts. Thus, it provides services to the administrator, such as VM provisioning, database and image storage. Additionally, OpenStack has its own identity key and access management, business-to-business toolkits, standardized services and Amazon EC2 compatibility. Therefore, it makes the OpenStack a flexible cloud provider.

The OpenStack architecture has some keys components that composes this particular cloud environment. These individual services interact with each other through public APIs and, for communication between the processes of one service, a protocol is used for message-orientated middleware (AMQP).



Source: Extracted from OpenStack (2017a).

Figure 2.20: OpenStack Architecture.

As mentioned by Shrivastwa and Sarat (2015), the core components with the gray box in Figure 2.20 are the minimum required to run OpenStack. As a characteristic, each one has its own database and can run independently, but has dependencies among them.

- Keystone: Provides authentication for others OpenStack components, being the first to be installed. Thus, it manages access to services such as identity, resource, assignment, token, catalog and policy.
- Horizon: Provides the web-based user interface to administrate OpenStack via dashboard.
- Nova: Computing component that supports hypervisors for virtual machines. It also has some subcomponents responsible for managing access to a virtual machine console.
- Glance: Allow storage of current images, templates and snapshots for deployment. Storing them on swift.
- Swift: Is the OpenStack object storage service. It is responsible to store and retrieve the Binary Large Object (BLOBs), used to store Glance images.
- Cinder: Manages the block storage from the Nova VMs, providing the use of various storage systems. Neutron: Provides network service and other features such as firewall and load balancing.
- Heat: Is the orchestrator service, responsible to coordinate and integrate the others services.
- Ceilometer: Its components are responsible to collect metering data. This is specially used on the model pay-as-you-go.
- Sahara: Provides a data-intensive application cluster which is useful on big data.
- Designate: Provides DNS service, enables the use of PowerDNS, BIND, NSD, and DynECT.
- Ironic: It is an API for bare metal technologies, which interacts with bare metal hypervisors using PXE boot and the Intelligent Plataform Management Interface (IPMI).

- Zaqar: Provides a messaging API for notification service.
- Barbican: REST API designed for the secure storage. It manages various types of secret data, such as passwords, encryption keys and certificates.
- Manila: It is a shared file system service based on Cinder. Manila provides coordinated access to shared or distributed file systems using a file-based storage (OPENSTACK, 2017b). It can be used to mount a single file system on multiple Nova instances.
- Murano: Is an application catalog, allowing the applications developers to publish cloud-ready applications in a categorized catalog through the dashboard.
- Magnum: Focused on Linux Containers, Magnum uses the Heat orchestration to improve and make a container orchestration more efficient.
- Kolla: Also focused on containers, provides production-ready containers, allowing the execution of OpenStack services in containers, facilitating governance.
- Congress: Provide Police as a Service, which offers governance and compliance in others cloud services. It helps to specificity more restricted politics to the cloud environment.

2.2.9.3 OpenNebula

According to Chandrasekaran (2014), OpenNebula is a cloud solution that helps virtualized data centers oversee private, public and hybrid clouds. Also, it is a flexible tool to provide storage, network and virtualization technologies, allowing dynamic deployment of services on distributed infrastructures.

OpenNebula[4] was created as a research project by the Distributed Systems Architecture (DSA) Research Group [5] at Madrid in 2005, and later the first public ver-

[4]https://opennebula.org/
[5]http://dsa-research.org

sion was released in 2008. It evolved and became completely open source, being frequently updated by the community. The differential of OpenNebula, is that it guarantees users full interoperability with the existing infrastructure components available today. Since 2010, OpenNebula has obtained commercial support from C12G Labs, giving a boost to the vitality of the project (TORALDO, 2012).

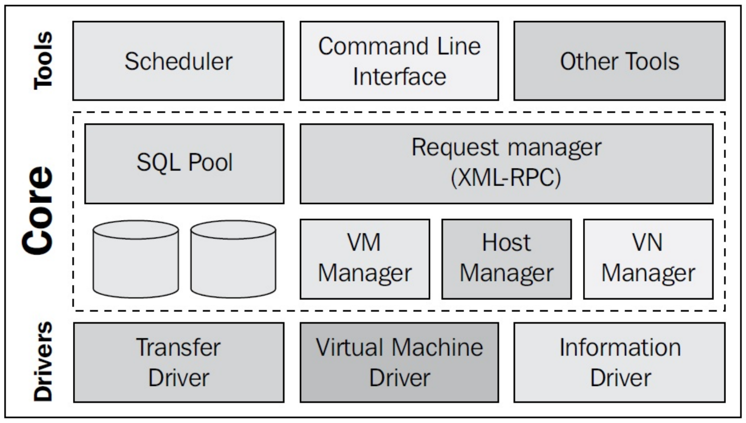
Its deployment does not require a particular hypervisor, being the choice of this free. It also has no specific infrastructure requirements, fitting in any pre-existing environment, storage or network. OpenNebula has a dedicated Quality Assurance (QA) team, to perform tests with a large number of scenarios. This considerable reduces the number of bugs. There is also a continuous integration system, which automatically tests every change in OpenNebula code realized by its development team. The Figure 2.21 shows the components of OpenNebula and also the level at which they operate.

Toraldo (2012) emphasized that security is another point taken seriously in OpenNebula, which performs all communications between hosts through security connections protected by SSH (Secure Shell) RSA keypairs and SSL (Security Socket Layer). In addition, each virtual network is protected with a firewall, known as ebtables [6].

According to Toraldo (2012), the lowest level is composed by drivers, which talk directly to the OS components. They are divided into three types, described bellow:

- **Transfer drivers:** Works with disk image management on the storage system.
- **Virtual Machine drivers:** Works with managing virtual machine instances on hosts. These drivers are hypervisor-specific.
- **Information drivers:** Remotely executed through SSH, these drivers are used

[6]http://ebtables.sourceforge.net



Source: Extracted from Toraldo (2012).

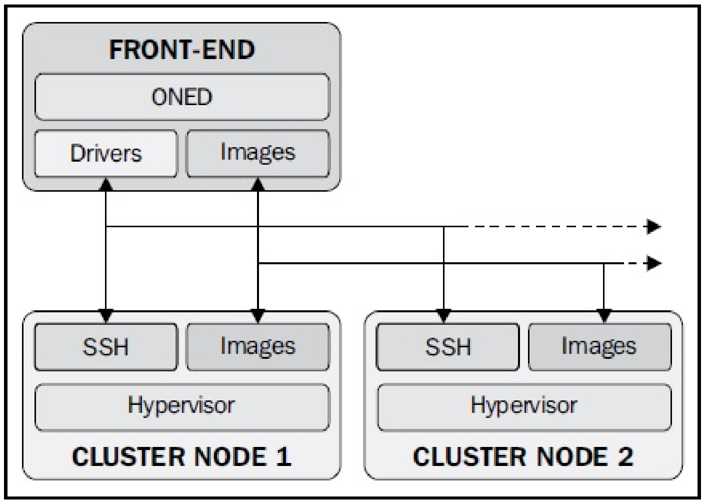
Figure 2.21: OpenNebula Components.

to retrieve the current status of VMs instances and hosts.

The middle level known by Core, is developed in full optimized C++, providing a good scalability and robustness. According to Sotomayor et al. (2008), the OpenNebula core manages the lyfe-cicle of the VMs performing basic operations, such as deployment, monitoring, migration or termination. In addition, all information collected from the drivers is stored in a SQLite database or in a replicated MySQL database in the OpenNebula core, which can be modified by custom scripts or softwares. The latter, called Tool level, is the closest to the end user. It represents the interactions by command line and tasks scheduling (TORALDO, 2012).

Because OpenNebula is deployed based on a cluster model, it uses a frontend that runs the main services of the tool and manages the rest of the nodes that are the slaves. The Figure 2.22 is a diagram which represents an OpenNebula system and below it, the description of each component:

- **Frontend:** The machine that runs OpenNebula services is called frontend. It manages the nodes and must be able to communicate with all hosts and access the network storage mounts.



Source: Extracted from Toraldo (2012).

Figure 2.22: OpenNebula System.

- **Hosts:** Hosts are physical machines that will run the virtual machines. They are managed through SSH from the frontend machine.
- **Image Repository:** It maintains the VM images. Hence, it must be accessible through the frontend, using any storage technology such as NAS, SAN (Storage Area Network) or any GNU/Linux distributed network-filesystem. In addition it should be large enough for the storage of all VMs images in its infrastructure.
- **Physical Network:** It provides an easily adaptable and customizable network subsystem, in order to better integrate with specific network requirements.

2.2.10 IaaS Tools Comparison

Cloud computing architecture is a complex environment composed of many technologies that integrate and manage various types of components, providing a stable structure without comprising performance (CHANDRASEKARAN, 2014), (HURWITZ; BLOOR; KAUFMAN; HALPER, 2010). Frequently, a question is asked “what cloud platform is best?”. Unfortunately, this question does not have a simple answer because many technologies are involved and different performance results may vary depending on the deployment made. Therefore, the most appropriate approach is to analyze the technologies and tools supported by the cloud platform and analyze what best fits the needs of the user.

For a better understanding, we use a similar approach made by Vogel (2015), which analyzes the robustness (flexibility and resilience) of the most well-known cloud computing platforms. In addition, this method provides an overview, and also helps to conceptualize and characterize many components and technologies presented in the studied cloud computing platforms.

2.2.10.1 Resiliency

Support	**OpenNebula**	**OpenStack**	**CloudStack**
Virtualization Technology	Xen, KVM, Vmware, Hyper-VirtualBox, OpenVZ LXC/LXD, ONEDock	Bare Metal (via IPMI), Hyper-V, KVM, vSphere (via vCenter), Xen, Oracle VM 3.0+, VMware, LXC	Bare Metal, Hyper-V, VMware, Xen, KVM, QEMU, UML, OpenVZ, LXC/LXD, Nova-Docker
Storage Technology	NFS, SSH(transfer), Ceph, iSCSI - Libvirt Datastore, VLM, vCenter, DatastoreStorage DRS	NFS, SMB/CIFS, Ceph RBDSAN, iSCSI, Scale Computing	LVM, Ceph RBD, GlusterFS, NFS, Sheepdog, SambaFS, Block Bridge EPS, iSCSI, Software-Defined Storage, Fiber channel
Virtual Disk Formats	QCOW2, LVM, Ceph (Shared FS), Raw Device, Mapping((RDM) Datastore), VMDK, VMFS	LVM, VMDK, VHD, QCOW2, VMFS, ISO, OVA	QCOW2, RAW, VHD, VMDK, VDI, VHDX, AKI, AMI, ARI, ISO, PLOOP, BARE, OVF, OVA, AUFS
Network	vCenterBridged(dummy, Security Groups, ebtables), VLAN, VXLAN, Open vSwitch, IPAM	NVP, VSP, MidoNet, VXLAN, Open vSwitch, AutoScale	Neutron (NaaS), Open vSwitch, NVP, OpenFlow
Operating System	Ubuntu, Debian, RedHat, SUSE, CentOS, Windows (>=7)Devuan, VyOS, gUSE, CloudBroker, Wrapper, CoreOS alpha, FreeBSD 10.3	XenServer, Ubuntu, RHEL, CentOS, Windows, FreeBSD, Fedora	Debian, Ubuntu, RHEL, CentOS, Fedora, Suse, KVM for IBM z Systems, Oracle Linux, SLES, Windows
Storage (Block Storage / Object Storage)	Ceph RBD (Block Storage)	Ceph RBD (Block Storage), Swift (Object Storage)	Cinder (Block Storage), Swift (Object Storage), Ceph RBD (Block Storage)

Source: Updated from Vogel et al. (2016).

Table 2.3: IaaS tools support for resiliency.

The table 2.3 is based on the previous study by Vogel et al. (2016). A deep investigation was conducted using the official documentations of the tools as reference, to fill the table with the new technologies developed and resources brought by the analyzed platforms. Therefore, modifications were made to update the information and categorize according to the context of this study.

The table comprises the main current technologies supported by the three open source IaaS tools used in our study. These technologies form the infrastructure elements that composes the cloud environment and also support for the resilience, which is the ability to adopt the system to constant changes or failures, while maintains the availability (VOGEL; GRIEBLER; MARON; SCHEPKE; FERNANDES, 2016). Furthermore, according to Prodan and Ostermann (2009) and by Dukaric and Juric (2013), these elements are particularly critical in cloud computing because different configurations can be made, and as a consequence, application performance may vary depending on the deployment performed.

- **Virtualization Technology:** refers to hypervisors and container-based, supported by the tools used to emulate hardware resources used by the instances. Consequently, different hypervisors may form different based clouds, for example, KVM-based cloud or LXC-based cloud.
- **Storage Technology:** Comprehends the technology and architectures used to provide storage solutions for the tools.
- **Virtual Disk Formats:** It is used by virtualization technology to provide storage functionality and enable the provisioning, migration and maintenance of virtual machine across different platforms (VMWARE, 2008).
- **Network:** Refers to the component responsible for providing network resources such as DHCP, VLAN and Iptables. It is appointed by Dukaric and Juric (2013) as being a complex component for cloud computing because it is not only responsible to manipulating the entire network, but used to connect virtual machine instances.

- **Operating System:** It is a list of guest OSs supported by VMs. Only major distributions are listed, but it should be emphasized that in the case of OpenStack, basically all the Linux distributions are supported, provided they meet the minimum requirements.

- **Storage (Block Storage / Object Storage):** Different technologies used to provide data storage that are capable of handling large amounts of data (MUKHEDKAR; VETTATHU; CHIRAMMAL, 2016).

The main conclusion in comparison with the study conducted by Vogel et al. (2016), is that many more technologies are currently supported by IaaS tools in relation to resilience. Through this, we can assume that the IaaS cloud tools, along with the support of companies and the community, have made some huge improvements to deliver new technologies, as well as adding new features. Foremost, the main difference we can notice is the introduction of new storage technologies and virtual disks formats. This reveals an effort not only to make the open source tools more attractive but also to make the tools intercompatible, as is the case with the introduction of the OVF (Open Virtual Format) disk formats, which is a technology that can provide portability between clouds, both public and private. In addition, is the introduction of support several new operating systems that we did not have before, moving from some restricted OS families to all families, having only restricted OS hardware requirements. This allow us to conclude that the cloud platforms will continue to improve their compatibility between public and private tools as well as support from companies that develop and deliver new technologies, such as Dell, IBM and SolidFire. Thus, we will have an even more rich, flexible and intercompatible environment.

2.2.10.2 Flexibility

The Table 2.4 represents the detailed description of the main components that are present in the architecture of cloud management tools. The terminology and defini-

Support	OpenNebula	CloudStack	OpenStack
Resource abstraction layer			
Compute	Oned	Libcloud	Nova
Storage	Internal	Internal	Object storage (Swift) /Block Storage(Cinder)
Volume	Internal	Internal	Nova-Volume
Network	Virtual Network Manager	Internal	Neutron/Nova-network
Core service layer			
Identity service	Internal	IAM plugin	Keystone
Scheduling	Scheduler	Internal	Nova-scheduler
Image repository	Internal	Internal	Glance
Charging and billing	Showback	CloudStack Usage	Ceilometer
Logging	Internal	Internal	Internal
Support layer			
Message bus	Internal	internal/RabbitMQ	RabbitMQ
Database	sqlite/MySQL	MySQL	MySQL/Galera/MariaDB/MongoDB
Transfer service	Internal	Internal	Nova Object store/cinder
Management layer			
Resource management	Internal	Internal	Nova
Federation management	Internal	/	Federated Keystone
Elasticity management	Auto-scaling	Elastic Load Balancing	Elastic Recheck
User/group management	Internal	Internal	Internal
SLA definition	External	/	External
Monitoring	probe/ssh/OneGate	External	Monasca/ Telemetry
Reporting	code reporting	/	Monasca/ Telemetry
Incident management	Internal	Internal	Monasca/ Telemetry
Power management	External	External	Blueprint driver
Lease management	External	External	Blazar
Management tools			
CLI tools	OpenNebula CLI	cloudmonkey	OpenStack (CLI)
APIs	Public cloud and Plugins	Public cloud and Plugins	Public cloud and Plugins
Dashboard	Sunstone (Admin UI, User UI)	Admin UI	Horizon(Admin UI)
Orchestrator	oneflow	/	heat
Security layer			
Authentication	Basic Auth/OpenNebula Auth/ x509Auth/LDAP	SAML/LDAP	LDAP/Tokens(APIs)/X.509/ HTTPD/ Kerberos
Authorization	Auth driver	SAML	Keystone
Security groups	Internal	Internal	Internal
Single sign-on	/	SAML	SAML
Security monitoring	External	External	External
Control layer			
SLA enforcement	/	/	/
SLA monitoring	/	/	/
Metering	External	External	ceilometer
Policy control	/	(IAM) Plugin	DynamicPolicies
Notification service	/	Internal	/
Orchestration	OneFlow	Internal	heat
Value-added services			
Availability zones	Internal	Internal	Internal
High Availability	External	External	External
Hybrid support	Amazon EC2/Microsoft Azure/ IBMSoftLayer	Amazon EC2	HP Helion/Amazon EC2/ IBM/Microsoft Azure
Live migration	Internal	Internal	Internal
Portability support	Internal	/	/
Image contextualization	one-context	/	/
Virtual application support	/	/	/

Source: Updated from Vogel et al. (2016) and Dukaric and Juric (2013).

Table 2.4: IaaS tools support for flexibility.

tions are based on previous studies made by Dukaric and Juric (2013) and Vogel et al. (2016). A thorough investigation was performed analyzing the documentation of cloud platform tools, in official web-sites and documentations, making possible to update the information. Thus, these changes represent the latest implementations and capabilities developed by the cloud management tools studied, to introduce new components and consequently, more flexibility for users.

In addition, as mentioned in previous studies, the "/" character is used to represent a resource that is not supported or the status is unknown. We also focus on keeping existing components (blue color), adding the new components (red color). On the other hand, the word "internal" is used to mean that the corresponding resource is supported by the tool, but it is unknown which present component is responsible to manage it. Finally, the word "external" is used when the tool requires a third-part solution to implement the component.

Resource abstraction layers, according to Dukaric and Juric (2013), comprise the components most crucial to cloud architecture. First, the compute is responsible for managing virtual machines (create, terminate and reboot). The storage component is responsible for providing object storage to the cloud and also for uploading and downloading VM images in a scalable and redundant way. On the other hand, the volume component provides persistent block storage volumes (data is not lost when the instance is rebooted or disconnected), volumes are managed by the instances, in most cases using Network File System (NFS) and Internet Small Computer System Interface (iSCSI). Finally, the network provides the ability to manage the network infrastructure by offering virtual networks, firewalls, DHCP to other components and also the VMs.

Core service layer, is also another important component, which provides provisioning, billing and connection service (BUYYA; VECCHIOLA; SELVI, 2013). The identity service is in charge to manage the authentication process for users among the cloud components (compute, storage and image repository). The scheduling component is responsible to verify the resources available to manage the requests made by VM instances, ensuring the efficient use of physical resources. The image repository gives to the user with a catalog that provides virtual disk images required to instantiate the VMs. The most frequent disks formats used are QCOW2, RAW and VHD. However, different formats of disks can affect instance performance, and also introduce some compatibility issues. Charging and billing are used with registration events to create and present the customer billing information based on the resources used. The log is used for audit purposes, and must be able to track all cloud operations (security and performance).

The support layer provides the means for other layers to communicate and interact (DUKARIC; JURIC, 2013). The message bus acts as a central hub used to coordinate pass-through messages between different cloud services. The Database stores all the configuration of the cloud infrastructure, such as VMs, networks, security, users, etc. Transfer services, as the named suggests, are responsible for transferring the files among the clusters and the images repository, aiming for the correct deployment of VMs.

The Management layer specifies the APIs and tools to interact with underlying services and resources, which are responsible to orchestrate the infrastructure and brings the highest degrees of agility to the entire architecture (CHANG, 2015). Thus, resource management is responsible for creating, allocating, suspending and terminating the VM, managing resources relative to physical hosts. Federation administration has the challenge and task of reaching and unifying the federation service in existing cloud services using single sign-on (SSO) mechanisms. Elastic management provides automatic and dynamic provisioning of resources for the user, increasing according to their needs with respect to defined policies. User/group management is used to add or create members with a defined policy and rules for a particular project, it is a way to define authorizations for a user or group. SLA definition corresponds to the defined QoS, it is essentially a contract that guarantees services delivered between the consumer and a cloud provider. The monitoring component is used to monitor de use of computational resources by collecting relevant metrics, being an important tool to also verify the need of investment, some cloud management tools have their own monitoring components, but other monitoring tools, such as Nagios or Ganglia. The reporting component generate the reports necessary for billing, through detailed usage of capacity based on system metrics. Incident management addresses the reestablishment of a partial or complete unavailability of resources offered by the cloud, focused on IT performance. The power management allows administrators to monitor different levels of power consumption, and also define policies to make rational use. It is an important role, because consumption will direct the impact on the return of investment, and consequently the cost savings. Lease management enables the user to request and contract resources for a certain period of time.

Management tools provide the capability to interact with underlying services and tools to manage the cloud (DUKARIC; JURIC, 2013). The command-line interface (CLI) gives the administrator a capability to manage the entire cloud environment with command-line tools, such as shell-based access. The APIs allows access to the cloud infrastructure using a set of protocols (REST, SOAP), enabling automation process to manage the cloud environment. The dashboard is the graphical user interface, which contains a set of cloud management features that give the user the ability to manage the environment, while respecting the policies applied to their own user (administrator or user). Orchestrator provides a capability to automate resource and processes management (creation, monitoring, and deployment).

Security layer plays a crucial role in the cloud environment because cloud computing has its own specific characteristics, such as scalability and multi-tenancy, dealing with costumer data in cloud data center. Thus, in according to Mell and Grace (2011), organizations should be aware of the security issues present in cloud computing, and that layer must be capable of guarantee, update and maintain high levels of confidence and transparency for costumers. Thus, the authentication process provides mechanisms such as SSH key pairs, certificates, usernames and passwords, used to access the cloud infrastructure. The authorization component, provides authority for a specifics tasks, applied to users. In addition, security groups correspond to the firewall rules applied to VMs, which can be part of a group with different subnets, and avoiding visibility to others groups. Single sign-on (SSO) uses the federation service to securely share identity between other networks by providing interoperability mechanisms for the authentication process. Security monitoring should be proactive and provide technologies to detect security threats that can compromise the cloud environment, and also enforce security policies.

The control layer provides the basic management capabilities for the IaaS cloud, including support for security policies for access control and data protection (CHANG, 2015). The SLA enforcement, and SLA monitoring ensures that QoS are performed by monitoring the required components. The Metering analyses the use of resources, providing the usage of resources, such as processing, storage, memory, and network). Policy control, involves policies to manage the security and privacy for the environment. The notification service is used to send different notifications (failures, purposes, registration, etc.), to members of the cloud. The orchestration service provides a catalog of workflows that helps integration and automation process, to manages the cloud infrastructure and other technologies.

The value-added services in according to Dukaric and Juric (2013), are a complimentary layer to the core service, introducing some key features in the analyzed cloud management tools. Thus, availability zones allow replication and high availability in different geographic regions. High Availability (HA) implements mechanisms that makes possible the redundancy of data among the nodes, with fault tolerance. The hybrid support allows integration with other cloud systems, such as, amazon or azure, it is possible to have a heterogeneous environment in order to maintain the efficiency and availability. The live migration offers the ability to migrate the virtual machine from one node to another in case of failover, without affecting the running services, transparently. The portability support is intended to enable the cooperation of different cloud systems, providing portability between different virtualization technologies, including different disks formats. That is, by using the federation service, the user can migrate the VM from one cloud technology to another (OpenNebula to Azure). Image contextualization is a resource that the user can add some features that characterize the image used by the VM, this is useful for providing preconfigured images, including features such as, DB characterization, network rules, authentication, etc. Finally, the virtual application support is used for multi-tier applications that use containers, which are capable of interacting with multiple VMs.

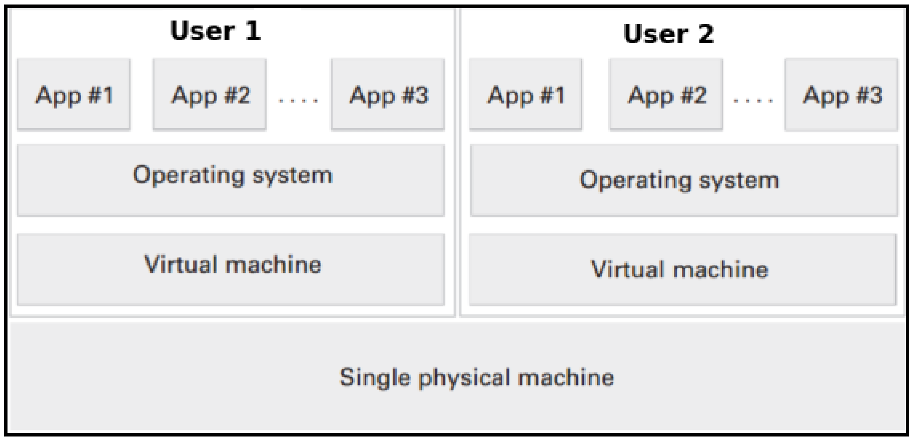
The update of Table 2.4, has shown the increasing effort to make the tools more flexible given the large number of new technologies introduced in the recent years. The case that most attracts attention is OpenStack, especially for the different features introduced in recent years (details of the components are listed in the OpenStack section), and new ones are emerging as shown on the official project website, revealing the intention to make an even more flexible tool. However, the large number of projects also have different maturity levels which makes the environment complex and dependent on interaction between them, which can lead to different performance results or even malfunctioning, being criticized by the community. On the other hand, we can realize that CloudStack introduced some new support technologies recently, but was overtaken by the OpenNebula project in terms of flexibility. Finally, it is important to emphasize that flexibility is not a parameter to predict “which cloud is best” as mentioned earlier, being only a reference guide to show the flexibility and the features offered by open source clouds.

2.2.11 Multi-Tenancy

Multi-tenancy in the cloud environment is defined by Mahmood (2013), as responsible for running multiple applications, database and hardware infrastructure among departments or companies, sharing the same central hardware and software infrastructure. Therefore, it is one of the most important features of cloud computing allowing to maximize the usage of resources by sharing them.

The cloud system architecture was designed to allow resources to be apportioned and shared efficiently between multiple tenants (MAUCH; KUNZE; HILLENBRAND, 2012). Thus, cloud computing offers a propitious environment to deploy diverse types of applications and products among the costumers, giving the low cost and a high availability infrastructure. However, different aspects of clouds require multiple customers with disparate requirements to be served by a single hardware infrastructure. Therefore, the virtualization process is responsible to create the virtual infrastructure that separates the resources among multiple tenants (BUYYA; BROBERG; GOSCINSKI, 2010).

The IaaS cloud tool is responsible for manage, implement and enforce the segregation in all forms of computer resources (compute, networking and storage) between the instances, thus allowing to isolate the tenants (TELFER, 2016). The Figure 2.23 shows an example of a physical machine from a cloud computing provider, which shares the same pool of hardware infrastructure to the hosting virtual machines.



Source: Extracted from Ruparelia (2016b)

Figure 2.23: Multi-Tenancy Overview.

Despite the fact that the benefits of the cloud computing are already known by the scientific and enterprise community, and the multi-tenancy is a core technological approach to create efficiencies in the cloud. The effect between multi-tenancy demands more research, because of the isolation of each individual user or workload, plus the difficulty to meet all the applications requirements in a virtualized environment at the same time, can affect the overall performance of a cloud (MAUCH; KUNZE; HILLENBRAND, 2012). In addition, Buyya, Broberg and Goscinski (2010) and Carroll, Kotzé and Merwe (2012) emphasizes that cloud environments are multi-tenancy in nature, and the environment may have performance issues if implemented or maintained improperly causing data corruption, contamination or unauthorized access.

2.2.12 Parallel Programming Models

With the advent of the multiprocessor system, different parallel programming models have made it possible to improve overall system performance. Allowing a process to perform more than one independent computation at the same time (TANENBAUM; BOS, 2014), (NICHOLS; BUTTLAR; FARRELL, 1996b).

Therefore, as emphasized by Butenhof (1997), the parallel application running on two processors can achieve almost double the performance compared to a serial version. However, as the parallelism scales up, more synchronization is required and there is more chances of lock and memory collisions, resulting in a poor performance. Thus, according to Amdahl's law, the level of parallelism is limited by the amount of serialization required (AMDAHL, 1967).

There are many different classes of parallel hardware and consequently many different parallel programming models. In this thesis, three main models are described: distributed-memory (MPI), shared-memory (OpenMP) and the POSIX Threads (Pthreads).

2.2.12.1 MPI - Massage Passing Interface

According to Gropp et al. (2014), it is a portable and standard interface for developing parallel programs using the distributed-memory programming model. It is used in large scale for parallel applications development, in science and engineering domains, in all sizes systems. This model idealizes that the programs will be executed by one or more processes, each of which has its own private address space (CHAPMAN; JOST; PAS, 2008).

According to Waku (2012), MPI is easy to use and has high portability. Easy to use is justified by the use of clearly specified simple interfaces as to their behavior, use and parameters for operation. The high portability is justified by the fact that with the use of message passing, it is possible to write programs that will work on machines of various types, such as distributed memory multiprocessors, workstation networks or hybrid environments of the most varied types. In the Listings 2.1 (C/C++) and 2.2 (Fortran), programs are shown that use MPI, representing master/slave communication. This programs are made by Snir (1998). According to Snir (1998) these codes make a simple exchange of messages between processes. All MPI calls are procedures and an additional parameter is used to return the value by the corresponding function. It can be seen that the Fortran strings have fixed length and are not null-terminated.

```c
char msg[20];
int myrank, tag = 99;
MPI_Status status;
...
MPI_Comm_rank( MPI_COMM_WORLD, &myrank ); /* find my rank */
if (myrank == 0) {
  strcpy( msg, "Hello there");
  MPI_Send( msg, strlen(msg)+1, MPI_CHAR, 1, tag, MPI_COMM_WORLD);
} else if (myrank == 1) {
  MPI_Recv( msg, 20, MPI_CHAR, 0, tag, MPI_COMM_WORLD, &status);
}
```

Listing 2.1: MPI implementation in inter-process message exchange (C - C++). Extracted from Snir (1998).

```
]
CHARACTER*20 msg
INTEGER myrank, ierr, status(MPI_STATUS_SIZE)
INTEGER tag = 99
...
CALL MPI_COMM_RANK( MPI_COMM_WORLD, myrank, ierr)
IF (myrank .EQ. 0) THEN
  msg = "Hello there"
  CALL MPI_SEND( msg, 11, MPI_CHARACTER, 1,
    tag, MPI_COMM_WORLD, ierr)
ELSE IF (myrank .EQ. 1) THEN
  CALL MPI_RECV( msg, 20, MPI_CHARACTER, 0,
    tag, MPI_COMM_WORLD, status, ierr)
END IF
```

Listing 2.2: MPI implementation in inter-process message exchange (Fortran). Extracted from Snir (1998).

The main idea is to allow development of applications that make use of MPI as a standard programming model, providing interoperability between programs and consequently increasing their scalability.

2.2.12.2 OpenMP - Open Multi-Processing

According to Chapman, Jost and Pas (2008), it is a shared-memory API, which is used to facilitate parallel programming. It is suitable for implementations of Symmetric multiprocessing (SMP) architectures.

OpenMP was defined by ARM (Architecture Review Board), with main objective of proving a common means for programming SMP architectures. Its creators developed an approach that was easy to learn and implement. The OpenMP API is designed to allow an incremental approach to an existing code, being part of this program parallelized. This model idealizes that programs will be executed on one or more processors that share some or all of the available memory. The main characteristic of OpenMP codes is the "pragma" (C/C++) and !$OMP (Fortran) use. They are guidelines that must be informed by the programmer, in the region of the code that will be parallelized.

In addition, the OpenMP is not a new programming language. It can be added to a sequential program in Fortran, C, or C++ to describe how the work should be shared between threads that will run on different processors or cores. Properly adding of OpenMP features into a sequential program makes most applications take advantage of shared-memory parallel architectures. Many applications have considerable parallelism that could be exploited. The success of OpenMP is attributed to some factors. They are:

- It is strongly intended for structured parallel programming.
- Compared to other languages, it is simple to use.
- It is widely used, being used in many platforms.
- OpenMP is timely.

```c
#include <stdio.h>
#include <stdlib.h>

void mxv(int m, int n, double * restrict a,
    double * restrict b, double * restrict c)
{
  int i, j;

#pragma omp parallel for default(none) \
    shared(m,n,a,b,c) private(i,j)
  for (i=0; i<m; i++)
  {
    a[i] = 0.0;
    for (j=0; j<n; j++)
      a[i] += b[i*n+j]*c[j];
  } /*-- End of omp parallel for --*/
}
```

Listing 2.3: OpenMP implementation of the matrix times vector product in C. Extracted from Chapman, Jost and Pas (2008).

The Listing 2.3 is also an example made by the authors Chapman, Jost and Pas (2008). It describes an OpenMP code in C/C++ language. According to Chapman, Jost and Pas (2008), a single pragma is sufficient to parallelize the outer loop. It was explicitly specified, for each variable, whether it is shared by all segments or whether each segment has a particular copy. A comment string is used to clearly mark the end of the parallel region.

```fortran
]
  subroutine mxv(m, n, a, b, c)

  implicit none

  integer(kind=4):: m , n
  real (kind=8):: a(1:m), b(1:m,1:n), c(1:n)
  integer :: i, j

!$OMP PARALLEL DO DEFAULT(NONE) &
!$OMP SHARED(m,n,a,b,c) PRIVATE(i,j)
  do i = 1, m
    a(i) = 0.0
    do j = 1, n
      a(i) = a(i) + b(i,j)*c(j)
    end do
  end do
!$OMP END PARALLEL DO
  return
  end subroutine mxv
```

Listing 2.4: OpenMP implementation of the matrix times vector product in Fortran. Extracted from Chapman, Jost and Pas (2008).

The Listing 2.4 is an example made by the authors Chapman, Jost and Pas (2008). It describes an OpenMP code in Fortran language. According to Chapman, Jost and Pas (2008), an OpenMP directive is sufficient to parallelize the outer loop. It has been explicitly specified for each variable whether it is shared by all threads or if each thread has a particular copy. The !$ OMP END PARALLEL DO directive to mark the end of the parallel region.

2.2.12.3 Pthreads - POSIX Threads

POSIX threads (pthreads) is a standard parallel library for UNIX, capable to achieve a high level of scalability by dividing a program into subtasks whose execution can be interleaved or run in parallel (NICHOLS; BUTTLAR; FARRELL, 1996a), (TANENBAUM; BOS, 2014). According to Bienia and Li (2009), phtreads has been used as a threading standard since 1995 and is one of the most commonly used libraries for programming shared-memory Unix machines.

Pthreads is commonly used in HPC, being characterized to be a lightweight, causing a low system overhead compared to the cost of creating and managing process, and also offers functions to synchronize threads, (TANENBAUM; BOS, 2014). Therefore, the Linux kernel can schedule these threads as with any other process or threads in the system, speeding up performance through parallelism, allowing the process to be concluded in close to half of time (MUKHEDKAR; VETTATHU; CHIRAMMAL, 2016), (CARVER; TAI, 2005).

The POSIX threads standard defines more than 60 functions calls, the main ones, are listed on table 2.5.

Thread call	**Description**
Pthread_create	Create a new thread
Pthread_exit	Terminate the calling thread
Pthread_join	Wait for a specific thread to exit
Pthread_yield	Release the CPU to let another thread run
Pthread_attr_init	Create and initialize a thread‘s attribute structure
Pthread_attr_destroy	Remove a thread‘s attribute structure

Table 2.5: Main thread function calls.

Source: Extracted from Tanenbaum and Bos (2014).

All Pthreads threads have their own properties, such as identifier, registers, and attributes, necessary to the thread utilization. However, for a program to take full advantage of Pthreads, it must be able to perform parallel tasks. An exemple of creating a thread is listed in the Figure 2.5. The authors Nichols, Buttlar and Farrell (1996a) shows a creation and termination of a thread. They also argues that the higher the level of parallelism achieved by a program, the higher it can enhance the performance and efficiency to do more than one thing at the time.

```
]
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#define NUM_THREADS 5

void *PrintHello(void *threadid)
{
   long tid;
   tid = (long)threadid;
   printf("Hello World! I am, thread #%ld!\n", tid);
   pthread_exit(NULL);
}

int main(int argc, char *argv[])
{
   pthread_t threads[NUM_THREADS];
   int rc;
   long t;
   for(t=0;t<NUM_THREADS;t++){
     printf("In main: creating thread %ld\n", t);
     rc = pthread_create(&threads[t], NULL, PrintHello, (void *)t);
     if (rc){
       printf("ERROR; return code from pthread_create() is %d\n", rc);
       exit(-1);
       }
     }

   /* Last thing that main() should do */
   pthread_exit(NULL);
}
```

Listing 2.5: Pthread - Thread creation and termination. Extracted from Nichols, Buttlar and Farrell (1996a).

However, despite all the benefits introduced by Pthreds implementation, the overhead cost in threaded code can affect the overall performance. As appointed by Butenhof (1997), the time it takes to synchronize threads by constantly write the same memory locations, causes the system spend a lot of time synchronizing the memory system on processors (overhead). Thus, it is necessary to the programmer design the code to avoid such problems.

2.3 SCIENTIFIC AND ENTERPRISE APPLICATIONS WORKLOAD

As emphasized by Cheveresan et al. (2007), workload is a utilization of a computer system to perform a representation from resource utilization in determinate amount of time. The utilization can guide performance optimizations both at the software and system configuration level. It can be customized and performed the tests, in a period of time and repeat them as long as needed. Thus, it is an important way to represent a behave of a real application.

Business processes inside enterprise fields relies on IT to offer different applications and softwares (eg., financial analysis, rendering, similarity search, enterprise backup) (SHROFF, 2010). With the growing diversity of softwares and applications inside the organizations, the enterprise workloads enable the mimic of this specific environment and became a natural way to represent and quantify the results.

Scientific workloads are used to simulate a real-world application with mathematical means on a research field. It is characterized by been a parallel jobs, small bags-of tasks and sometimes by small workflows, comprising most of sequential tasks (IOSUP; OSTERMANN; YIGITBASI; PRODAN; FAHRINGER; EPEMA, 2011). Such applications are used on a weather prediction, space exploration, genomic research, and others. This kind of research, requires a huge processing capability, which is able to supply the calculation level using a great number of floating point operations.

2.3.1 PARSEC

The Princeton Application Repository for Shared-Memory Computers (PARSEC[7] is a benchmark suite composed of multithreaded programs offering a wider variety of applications, (BARROW-WILLIAMS; NICK; FENSCH; CHRISTIAN; MOORE, 2009).

[7]http://parsec.cs.princeton.edu/index.htm

The PARSEC research project started in 2005 and was created to study the emerging processor architecture at the time, with programs that were not in the HPC domain. Due the lack of workloads available at the time, the early stages of PARSEC quickly received attention from the open source community, universities and enterprises. Currently at the version 3.0, the PARSEC is one of the most popular benchmark suite.

Focuses on programs from several applications domains, the suite includes 13 different workloads (Table 2.6). This programs cover different applications domains, with different models of parallelism and granularity. The benchmark is able to simulate the behavior of real applications, such as computer vision, video encoding, financial analytic, animation physics and image processing, (PARSEC, 2017a).

Program	**Application Domain**	**Parallelization**		**Working Set**	**Data Usage**	
		Model	**Granularity**		**Sharing**	**Exchange**
blackscholes	Financial Analysis	data-parallel	coarse	small	low	low
bodytrack	Computer Vision	data-parallel	medium	medium	high	medium
canneal	Engeneering	unstructured	fine	unbounded	high	high
dedup	Enterprise Storage	pipeline	medium	unbounded	high	high
facesim	Animation	data-parallel	coarse	large	low	medium
ferret	Similarity Search	pipeline	medium	ubounded	high	high
fluidanimate	Animation	data-parallel	fine	large	low	medium
freqmine	Data Mining	data-parallel	medium	unbounded	high	medium
raytrace	Rendering	data-parallel	medium	unbounded	high	low
streamcluster	Data Mining	data-parallel	medium	medium	low	medium
swaptions	Financial Analysis	data-parallel	coarse	medium	low	low
vips	Media Processing	data-paralel	coarse	medium	low	medium
x264	Media Processing	pipeline	coarse	medium	high	high

Source: Extracted from Bienia (2011)

Table 2.6: Different characteristics of PARSEC benchmarks.

The suit focuses on emerging workloads and was designed to be representative of next-generation shared-memory programs for chip-multiprocessors (INTEL, 2017b). Furthermore, some benchmarks allow increase the number of threads more than the number of cores available, as emphasized by Barrow-Williams et al. (2009), it gives to the operating system an effective manner to schedule the work. Additionally, some characteristics meet the requirements to this study, they are:

- **All applications are multithreaded:** The parallelism is widely used on science and enterprise fields, executing multiple programs instructions at same time. PARSEC is one of few benchmarks that are parallel.

- **Emerging workloads:** One of the main features from PARSEC is that algorithmic structure represents the behavior of applications that is likely to become commonly used in the near future.

- **Different applications domains:** Scientific and enterprise applications do have their own subcategories, like meteorological and space exploration, they both are from scientific fields. The suite includes programs which are wide and tries to be a representative as possible.

The workloads presented on PARSEC are composed of 10 applications and 3 kernels which represent common desktop and server programs (BIENIA, 2011). There are six input sets for each benchmark:

- **Test:** A very small input set to verify the minimum level of functionality of the program.

- **Simdev:** A very small input set used to check the behavior of real inputs, and simulation.

- **Simsmall, Simmedium and Simlarge:** They have different input sizes to simulate micro-architectural studies. Larger input sets contain bigger workload and consequently more parallelism.

- **Native:** The largest workload intended for native execution. Generally used to simulate a real program behavior, this input set gives us an extended utilization from computational resources, this allows a better methodology approaches related to our study.

2.3.1.1 Blackscholes

Blackscholes application is original from Intel RMS benchmark. It was merged on PARSEC project and was chosen to represent the field of analytic PDE (Partial Differential Equation) solvers. This kind of application is used in computation finance, which is intended to explore the amount of floating-point calculations a processor can perform (BIENIA, 2011).

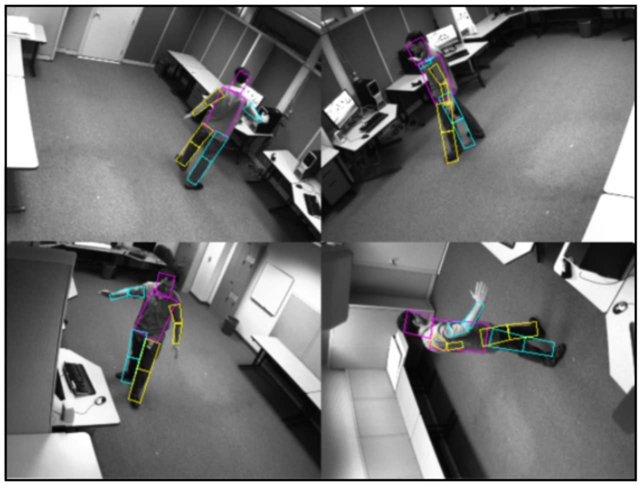
The native input that will be used in our tests has 10.000.000 values options, which serves as a reference. Initially, Blackscholes stores the portfolio with derivatives in array, so the data can be replicated if necessary to obtain sufficient derivatives for the benchmark. The program divides the portfolio into a number of work units equal to threads and processes them concurrently. Each thread repeats all derivations and calls a function to compute its price, (PARSEC, 2017b).

2.3.1.2 Bodytrack

The bodytrack simulates the application of computerized vision. The workload tracks the 3D pose of an unmarked human body with multiple cameras through a sequence of images, (PARSEC, 2017c). A filter is used to search for high dimensional spaces that do not assume markers of body movements or restricted movements. Therefore, the alignment is made by collecting samples from the foreground and edges of the images, and computing them. Thus, the program can recognize the body position and adds boxes to mark it in parts.

According to Bienia (2011), Bodytrack has a thread pool, and the input images are loaded using asynchronous I/O. The main thread sends a task to the thread pool whenever it reaches a parallel kernel. Bodytrack also use tickets to distribute the workload among the threads to load balance dynamically. Thus, the program has three parallel kernels:

- **Edge detection:** Bodytrack find the edge of the image, and then compare to eliminate false edges.
- **Edge smoothing:** A filter is applied to smooth the edges. The result is remapped to produce a map to correlate the distance from the edges. This kernel, has parallel phases to image rows and to columns. Calculate particle weights: Computes particles weights, by evaluating the foreground silhouette and the image edges. This step requires most computational power.
- **Particle resampling:** This kernel resizes particles by adding noise. Thus, a new set of particles is created.



Source: Extracted from Bienia (2011).

Figure 2.24: Bodytrack Structure.

The native input (Figure 2.24) used in our tests simulates 4 cameras, 261 frames, 4.000 particles and 5 annealing layers. The output images contain the exact location with recognized body and with marks to represent each limb, torso and head.

2.3.1.3 Canneal

The Canneal represents engineering workload, it uses a very aggressive synchronization strategy based on data race. This kernel employs cache-aware simulated annealing (SA) to minimize the routing cost of a chip design. Simulated annealing, is a thermal dynamic process which consists in raising the temperature of a solid and conducting controlled slow cooling (KIRKPATRICK; GELATT; VECCHI et al., 1983). The algorithm discards one element during each interaction to reduce the cache capacity misses, it increases data reuse.

On native input, 15.000 swaps are made per temperature, starting with 2.000°, with 2.500.000 netlist elements, and 6.000 temperature steps. The program focuses on write the final routing cost to the console. Thus, each thread randomly selects pairs of new elements with a Mersenne Twister (BIENIA, 2011), compute the change in cost and the actual temperature to decide if a change needs to be made. To avoid deadlocks, Canneal processes the pointer in the lower memory location first, which means that is the memory segment that the Linux kernel can directly address.

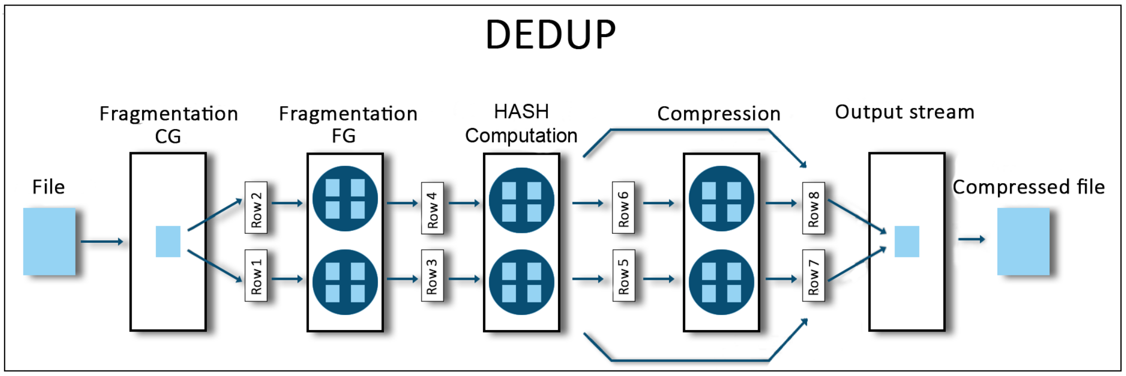
2.3.1.4 Dedup

Dedup is an application that compresses data by eliminating redundancy in a process called Data Deduplication. The deduplication method it is commonly used in a backup system because it achieves a good compression rate, reducing the size of the data (BIENIA, 2011), benefiting storage and network devices.

Dedup uses five kernels pipeline stages (2.25):

- **Coarse-grained fragmentation:** Serial stage responsible to read the input stream from disk, and split up in chunks with coarse-grained fragmentation.
- **Fine-grained fragmentation:** Parallel stage uses an algorithm to divide the elements in chunks with fine-grained fragmentation.
- **Hash computation:** Parallel kernel, used to computes the SHA1 in order to identify the chunks using a HASH table.

- **Compression:** Is responsible to compress the data blocks in parallel, using an algorithm to build a global HASH table.
- **Assemble output stream:** Serial stage responsible to reorder the block stages using the HASH table and compress the output stream.



Source: Extracted from Bienia (2011).

Figure 2.25: Dedup Structure.

The tests used in the native input, contains several individual files that sum 672MB in total. The compressed data stream can be decompressed to restore the original input data.

2.3.1.5 Facesim

Facesim is an application that simulates elements for animations. It uses a model of a human face and in a time sequence to simulate a muscle activation and calculate a visually realistic animation using underlying physics. Thus, the benchmark uses a face (Figure 2.26) mesh with 80.598 particles, 372.126 tethrahedra with 100 frames, to create a visually realistic result. The final state of the face is written in several files with different names (PARSEC, 2017d).

The parallelization employs a static mesh partitioning. All data covered by nodes belonging to more than one partition is replicated. Each time, partitions process all elements that contain at least one node owned by the particle, but only the results for nodes which are owned by the partition are written (BIENIA, 2011).

Source: Extracted from Bienia (2011).

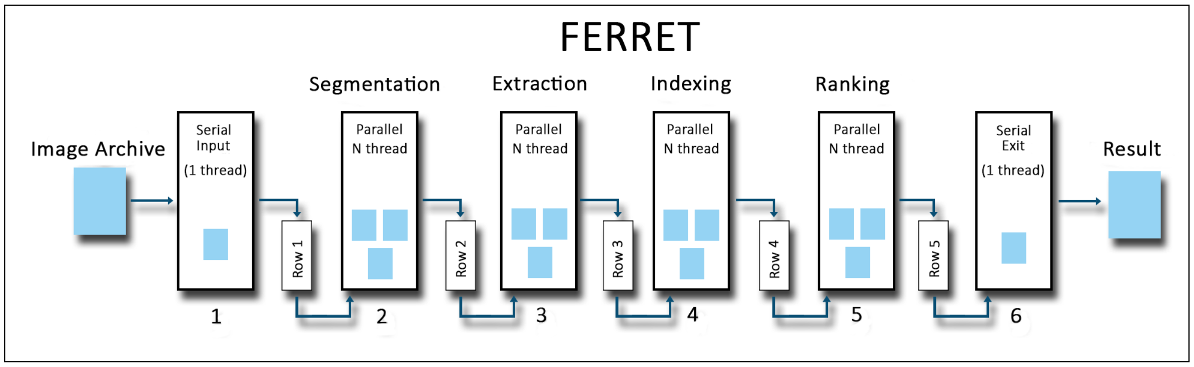
Figure 2.26: Facesim Structure.

There are three parallel kernels for its computations:

- **Update state:** Uses a method to solve the nonlinear equations to find the steady state of the simulated mesh. This technique simulates the effects such as flesh deformation.
- **Add forces:** This module calculates the independent forces of velocity acting on the simulation mesh. Sequentially, it reads the positions of the vertices and computes the force contribution to each one of the four nodes.
- **Conjugate gradient:** At this stage, the kernel uses an algorithm to solve the linear equation assembled by the previous two modules.

2.3.1.6 Ferret

Ferret is an application that employs a data similarity search such as audio, images, videos and 3D forms. This benchmark represents a next-generation of search engines for non-text document data types.



Source: Extracted from Bienia (2011).

Figure 2.27: Ferret Structure.

Ferret is parallelized using a six stages model (Figure 2.27, but the first and the last stages are serials. The intermediate stages are:

- **Image segmentation:** The image is decomposed using a method to segment the image into different objects, to find relevant areas.
- **Feature extraction:** Next, a multidimensional vector stores the color, shape and area properties in a mathematical description of the segmented image.
- **Indexing:** This stage searches the image in a database using HASH table to find a more accurate candidate image.
- **Ranking:** Computes detailed similarity and orders the images based on these results.

The similarity search using the native input simulates a queries with 3.500 images consulting a database with 59.695 images to find top 50 images.

2.3.1.7 Fluidanimate

Originally from Intel RMS benchmark, this application uses a Smoothed Particle Hydrodynamics (SPH) method to simulate an incompressible fluid for interactive animation purposes. This are capable to provide realistically animated fluids, adding substantial realism to interactive applications, such as virtual surgery simulators or computer games (Figure 2.28) (MÜLLER; CHARYPAR; GROSS, 2003).

Basically the process involved is a fluid detection and rendering surface, using five kernels to compute the mathematics involved in the process, which are:

- **Rebuild spatial index:** The program classifies the spatial structure to explore proximity information and verify the number of particles to be evaluated.
- **Compute densities:** Calculate the fluid density at the position of each particle, by analyzing its proximity.
- **Compute forces:** Next, the result will be used to compute forces, by evaluating internal and external forces (viscosity, pressure, gravity) and collisions.
- **Handle collisions with scene geometry:** Use geometry techniques to update particle collisions in the scene.
- **Update positions of particles:** Computes the acceleration of each particle and updates the position.

In our tests, native input is used, which employs an environment with 500.000 particles, and 500 frames. Finally, the state of the fluid is written at the end of computations.

2.3.1.8 Freqmine

Freqmine is used to simulate the data mining task for Frequent Itemset Mining (FIMI), which is very common in relevant areas, such as protein sequences, market data or log analysis. Its main feature is to find frequent pattern in large database. Thus, an array version of the Frequent Path Algorithm (FP-growth) is used, which is

Source: Extracted from Bienia (2011).

Figure 2.28: Fluidanimate Structure.

a method for mining the complete set of frequent patterns by fragment growth (HAN; PEI; YIN, 2000). In our tests native input is utilized, which is composed of a collection of 250.000 HTML web documents.

Freqmine is parallelized with OpenMP and have three parallel kernels:

- **Build FP-tree header:** Responsible to scan the transactions database and count the number of occurrences to create a header table for the FP-tree. The FP-tree will contain the item frequency information.
- **Construct prefix tree:** This stage has four parallelized loops, which are responsible to perform the second and final scan to build the initial database structure.
- **Mine data:** This stage has similarities to the previous two kernels, because it builds a new FP-tree for each recursion step. It uses the previously computed data structures to mines them and obtain the frequent item set information. Each thread calls the function independently, so the numbers of recursions are equivalent to the number of active threads.

Initially, the FP-growth identifies frequently occurring patterns and stores the relevant transaction database information in a compact data structure, the FP-tree. The first step is responsible to construct the header table, by encoding the data to each branch to represent a frequent itemset in decreasing order of frequency of the corresponding item. Then, the header table stores the number of occurrences of items in decreasing order of frequency. The data is crossed and computed, the entry is associated with a pointer to a node, which are related on a list. Each node also correlates the occurrence itemsets from the root to the current node. The last component, is responsible to lookup table to give all occurrences of items stored in the frequencies. This can be used during the mining phase to skip some scans, this is one of the key benefits of this algorithm. Finally, Freqmine provides the results of its computation to the console.

2.3.1.9 Raytrace

Raytrace is an application that simulates rendering and 3D scenes. The result is a photorealist image using a shading model that employs global information to calculate intensities of the shadows (WHITTED, 2005). Thus, this technique is commonly used in real-time animations, such as movies and computer games.

The computational complexity of the algorithm depends on the resolution of the output image and the scene (WHITTED, 2005). Thus, the native input workload uses a method that samples the object surfaces (Figure 2.29) and computes the shadows and the reflected light to render the image which have HDTV resolution (1920 X 1080 pixels).

2.3.1.10 Streamcluster

This kernel is used to simulate the data stream cluster, which represents continuous incoming data, such as multimedia data, financial transactions and telephone

Source: Extracted from Bienia (2011).

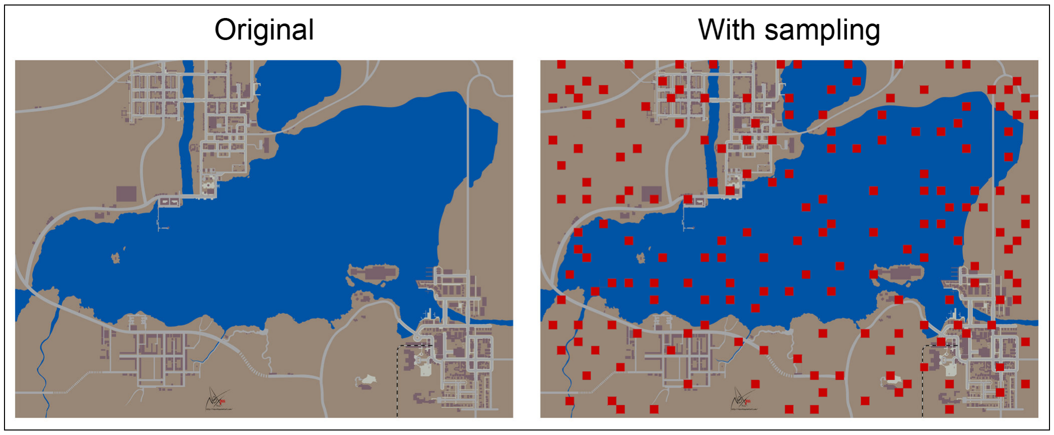
Figure 2.29: Raytrace Structure.

records (Streaming-Data Algorithms for High-Quality Clustering). The operation of stream clustering continuously uses a large amount of data and has to organize with real-time conditions. Thus, the program spends most of the time evaluating the opening gain of a new center and the cost that can be saved. This operation uses a parallelism with static partitioning of data points and the low-dimensional (original data) is memory bound, and becomes increasingly computationally intensive as the dimensionality increases.

The parallel function is used to gain computing power, this is a preliminary solution to compute how much cost can be saved by opening a new center. For every new one, makes a comparison to analyze the cost to make a new center, or reassigning from the existing point. If the comparison is advantageous, the results are committed. Thus, the native has 1.000.000 input points, 200.000 block size points and 128 point dimensions, 10-20 centers, up to 5.000 intermediate centers allowed (BIENIA, 2011).

2.3.1.11 Swaptions

Swaptions utilizes the HJM (Heath-Jarrow-Morton) framework to compute the evolving interest rate of risk management and equity liability for a pre-defined class of models. Swaptions utilizes the Monte Carlo (MC) method, which is used to calculate the numerical probability based on taking massive random samples at a high number of times. The Figure 2.30, shows an example of the method applied to calculate the area of a lake.



Source: Extracted from Bienia (2011).

Figure 2.30: Example of Monte Carlo Method used on Swaptions.

The Swaptions workload stores the portfolio in the array, where each entry corresponds to one derivative. The array is divided into a number of blocks equal to the number of threads and one block is assigned to each thread. Consequently, the threads replicate through all the swaptions and call the function for all the entries to compute the price generation a random HJM path for each MC. Based on the path that will be generated, it will be computed. The native input has 128 swaptions with 1.000.000 simulations.

2.3.1.12 Vips

The benchmark version of Vips is based on VASIRI Image Processing System (VIPS), which consists in an image processing system for larger images, similar to the conventional image processing package. However, VIPS can evaluate the image in parallel approach, meld together the operations, requiring no disc space for intermediates images and no unnecessary disc IO (CUPITT; MARTINEZ, 1996).

In its functionality, Vips performs an image transformation (common task on desktop computers), and constructs a transparent multithreaded image (Figure 2.31) processing pipelines on the fly, which means that the libraries required for the calculations are loaded in one compute resource, and automatically replicated to other computers resources on demand (TAN; ARZBERGER; KONAGAYA, 2006).

The image process transformation, has 18 stages grouped into the following kernels:

- **Crop:** The first step of pipeline, removes 100 pixels from all edges.
- **Shrink:** Reduces the image by 10%, using bilinear interpolation to compute the output values.
- **Adjust white point and Shadows:** VIPS adjust the brightness of the image to improve the visual quality. Then, a method is applied to the resulting image to obtain a sharp image.
- **Sharpen:** When printing the image, a blurring effect occurs, to correct this, a filter is applied to isolate the high-frequency signal component of the image.

The native input for VIPS uses an image with 18.000 x 18.000 pixels. It computes the image transformation in pipeline and replicates to multiple image regions

Source: Extracted from Bienia (2011).

Figure 2.31: Vips Structure.

concurrently. Actual image processing and any I/O is delayed as long as possible, and the intermediate results are represented in an abstract way. Thus, VIPS uses memory-mapped I/O to load parts of an input image on demand. After that, the operations are applied to the image region before the output region is written back to disk (BIENIA, 2011).

2.3.1.13 X264

The x264 is an application based on H.264 (Advanced Video Coding) video encoder. This benchmark is used to explore and eliminate data redundancy, in order to improve video encoding and decoding time. One of the most commonly used techniques is the motion compensation, which is used to reduce redundancy between sequential frames, the result will affect the final compression ratio. There are three possible ways to compress output frames to be encoded:

- **I-Frame:** Utilizes the entire image and does not depend on other frames. A prediction block is made by subtracting the current block before encoding.

- **P-Frame:** Includes only the changed parts of an image form the previous frame. It is formed by shifting samples from previously encoded frames to compensate the camera movement.

- **B-Frame:** Constructs the data from previous frames, using inter prediction with two compensated motion signals.

The X264 uses a parallel algorithm with pipeline model, which consist in one stage per input frame. Thus, the number of pipeline stages is equal to the number of encoder threads in parallel. However, fast video movements can cause delays reducing the speedup of X264. To compensate this effect, the parallelization needs to execute a greater number of threads than the number of cores, this allows a better performance.

Source: Extracted from Bienia (2011).

Figure 2.32: X264 Structure.

The video utilized on the benchmark is uncompressed version of Elephants Dream short film (Figure 2.32), the native input has 1920 x 1080 pixels (HDTV resolution), with 512 frames.

2.3.2 NPB (NAS Parallel Benchmark)

Developed by NASA (National Aeronautics and Space Administration) in the 1990s, NAS (NASA Advanced Supercomputing Division, previously known as Numerical Aerodynamic Simulation Program) Parallel Benchmark[8] is a suite of applications designed to aid performance benchmarking of parallel supercomputers. Derived from computational fluid dynamics (CFD) applications, these benchmarks consist of five kernels and three pseudo-applications including new benchmarks such as unstructured adaptive mesh, parallel I/O, multi-zone applications, and computational grids. Currently, in version 3.3.1, NPB is one of the most popular benchmark suites that work with scientific applications (NPB, 2017).

Benchmark	Focus	Language	Version
EP	Math Fucntions (Floating point performance)	Fortran	OpenMP, MPI
IS	Network Bandwidth / Memory Bandwidth (Integer performance)	C	OpenMP, MPI
CG	Network Bandwidth / Memory Bandwidth (Irregular Communication)	Fortran	OpenMP, MPI
MG	Memory Bandwidth (Regular communication)	Fortran	OpenMP, MPI
FT	Network Bandwidth / Memory Bandwidth (All to All communication)	Fortran	OpenMP, MPI
BT	Network Bandwidth / Instruction Cache (Floating point performance)	Fortran	OpenMP, MPI
SP	Memory Bandwidth (Floating point performance)	Fortran	OpenMP, MPI
LU	Network Latency / Intruction Cache (Regular communication)	Fortran	OpenMP, MPI

Source: Extracted from NPB (2017) and Roloff et al. (2012).

Table 2.7: Overview of the eight original NAS benchmarks.

According to (NPB, 2017), the eight original benchmarks mimic the computational and data movement in CFD applications, in particular each one is created to test a specific area (Table 2.7). They are divided into five kernels and three pseudo-applications. The five kernels are used for calculations and computational fluid dynamics (CFD). The three pseudo-applications apply the resolution of a system of nonlinear partial differential equations, representing the main benchmarks of the computationally-intensive CFD building block in common use today for the numerical solutions of three-dimensional equations using finite-volume, finite-difference on structured grids. Both

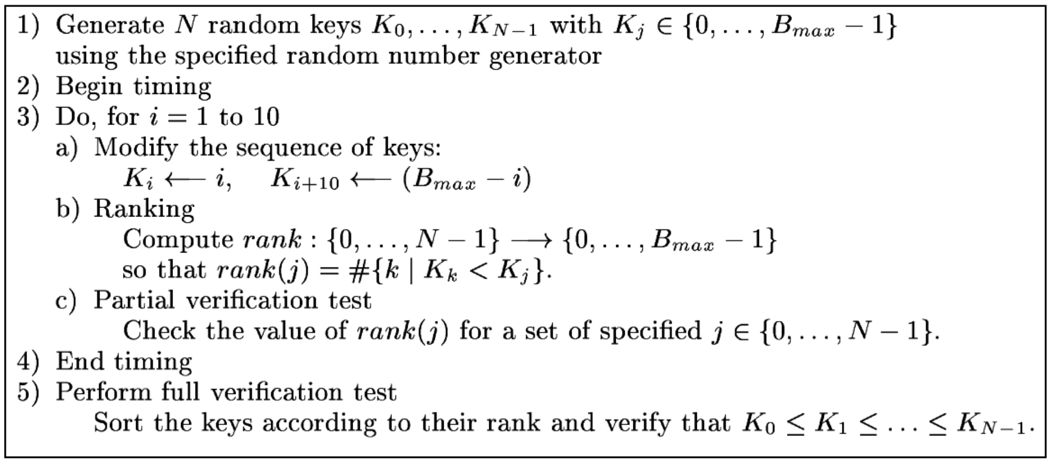
[8] https://www.nas.nasa.gov/publications/npb.html

are described below.

2.3.2.1 Kernel IS: ***I***nteger ***S***ort

Sorts whole numbers using "bucket sort". According to Bailey (2009), the IS Kernel performs a large integer sort operation, important in "particle method" codes. It makes tests both integer computation speed and communication performance. In addition, the same author defines this benchmark as a specific generator of a large array by a scheme and then sort it. The test of this benchmark is to certify that the array is in a sorted order.

According to Grün and Hillebrand (1998), the IS kernel does not realy sort a list of small integer vallues, but it classifies them by assigning each element of the list a number indicating its position in the ordered list. The benchmark is parameterized and scalable, that is, there are six classes (Table 2.9) that define the parameters that control memory consumption and the benchmark runtime. The Figure 2.33 shows the Integer Sort Specification.



```
1) Generate $N$ random keys $K_0, \ldots, K_{N-1}$ with $K_j \in \{0, \ldots, B_{max} - 1\}$
   using the specified random number generator
2) Begin timing
3) Do, for $i = 1$ to $10$
   a) Modify the sequence of keys:
         $K_i \longleftarrow i, \quad K_{i+10} \longleftarrow (B_{max} - i)$
   b) Ranking
         Compute $rank : \{0, \ldots, N-1\} \longrightarrow \{0, \ldots, B_{max} - 1\}$
         so that $rank(j) = \#\{k \mid K_k < K_j\}$.
   c) Partial verification test
         Check the value of $rank(j)$ for a set of specified $j \in \{0, \ldots, N-1\}$.
4) End timing
5) Perform full verification test
      Sort the keys according to their rank and verify that $K_0 \leq K_1 \leq \ldots \leq K_{N-1}$.
```

Source: Extracted from Grün and Hillebrand (1998).

Figure 2.33: Integer Sort Specification.

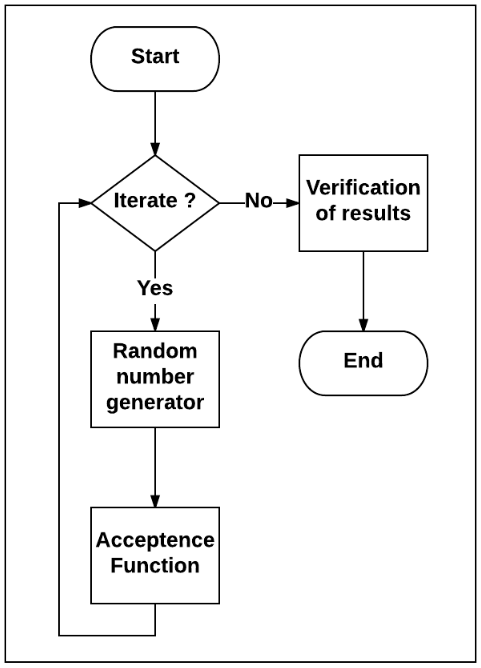
The random numbers for the keys are generated in two steps. First, a linear congruence method produces uniformly distributed random pseudo-numbers r_i using the following recursive algorithm: r_o = 314159265, r_{i+1} = 5^{13} . r_i mod 2^{46}. Then, the keys K_i are computed as K_i = $\frac{1}{4}\cdot \left(\sum_{k=0}^{3} r_{4i+k}\right) \cdot \frac{B_{max}}{2^{46}}$ resulting in a Gaussian distribution of keys (GRÜN; HILLEBRAND, 1998).

*2.3.2.2 Kernel EP: An **E**mbarrassingly **P**arallel Benchmark*

Independent generation of Gaussian values and random variables using the Polar Marsaglia method. According to Bailey (2009), the EP Kernel provides an estimate of the attainable limits for floating-point performance, that is, processor without interprocessor communication. The test of this benchmark requires that the sums agree with reference values for a specified tolerance, and also that the ten counts of deviates in square annuli agree with reference values.

According to Fernández et al. (2006), the EP kernel measures floating-point performance by tabulating statistics on pseudo-random data. It displays the simplest possible communication pattern between processes. There are two main parts to this benchmark: the random number generator, and the main processing loop that tests successive pairs of random numbers. The parallelization of the problem is direct: each member of a group of processors works independently on a subset of the random numbers and the annuli counts are reported at the end of processing (WHITE; ALUND; SUNDERAM, 1995).

The Figure 2.34 represents the organization of the benchmark in a flowchart. Internally to the loop, calculations are present in non-parallelizable loops, which involve the generation of random numbers and the verification of the adequacy of these numbers according to the acceptance function.



Source: Extracted from Pilla (2009).

Figure 2.34: Representation of the EP benchmark in a simplified flowchart.

*2.3.2.3 Kernel CG: **C**onjugate **G**radiant*

Calculation of matrix values. According to Bailey (2009), a gradient method is used to compute an approximation to the smallest eigenvalue of a matrix. It is typical of unstructured grid computations, testing long and irregular communication, using unstructured matrix vector multiplication. The test of this benchmark is that the value of the equation, must agree with a reference value for a specified tolerance.

According to Zhang et al. (2005), the CG is a memory intensive benchmark. It uses the inverse power method to find an estimate of the largest eigenvalue of a symmetric positive definite sparse matrix with a random pattern of nonzeros. The inverse power method involves solving a linear system of equation $Az = x$ using the conjugate gradient method (LI; HUANG; CAMERON, 2008).

The Figure 2.35 shows the main iteration in CG, where "nitter" represent the size of the system, the "λ" represents the different problem sizes and the "ζ" represents the calculated eigenvalue estimate. All of these parameters are clearly specified in the Table 2.9.

```
Initialize random number generator
Use Makea() to generate sparse matrix
$x = [1, 1, ..., 1]^T$
DO it =1, niter   # For CLASS A, niter=15
    Running the function conj_grad to solve the
    system $Az = x$ and return $\| r \|$,
    $\zeta = \lambda + 1/(x^T z)$
    $x = z / \| z \|$
ENDDO
```

Source: Extracted from Li, Huang and Cameron (2008).

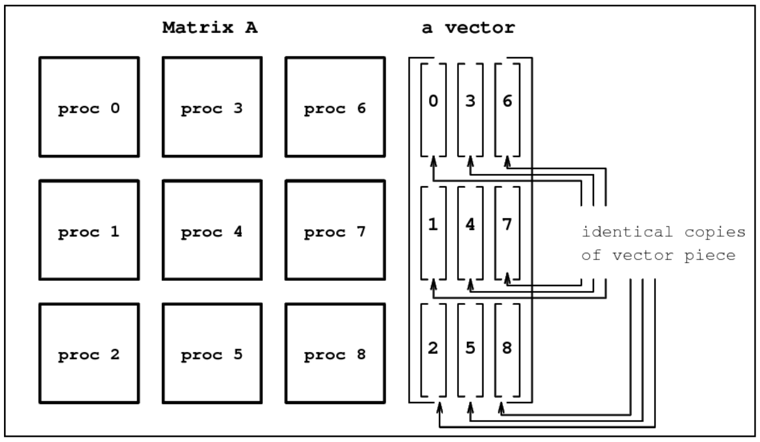
Figure 2.35: The main iteration in CG benchmark.

According to White, Alund and Sunderam (1995), the primary design issue in kernel CG is how to store the various work vectors used in the power and conjugate gradient methods. The operations on these vectors are two dot products, three vector-vector additions, and one matrix-vector multiply for each of the 25 conjugate gradient iterations in each of the 10 power method iterations. The NAS choice is to divide a vector into as many pieces as there are rows of processors, with each processor on a row holding an identical copy of that piece of the vector as shown in the Figure 2.36.

*2.3.2.4 Kernel MG: **M**ulti **G**rid*

Intensive communication for short and long distance memory. According to Bailey (2009), it is a simplified multigrid calculation. It requires long and structured communication and short and long distance data communication test. The test of this benchmark requires that a specified number of iterations must agree with a reference value to within a specified tolerance.

According to White, Alund and Sunderam (1995), the MG kernel specifies a



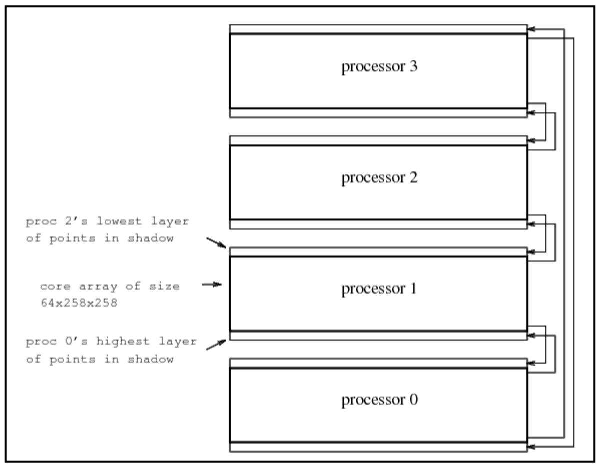
Source: Extracted from White, Alund and Sunderam (1995).

Figure 2.36: Matrix and vector organization in Kernel CG with nine processors.

Poisson problem $\nabla^2 u = v$ with periodic boundary conditions defined in each class (*i.e.*, 512 x 512 x 512). v is 0 at all coordinates except in 10 points which are +1.0 and points which are -1.0.

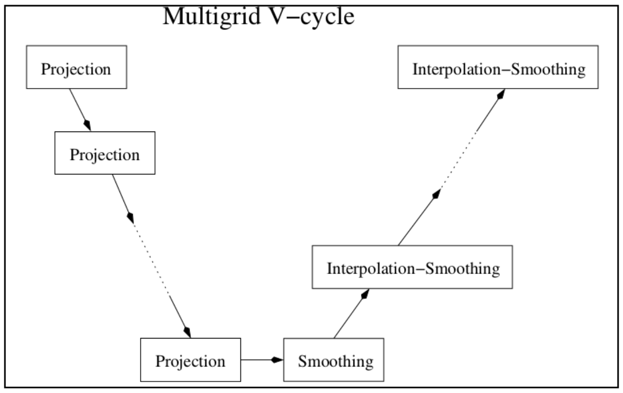
In addition, the authors White, Alund and Sunderam (1995) emphasized that any scheme for partitioning data from this problem will involve operations in border regions where a processor must "touch" the points that were assigned to the neighboring processor. Two schemes are typically used in this type of parallel problem: keep the border of the neighboring process "in the shadow" along with the schema to update them before they are outdated; Or requesting the points on demand. The Figure 2.37 illustrates the arrangement of the processors and the communication pattern for the case of four processors.

According to Frumkin (2005), The MG kernel uses a V-cycle pattern start from the projection of the finest grade waste for the coarser grades. Then it finds a solution in the coarser grid. Finally, it raises the solution on the finer grid by an interlaced sequence of interpolating and smoothing operations (Figure 2.38).


processor 3
processor 2
proc 2's lowest layer of points in shadow
core array of size 64x258x258
processor 1
proc 0's highest layer of points in shadow
processor 0

Source: Extracted from White, Alund and Sunderam (1995).

Figure 2.37: Kernel MG processor topology and communication standard with four processors.



Source: Extracted from Frumkin (2005).

Figure 2.38: Multigrid V-cycle pattern.

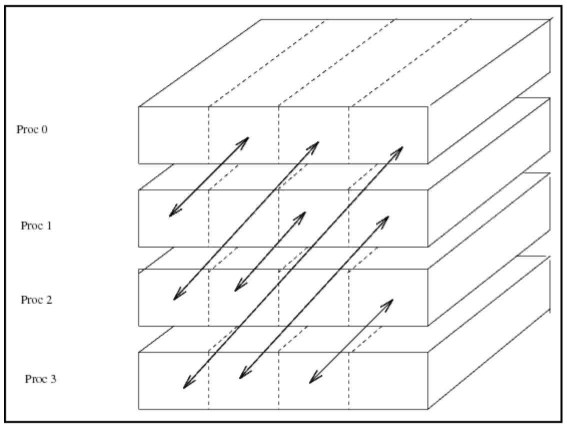
*2.3.2.5 Kernel FT: Fast **F**ourier **T**ransform*

Fourier transform method, using all-to-all communication. According to Bailey (2009), it is 3-D partial differential equation solved using FFTs, performing the essence of many "spectral" codes. As a definition, it is a rigorous test of heavy long-distance communication performance.

In addition, the same author define this benchmark, as a discrete version of the original PDE (Partial Differential Equation), by computing the forward 3-D discrete Fourier transform (DFT) of the original state array, multiplying the results by exponentials, and then performing reverse 3-D DFT. The DFTs can be quickly evaluated using a 3-D FFT algorithm. The test of this benchmark is to match the checksum of a subset of the final array with reference values.

The authors White, Alund and Sunderam (1995), emphasized that FT solves a discrete form of PDE $\frac{\partial u(x,t)}{\partial t} = \alpha \nabla^2 u(x,t), x \in R^3, u \in C$ using the Discrete Fourier Transform. After 3-D Fourier transformation of each side, this equation becomes $\frac{\partial u(z,t)}{\partial t} = -4\alpha\pi^2|z|^2 v(z,t)$ with the initial solution $v(z,0) = v_o(z)$ where $v_o(z)$ is the 3-D Fourier transform of $u_o(x)$. The solution of this equations is $v(z,t) = e^{-4\alpha\pi^2|z|^2 t} v(z,0) = e^{-4\alpha\pi^2|z|^2 t} v_o(z)$ where $v(x,t)$ is obtained as the inverse 3-D Fourier transform of v. The above is also, true for the discrete form of the original PDE when the 3-D Discrete Fourier Transform (DFT) is applied. The serial version of Kernel FT first generates and initial $n_1 x n_2 x n_3$ array of complex numbers, $U(j,k,l), 0 \leq j < n_1, 0 \leq k < n_2, 0 \leq l < n_3$ initialized with values from a pseudorandom number generator. Then 3-D DFT of U, called V, is computed using a 3-D FFT. Finally for each $t \in [1,6] W_{j,k,l}(t) = e^{-4\pi^2\alpha|j^-2+k^-2+l^-2|t} V_{j,k,l}$ is computed, where $\bar{j}$ is defined as $j\ for 0 \leq j < n_1/2$ and $j - n_1$ for $n_1/2 \leq j \leq n_1$ and $\bar{k}$ and $\bar{l}$ similarly defined with n_2 and n_3. The inverse DFT is then applied to obtain the solution array $X_{j,k,l}$.

The DFT 3-D is implemented as 3 scans with a 1-D FFT routine, one in each of the j, k and l directions and the results are multiplied to form the 3-D DFT. The scans in the directions j and k can be made on the date initially assigned to each processor, but for scanning in the direction l, the matrix must first be transposed, involving communication as shown in Figure 2.39.



Source: Extracted from White, Alund and Sunderam (1995).

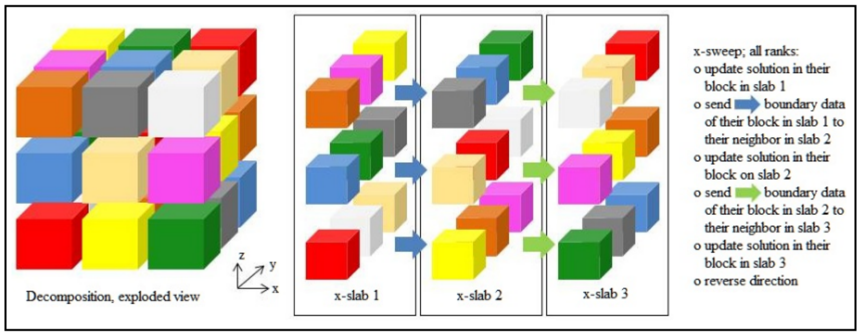
Figure 2.39: Kernel FT partitioning/communicating with four processors.

*2.3.2.6 BT Pseudo-Application: **B**lock **T**ridiagonal*

Tridiagonal blocking algorithm. According to Bailey (2009), it runs a synthetic CFD benchmark solving multiple independent systems of non-diagonal dominant tridiagonal equations with a block size (5 X 5). This kernel exhibits more border parallelism compared to the other two pseudo-applications.

The author Frumkin (2005), emphasized that BT, SP and LU solve a discretization of the Navier-Stokes equation $K(u^{q+1} - u^q) = Lu^q$, where u is a five-dimensional vector of density, moments and energy, K is a discretization of the Navier-Strokes operator and L is the right-hand site operator. The benchmarks represent three methods to approximate (or split) K as a product of linear operators. The BT benchmark splits K into a product of three Block Tridiagonal operators one along each dimension of the computational mesh, SP uses the Beam and Warming split that involves three Scalar Pentadiagonal operators interleaved with diagonal matrices of 5x5 blocks and LU employs a Lower and Upper diagonal splitting.

According to Garcia and Freitas (2015), BT is an essential part of computational fluid dynamics, solving a synthetic system of nonlinear partial differential equations, which involves dependencies of global data.


Decomposition, exploded view
z
y
x
x-slab 1
x-slab 2
x-slab 3
x-sweep; all ranks:
o update solution in their block in slab 1
o send boundary data of their block in slab 1 to their neighbor in slab 2
o update solution in their block on slab 2
o send boundary data of their block in slab 2 to their neighbor in slab 3
o update solution in their block in slab 3
o reverse direction

Source: Extracted from Wijngaart, Sridharan and Lee (2012).

Figure 2.40: NPB BT multi-partition decomposition and algorithm with nine MPI ranks.

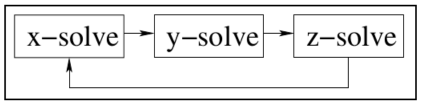
In Figure 2.40 it is showed an example of multipartition decomposition for 9 ranks. In this figure, each rank owns $\sqrt{9}$ (*i.e.*3) blocks. Blocks of same color belong to the same rank. Y-solver (sweeps into plane of paper) and z-solver (sweeps from bottom to top) are executed similarly

In addition, the authors Wijngaart, Sridharan and Lee (2012) emphasized that BTs workload is derived from a mature CFD (Computational Fluid Dynamics) code and used in several research centers and the aeronautics industry. It is based on the algorithm of Implicit Alternate Direction that presents strong dependencies of data that change periodically during the course of the computation. This requires advanced parallelization techniques, especially in clusters, which provide a good load balancing, avoiding an enormous amount of very fine grain communications.

*2.3.2.7 SP Pseudo-Application: **S**calar **P**entadiagonal*

Pentidiagonal Algorithm. According to Bailey (2009) it performs a synthetic CFD problem solving multiple, independent systems of non diagonally dominant, scalar, pentadiagonal equations. SP and LU involve global data dependencies. According to Bailey et al. (1991), SP and BT are representative of computations associated with CFD codes operators.

In addition, SP and BT are similar in many ways, with their main difference being in relation to the ratio of communication to computation.



Source: Extracted from Frumkin (2005).

Figure 2.41: The ADI pattern.

The author Frumkin (2005) emphasized that BT and SP represent the Alternative Directions Implicit (ADI) pattern. In this pattern, the algorithm is structured in order to perform iterative solutions of linear systems in the x, y and z directions, 2.41.

*2.3.2.8 LU Pseudo-Application: **L**ower-**U**pper*

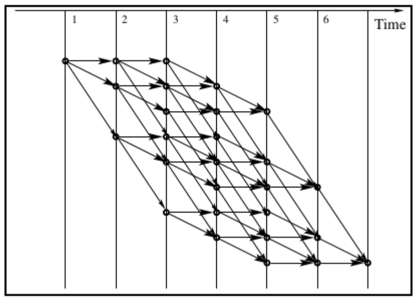
Gauss algorithm. According to Bailey (2009), it performs a synthetic computational fluid dynamics calculation by solving regular, block (5 X 5) lower and upper triangular systems.

Moreover, the author Pennycook et al. (2011) emphasized that LU code implements a simplified compressible Navier-Strokes equation solver which employs a Gauss-Seidel relaxation scheme with symmetric successive over-relaxation (SSOR) to solve linear and discretized equations.

The author Frumkin (2005) emphasized that LU benchmark have the following matrices:

$$\begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & -1 \end{bmatrix} and \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

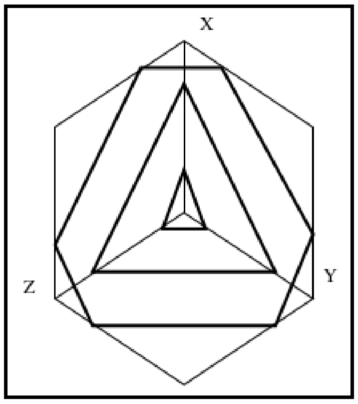
Thus, LU can be structured as a two-dimensional pipeline or as a hyperplane-based computation. In the two-dimensional pipeline (Figure2.42), on k^{th}-iteration the computations are performed in each horizontal plane along the diagonal $x + y = w$, where $w = k - z$ or $w = 3N - k - z$ depending on whether L or U operator is applied.



Source: Extracted from Frumkin (2005).

Figure 2.42: Seven stages of two-dimensional pipeline for processing points of 3x3x3 lattice.

In the hyperplane version of LU (Figure 2.43), the update of the flow vectors performed simultaneously at all points of a hyperplane $x + y + z = k$.



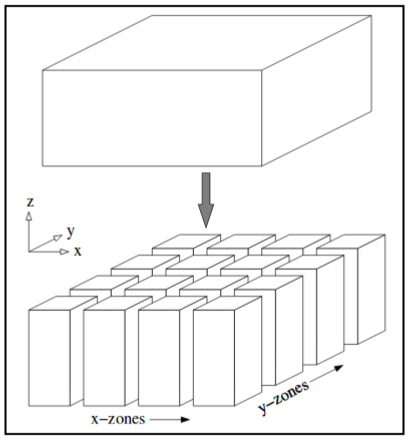
Source: Extracted from Frumkin (2005).

Figure 2.43: Several stages of processing in the hyperplane algorithm.

2.3.2.9 *New benchmarks added to the NPB*

According to NPB (2017), the NPB has created other benchmarks to evaluate multi-zones, unstructured computation, parallel I/O, and data movement. The multi-zones versions derived from single-zone pseudo-applications, represented by the acronyms BT-MZ, SP-MZ and LU-MZ. They are build to explore various levels of parallelism in applications and test the effectiveness of multi-level and hybrid parallelization paradigms and tools.

In addition the author Wijngaart and Haopiang (2003), emphasized that in each multi-zone version, a logically rectangular discretization mesh is divided into a two-dimensional horizontal tiling of three-dimensional zones of approximately the same aggregate size as the original NPB, as showed in Figure 2.44.



Source: Extracted from Wijngaart and Haopiang (2003).

Figure 2.44: Two-dimensional tiling of three-dimensional mesh.

- **BT-MZ (Block Tridiagonal - Multi-Zone)** There are zones of irregular size in a problem class, with an increasing number of zones as the problem class grows. According to Jin and Wijngaart (2003), the number of zones increases with problem sizes. However, the total mesh is divided in a way that the sizes of the zones cover a significant range. This is achieved by increasing sizes of successive zones in a particular coordinate direction of an approximately geometric shape.
- **SP-MZ (Scalar Pentadiagonal - Multi-Zone)** It has uniform-sized zones in a problem class, with an increasing number of zones as the problem class grows. According to Jin and Wijngaart (2003), the overall mesh is divided so that the zones are identical in sizes. However the number of zones grows as the problem size grows.

- **LU-MZ (Lower-Upper - Multi-Zone)** It has has irregular-sized zones in a problem class, with a fixed number of zones for all problem classes. According to Jin and Wijngaart (2003), the overall mesh is divided in a way that the zones are identical in size, which makes it easier to balance the parallel application load.

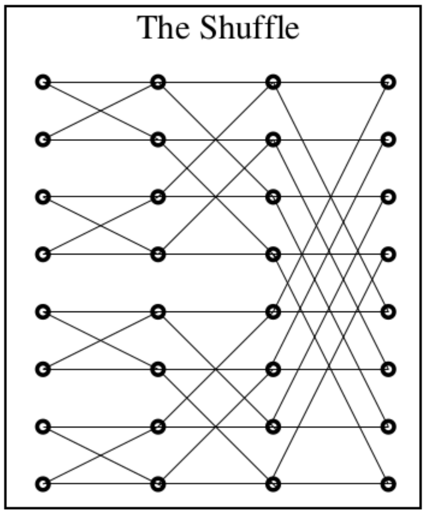
- **UA U**nstructured **A**daptive mesh, dynamic and irregular memory access. According to Feng et al. (2004), this benchmarks provides an standardized method for evaluating the performance of computer systems when running scientific applications whose memory access patterns are irregular and constantly changing.

- **BT-IO** Test of different parallel I/O techniques. According to Wong and Der Wijngaart (2003) this benchmark is based in the BT (**B**lock **T**ridiagonal) kernel of the eight original benchmark of NPB. It was created because a new bottleneck became evident in I/O applications. Limitations on I/O performance were due in part to the fact that application and user developers need data files whose structure does not depend on the number of processors involved in generating the files. IT evaluates I/O performance improvements through collective buffering.

- **DC D**ata **C**ube / **DT D**ata **T**raffic According to Frumkin and Shabano (2003), both work with randomly generated data and use trees and a shuffle as data flow patterns. According to Frumkin (2005), the DC benchmark creates RB tree to sort tuples from a dataset. The ability of DC to evaluate multiple levels of memory hierarchy depends on the size of the tree. The tree fits into L1-cache (S class) and grows beyond main memory (class B). The same author emphasized that DT benchmark uses quad-trees (black hole and white hole) and the binary shuffle (Figure 2.45) as the task graphs.

2.3.2.10 GridNPB

Another distribution of NPB is the GridNPB, designed specifically to evaluate the performance of computational grids. GridNPB is composed by four benchmarks

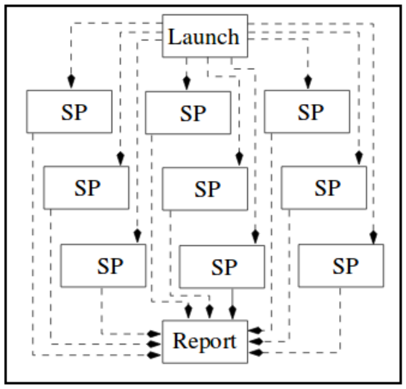


Source: Extracted from Frumkin (2005).

Figure 2.45: DT Shuffle.

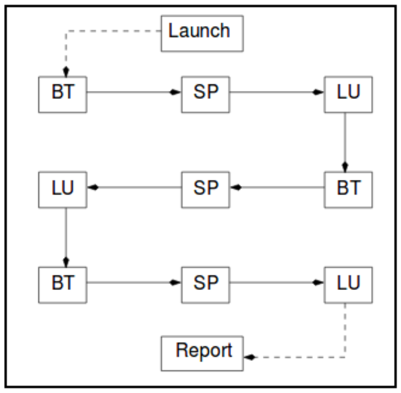
that consist in a collection of communicating tasks derived from NPB. They symbolize distributed applications that typically run on grids. The four benchmarks are described below:

- **ED - Embarrassingly Distributed:** According to Wijngaart and Frumkin (2002), it represents a class called parameter studies, which constitute several independent executions of the same program, with different input parameters, (Figure 2.46).
- **HC - Helical Chain:** According to Wijngaart and Frumkin (2002), it represents long chains of repeating process, such as a set of flow computations, executed one after another, (Figure 2.47).
- **VP - Visualization Pipeline** According to Wijngaart and Frumkin (2002), VP represents chains of compound process, such as those found in the visualization of flow solutions as the simulation progress, (Figure 2.48).
- **MB - Mixed Bag:** According to Wijngaart and Frumkin (2002), MB works with the sequence of flow computation, processing and visualization (similar to VP), but



Source: Extracted from Frumkin (2005).

Figure 2.46: ED - Embarrassingly Distributed pattern.



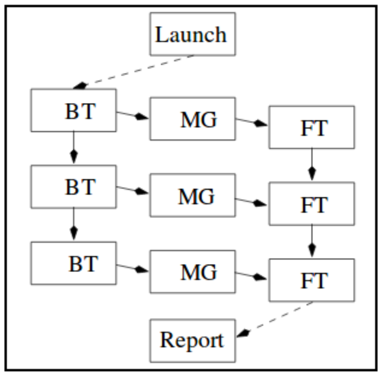
Source: Extracted from Frumkin (2005).

Figure 2.47: HC - Helical Chain pattern.

focuses on the introduction of asymmetric, (Figure 2.49).

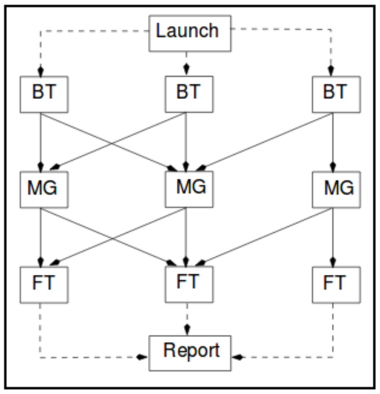
2.3.2.11 Benchmark Rules

According to NPB (2017), even through the NPB benchmark is specified in a technical document and deployments are usually free to code, some rules were defined for it. The purpose of these rules was to limit the deployments to "reasonable" code, similar as used in real scientific applications. The rules are described bellow:



Source: Extracted from Frumkin (2005).

Figure 2.48: VP - Visualization Pipeline pattern.



Source: Extracted from Frumkin (2005).

Figure 2.49: MB - Mixed Bag pattern.

- All floating point operations must be performed using 64-bit floating point arithmetic (at least).
- All benchmarks must be coded in either Fortran, C or Java, with certain approved extensions.
- One of the three languages should be selected for the entire implementation.
- Any language extension or library routine that is employed in any of the benchmarks must be supported by the system vendor and available to all users.

- Subprograms and library routines not written in Fortran, C or Java may only perform certain basic functions.
- All rules apply equally to subroutine calls, language extensions and compiler directives.

2.3.2.12 Benchmark Classes

According to NPB (2017), the intensities of workloads are classified into classes (W, S, A, B, C, D and E). This classes are described in the Table 2.8.

Class S	Small for quick tests purposes
Class W	Workstation Size
Classes A, B, C	Standard test problems. Increase about 4X it size from one class to the next.
Classes D, E, F	Large test problems. Increase about 16X it size from each of the previous classes.

Source: Extracted from NPB (2017).

Table 2.8: Benchmark Classes of NPB.

The Table 2.9 shows problem sizes and parameters for each class defined in NPB 3.3. In column, "Benchmark" all benchmarks of version NPB 3.3 are identified. In the "Parameter" column, the important parameters of each benchmark are identified. In the columns "Class S", "Class W", "Class A", "Class B", "Class C", "Class D", "Class E", are represented the values referring to the parameters. Empty cells indicate undefined problem sizes.

Benchmark	Parameter	Class S	Class W	Class A	Class B	Class C	Class D	Class E
CG	no. of rows	1400	7000	14000	75000	150000	1500000	9000000
	no. of nonzeros	7	8	11	13	15	21	26
	no. of iterations	15	15	15	75	75	100	100
	eigenvalue shift	10	12	20	60	110	500	1500
EP	no. of random-number pairs	2^{24}	2^{25}	2^{28}	2^{30}	2^{32}	2^{36}	2^{40}
FT	grid size	64 x 64 x 64	128 x 128 x 32	256 x 256 x 128	512 x 256 x 256	512 x 512 x 512	2048 x 1024 x 1024	4096 x 2048 x 2048
	no. of iterations	6	6	6	20	20	25	25
IS	no. of keys	2^{16}	2^{20}	2^{23}	2^{25}	2^{27}	2^{31}	
	key max. value	2^{11}	2^{16}	2^{19}	2^{21}	2^{23}	2^{27}	
MG	grid size	32 x 32 x 32	128 x 128 x 128	256 x 256 x 256	256 x 256 x 256	512 x 512 x 512	1024 x 1024 x 1024	2048 x 2048 x 2048
	no. of iterations	4	4	4	20	20	50	50
BT	grid size	12 x 12 x 12	24 x 24 x 24	64 x 64 x 64	102 x 102 x 102	162 x 162 x 162	408 x 408 x 408	1020 x 1020 x 1020
	no. of iterations	60	200	200	200	200	250	250
	time step	0.01	0.0008	0.0008	0.0003	0.0001	0.00002	0.000004
(BT-IO)	write interval	5	5	5	5	5	5	5
	Gbytes written	0.0008	0.022	0.42	1.7	6.8	135.8	2122.4
LU	grid size	12 x 12 x 12	33 x 33 x 33	64 x 64 x 64	102 x 102 x 102	162 x 162 x 162	408 x 408 x 408	1020 x 1020 x 1020
	no. of iterations	50	300	250	250	250	300	300
	time step	0.5	0.0015	2.0	2.0	2.0	1.0	0.5
SP	grid size	12 x 12 x 12	36 x 36 x 36	64 x 64 x 64	102 x 102 x 102	162 x 162 x 162	408 x 408 x 408	1020 x 1020 x 1020
	no. of iterations	100	400	400	400	400	500	500
	time step	0.015	0.0015	0.0015	0.001	0.00067	0.0003	0.0001
UA	no. of elements	250	700	2400	8800	33500	515000	
	no. of mortar points	11600	26700	92700	334600	1262100	19500000	
	levels of refinements	4	5	6	7	8	10	
	no. of iterations	50	100	200	200	200	250	
	heat source radius	0.04	0.06	0.076	0.076	0.067	0.046	
DC	input tuples	10^3	10^5	10^6	10^7			
	no. of dimensions	5	10	15	20			

Source: Extracted from NPB (2017).

Table 2.9: Problem sizes and parameters for each of the classes defined in NPB 3.3.

CHAPTER3: EXPERIMENTS AND RESULTS

In the following sections, the results obtained in the research will be discussed. It will be detailed the test environment, methodologies followed, the benchmarks used to perform the tests, statistical tests of hypotheses. In addition, the related works in the area will be presented, which were of great importance to acquire knowledge.

3.1 LARCC

The LARCC[1] (Laboratory of Advanced Researches for Cloud Computing) as its name suggests is a laboratory research for evaluation and experiments of cloud computing and related technologies. The core research in LARCC is about the performance of IaaS open source cloud management tools, such as OpenStack, CloudStack and OpenNebula. The front of the lab is showed in Figure 3.1. Moreover it has as a line of research the following topics:

- Cloud Computing

  Infrastructure as a Service (IaaS).

  Platform as a Service (PaaS).

  Software as a Service (SaaS).

- Distributed Systems (high performance, replication, high availability and redun-

[1] http://larcc.setrem.com.br/

dancy).

- Network programming (application protocols).
- Energy efficiency for data-center and private cloud.
- Data mining and machine learning for agriculture.

The research group is composed of nine members. Three with a doctoral degree, two with a master degree, two with a degree in technology and two with a degree in progress. This group is formed by three universities SETREM [2], PUCRS[3] and Unipampa[4]) which provide human resources for research. In addition, the LARCC has a collaboration with the companies ABASE[5] and Automasul[6].

Source: Baum, Maliszewski, Griebler, 2017.

Figure 3.1: LARCC - Laboratory of Advanced Researches for Cloud Computing.

Additionally, the HiPerCloud[7] is developed at LARCC. This project analyses and compares the clouds infrastructure, identifies behavior patterns of the tools relating them to the characteristics of the apps, creates customizable benchmarks to the company environment to predict cloud behaviors, identifies possible bottlenecks proposing efficient solutions and finally attends to proposal of new researches for future necessities of the corporate environment.

[2]http://www.setrem.com.br/
[3]http://www.pucrs.br/
[4]http://novoportal.unipampa.edu.br/novoportal/
[5]http://abase.com.br/
[6]http://www.automassul.com.br/site
[7]http://hiperfcloud.setrem.com.br/

3.2 RELATED WORKS

In this section is presented the related works that show the research already done and then a comparative table will be made to discuss the limitations and research methodologies used in each study or how this influences the proposal of this thesis.

3.2.1 Search Methodology

The entire process for searching related works is represented in the Figure 3.2. In this process, it was used the search engine from three of the largest and well-known libraries and a well-known search engine. In addition it was made the search string for each library and some rules and criteria were used for choosing the specific related works.

The formal logic expression was (performance and cloud and (nas or parsec) and (scientific or enterprise) and (application or workload or benchmark) and (characterization or evaluation or experiments or analysis or deployment) and (lxc or kvm)). The logic description is search for related papers which approach performance of cloud computing or virtualization through workloads, applications or benchmarks that through its results can be done characterization or evaluation or experiment or analysis or implementation in IaaS infrastructures or service models in native or virtual environment.

3.2.2 Search process for related work papers

In the next sections is described how the related work papers was searched in the libraries, criteria, rules, and finally the search strings logic.

3.2.2.1 IEEE

In this library, we use your own search engine. First, we enter in the IEEE[8] "Advanced Search" option and then in the command search. We checked the box "Full Text and Metadata" that allows the search in the abstract, title text, index terms and full text. Next, we created a search string with the main terms related to our search. Thus, making it possible to find specific related jobs.

3.2.2.2 Google Scholar

In this search engine, we first enter the advanced search in Google Scholar[9], and then in the item "find articles with all the words" we add the search string, the same one used in the IEEE library. Were shown around 600 results, and articles from the first 5 pages were chosen.

3.2.2.3 ACM

In this library, we use the specific string to ACM[10] environment, first we enter in the "advanced search" option and then in the option "Select items from" we selected "The ACM guide and computing literature". Next, in the options "Where", "matches all, matches any and matches none" "of the following words or phrases:" and then adding more conditions, we enter item by item of the string. In the end of the page of ACM there is a option "show query syntax", that shows the full string used.

[8]http://ieeexplore.ieee.org/Xplore/home.jsp
[9]https://scholar.google.com.br
[10]http://dl.acm.org/

3.2.2.4 Criteria

To apply parameters related to the search string in libraries and search engines, we've created a search criteria, which is defined as; Search terms of the string in the abstract, title text, indexing terms and full text.

3.2.2.5 Rules

For filtering the papers found we created some exclusion rules, witch help us to select the specific related work papers.

- The papers must be higher than the year 2011[11]. This is because older ones may have outdated information.
- Any paper that carries out research on energy efficiency, monitoring, storage evaluation, cost, green IT, SDN or migration will be discarded, precisely because this is not a work related to ours.

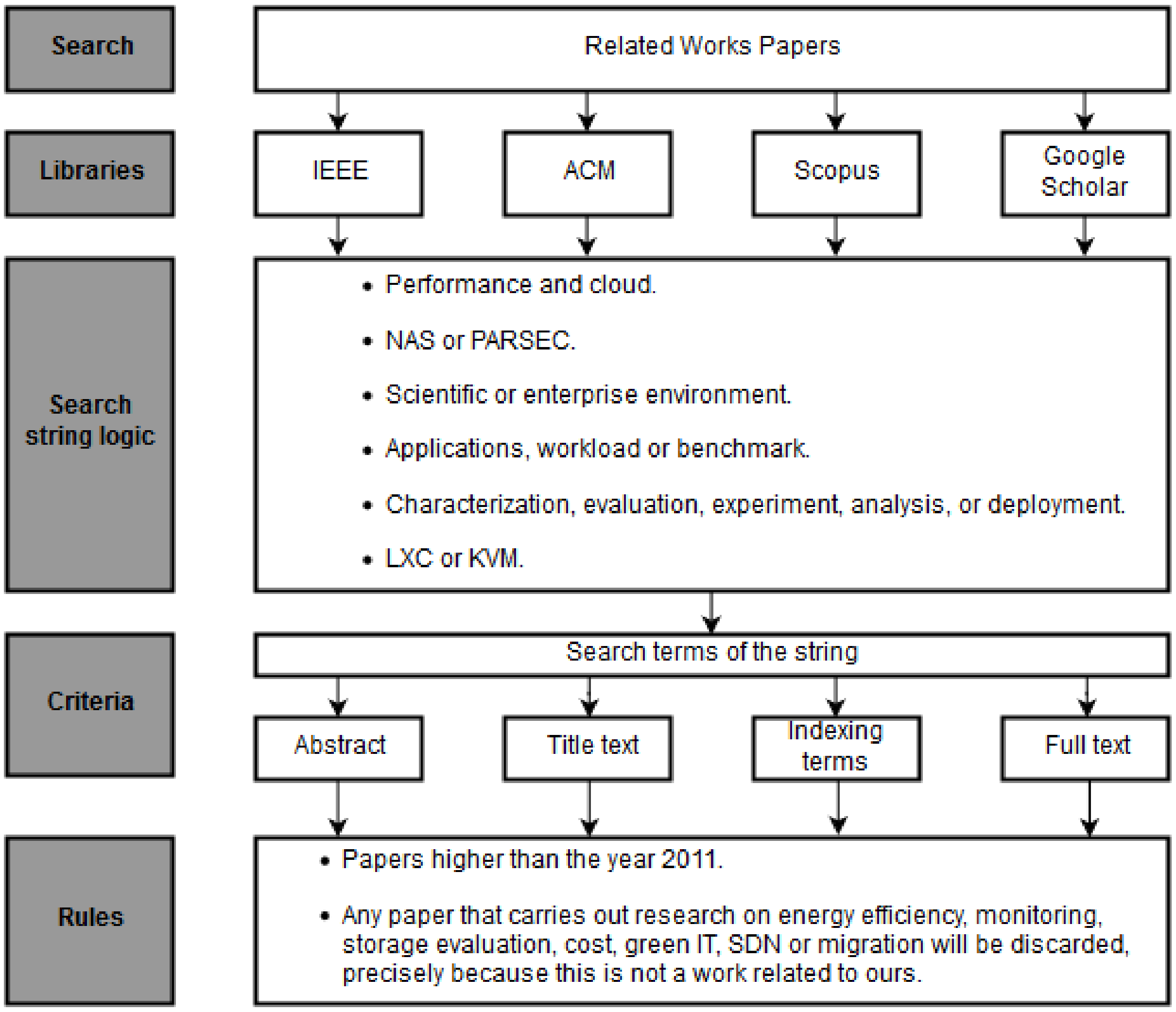
3.2.3 Search Strings Logic

As is known in the computer environment any research done on the Internet is conducted through a search string that is never seen by the user side. A search string is a combination of algorithms with the characters and words searched, which leads to the search for the correct results. In this work we have created our string to specifically conduct research on related works.

A search strings have to be created for each specific library because each uses a different way of searching. The logic of the sequence is the search for articles that contain the subject performance and cloud (key items), NAS or PARSEC (two

[11]There are some special cases that have proven to be very important, and need to be added.

suite of benchmarks) adding the scientific or business environment in which they must contain applications, workload or benchmark, and needs to approach characterization, evaluation, experiment, analysis, or deployment and finally uses LXC or KVM in the virtualization layer.


Search
Related Works Papers
Libraries
IEEE
ACM
Scopus
Google Scholar
Search string logic
• Performance and cloud.
• NAS or PARSEC.
• Scientific or enterprise environment.
• Applications, workload or benchmark.
• Characterization, evaluation, experiment, analysis, or deployment.
• LXC or KVM.
Criteria
Search terms of the string
Abstract
Title text
Indexing terms
Full text
Rules
• Papers higher than the year 2011.
• Any paper that carries out research on energy efficiency, monitoring, storage evaluation, cost, green IT, SDN or migration will be discarded, precisely because this is not a work related to ours.

Source: Baum, Maliszewski, Dalvan, 2017.

Figure 3.2: Search process for related works.

3.2.4 Presentation of Related Works

The paper presented by Vogel et al. (2016) deployed different IaaS tools and focused on parallel application benchmarking of the scientific domain. The tests were done using the NAS NPB on KVM instances on three different cloud platforms (OpenStack, OpenNebula and CloudStack). The results reveals that runtime among tools has been similar with the scientific workloads, proving that the tools can host such applications without compromising performance.

In the case of paper of Felter et al. (2015), the performance of traditional virtual machines deployments is explored and compared with the Linux containers. Focusing on the issue of overhead compared to the native environment. It was utilized workloads that can stress hardware and network resources using KVM (full virtualization) and Docker (container). The results reveals that the containers perform equal or better than KVM in almost all cases, but both virtualization technologies require an I/O improvement.

On the other hand, the article of Steinmetz et al. (2012) studies the performance of cloud computing platforms from the perspective of IT management using the Unix benchmark. Two separate clouds (Eucalyptus and OpenStack) were deployed on identical hardware, to verify which would have the least time to launch a VM instance (serialized and parallelized). The results showed a variation with different numbers of VMs that were launched. OpenStack has a much better performance to launch instance serially (3-6 seconds) compared to Eucalyptus (10-12 seconds). However, VM parallel launching time reveals that for launching one or two VMs, OpenStack decreases performance as a measure of instances were launched, at the time of launching three or more instances, Eucalyptus performs better.

The study of Barker et al. (2010) evaluates the real-world impact of various types of background interference, focusing on network issues. A variety of Amazon

EC2 features capabilities, technologies and performances from the It were compared. The results reveal that jitter and the throughput experienced by latency-sensitive applications have degraded performance varying from resource to resource, especially to the disk-bound tasks, which can be degraded by nearly by 75%.

The paper Huber et al. (2011) made a generic approach to predict the performance overhead of services running on virtualization platforms, such as Citrix XenServer and VMware. Passmark, SPEC and Iperf were the benchmarks used. The results showed that in the CPU and memory virtualization performance behavior is similar in both systems, as well as CPU scalability and overcommitment. However, the results also indicated that there is a deviation in the I/O virtualization and scheduling. Regarding performance, VMware is better including resource isolation than Citrix XenServer.

The evaluation of hypervisors were also done by Reddy and Rajamani (2014), different performance tests were conducted in XenServer, ESXi and KVM using SIGAR Framework, Passmark and Nimbus benchmarks. Utilizing CloudStack, the authors concluded that KVM needs a significant improvement compared to the other hypervisors, in order to be able to offer better performance.

The research conducted by Gupta and Milojicic (2011) argues that cloud platforms could be feasible to host HPC applications. To investigate, the authors performed NAS NPB, NAMD and NQueen benchmarks on different cloud platforms (Eucalyptus, Taub and Open Cirrus). The results reveals that Cloud is an compelling platform for some applications, specifically for non-communication intensive applications such as embarrassingly parallel and tree-structured computations up to high processor count and for communication-intensive applications up to low processor count.

The study of Mauch, Kunze and Hillenbrand (2013) provides an overview of the current state of high performance cloud computing and describes the underlying techniques and management methods. They focus on the AWS type following the

HPC2 model, presenting a novel approach to using high speed cluster interconnects such as InfiniBand in a high-performance cloud computing environment.

In addition, the paper of Roloff et al. (2012) conducts a detailed comparison of HPC applications running on three cloud providers (Amazon EC2, Microsoft Azure and Rackspace). The analyzed characteristics such as deployment facilities, performance and cost efficiency were sought and compared with a cluster machines. To do this, it was utilized OpenMP and MPI implementations of NAS, using XEN, Hyper-V and OpenStack. The results showed that HPC can run efficiently in the cloud, but the authors emphasize large differences among cloud providers, suggesting that behavior and applications types have different performance on the cloud scenary.

The paper of Morabito, Kjällman and Komu (2015) presented a detailed performance comparison of traditional hypervisors virtualization and container based. The experiments used the Y-crucher, NBENCH, Linpack, Bonnie ++, STREAM and Netperf benchmarks, running on LXC, Docker, OSv and KVM. The results showed that the containers performed better when compared to the virtual machines. In addition, the document highlights that the performance of KVM has been improved in recent years, making it more efficient by reducing the overhead.

The paper presented by Xu et al. (2014) analyzes performance in three cloud scenarios (single server virtualization, single mega datacenter and multiple geodistributed datacenters). They illustrate representative scenarios, discuss performance and cost-emphasizing methods, including mitigation techniques for overhead, on Amazon EC2. Finally, the results have brought a broad review and allow us to have a better understanding of the overload of the VM in IaaS clouds.

The paper of Jiang et al. (2012) studied the behavior of desktop cloud workloads and made a comparison evaluating a Xen-based virtualization platform. Desktops configured as a cloud were used to run fourth benchmarks (SPEC CPU2006, TCP-C, PARSEC and CloudSuite). The results demonstrate that there are significantly different benchmarking features, showing that desktop clouds are inefficient for this type of workload, with higher cache failure rates, and consequently undermining performance.

Performance evaluation was done by Xavier et al. (2013), performing several container-based virtualization experiments (LXC, OpenVZ and Linux-VServer) for HPC. The LINPACK, STREAM, IOzone, NetPIPE, NPB and IBS benchmarks were used to compensate performance and insulation by comparing containers with Xen. The study highlights that HPC will only be able to take advantage of virtualization if the overhead is reduced, it is also arguing that the containers have almost native hardware performance, having differences between them in the implementation of resource management. However, container-based systems are not mature yet because they do not have better isolation, but if HPC does not require resource sharing, a container-based can be attractive because of minimal performance overhead.

The article presented by Vogel et al. (2016) compares the OpenNebula, OpenStack, and CloudStack tools, assessing their differences in support for flexibility, resiliency and performance. The KVM hypervisor is deployed to perform the tests using intensive workloads, such as LINPACK, STREAM, IOzone, IPerf, and NAS. The study demonstrated that OpenStack is the most resilient, and CloudStack is the most flexible cloud for IaaS deployment. In addition, performance experiments showed differences in performance between tools when performing intensive workloads, including a small virtualization overhead. Therefore, concluding that private IaaS clouds are a good alternative for hosting scientific applications.

The study by Ogrizović, Car and Kovačić (2014) presented the economic and technical benefits of running scientific applications in cloud computing, and highlight some challenges for scientific applications strongly coupled with intensive communication. The authors argue that cloud computing is a viable environment for low and medium-scale scientific applications that are loosely coupled. It should be noted that for broad adoption, it is necessary to guarantee high bandwidth, low latency of interconnections with adequate isolation of virtualized resources.

The authors Jayasinghe et al. (2014) analyzed a variation in cloud performance and scalability, using the RUBBoS and Cloudstone benchmarks. The scenario presented a case where a multi-layered application is migrated from a traditional datacenter to one of the three Clouds IaaS (Amazon EC2, Emulab and Open Cirrus). Also used in the tests are the KVM, XEN and a commercial hypervisor. The results indicated that the best-performing configuration in one cloud can become the worst-performing configuration in another one. In addition, the study also identifies several bottlenecks at the system level that degrade performance, such as high-context switching and network overhead processing processing. As a consequence, there are significant performance variations between the three clouds.

The study of Scheepers (2014) compared the XEN and LXC as well as a discussion of its operational flexibility. The tests were conducted through the use of a request made by WordPress and PHP-scripts that are responsible for fetching data from the database and returning with a response. It was used JMeter to send an increasing number of concurrent requests until the server runs out of memory. The results showed better isolation of resources by XEN, but LXC improved performance as it introduces less overhead.

The study by Sadooghi et al. (2015) evaluates the performance of Amazon public clouds. Micro benchmarks such as CacheBench, Iperf, ping, tracerout and HPL were used to measure different instance types and were compared to a non-virtualized system to better understand the effect of virtualization. The study showed that in most cases the performance of instances is less than the expected performance that is claimed by Amazon. In addition, network latency is larger and less stable than supercomputers. Finally, scientific applications were run on Amazon EC2 and gross performance was compared with FermiCloud, which is well known as a high-end private cloud. The results revealed that the scientific applications evaluated in Amazon instances require a more powerful network capacity and better I/O performance. As a consequence, some cases are not suitable for hosting scientific applications.

The paper presented by Li, Xie and Zhang (2013), provides a performance comparison between three open source cloud platforms, such as Nimbus, OpenNebula and OpenStack in FutureGrid, through the measure of standardized Euclidean distance similarity. This work evaluates the contrasts of the overhead caused by virtualization to discover the most suitable platform. In this evaluation, the tests were performed using the KVM hypervisor and the HPC Challenge (HPCC) benchmark suite. The results indicate that the OpenStack platform has the best performance for HPC. Therefore, it is proposed to implement HPC applications on this platform.

The article presented by Kudryavtsev et al. (2012), explores the virtualization technologies used in the HPC environment. It provides a performance comparison between the KVM and the Palacios hypervisor, seeking to optimize them in relation to performance degradation and overhead. The HPC Challenge benchmark suite, NAS Parallel benchmark suite and the SPEC MPI 2007 benchmark were used to perform the tests. The results emphasized that KVM provides more stable and predictable results, while Palacios is much better at granulation tests Fine on a large scale but showing abnormal performance degradation in some tests.

The study presented by Beserra et al. (2015b) analyzed the performance of LXC (light virtualization) against KVM (hypervisor-based virtualization) in HPC activities. The experiment considers CPU and communication performance and uses the High Performance Linpack (HPL) tool and NetPIPE benchmarks. The purpose of this experiment was to determine how virtualization affects interprocess communication performance by considering two variables: first, communication using only intranode communication mechanisms and, secondly, communications using a physical network interface. In addition, it seeks to answer which scenarios LXC can deliver better performance than hypervisor-based virtualization or even the same as native environments for HPC applications. The results show that in an environment where physical resources are divided into multiple logical spaces, KVM does not perform well and LXC can be considered the most appropriate solution. Although the container-based solution is a good alternative to overcome the overhead problems coming from virtualization, the experiments also show that some problems especially in relation to LXC exist. LXC does not provide enough isolation, allowing guest systems to compromise the system host under certain conditions.

The article presented by Pflanzner et al. (2016), aims to provide test cases that can measure the performance of a private cloud (OpenStack), helping to find bottlenecks. In this experiment, three cases were examined: concurrency (execution of concurent scenarios), stress (stress to VMs with Phoronix benchmarks) and disk (varied disk sizes). The hypervisor used was the KVM. The research results reveal that the stress of a private cloud with targeted workloads introduces a certain degradation of performance, but the system returns to normal operation after the stressful load. In addition, failures have been noticed in certain cases, which means that new cloud deployments need to be improved for certain scenarios. The performance of general-purpose scenarios in a private cloud provides insight for business stakeholders planning to move to the cloud and provides suggestions where further development is needed in OpenStack.

The experiment presented by Expósito et al. (2013), provides an evaluation of high-performance messaging computing middleware in a cloud computing infrastructure, Amazon EC2 cluster computing instances, equipped with 10 Gigabit Ethernet. To analyze the impact of cloud network overhead on representative HPC codes, a test platform with similar hardware was built. This evaluation consists of a microbenchmarking of point-to-point data transfers, both between VM (through 10 Gigabit Ethernet) and intra-VM (shared memory), at the level of the message passing library and its underlying layer, TCP/IP. Next, we evaluated the significant impact of virtualized communication overhead on the scalability of representative parallel codes using the NAS Parallel Benchmarks (NPB) with class B. The result shows a significant impact that virtualized environments still have on the performance of communications. This requires more efficient communication middleware support to overcome the current limitations of the cloud network, such as replacing the TCP/IP stack on high-speed Ethernet networks.

The article presented by Paradowski, Liu and Yuan (2014), aims to analyze current performance research with regard to cloud computing at different levels and fill a gap in research on the specific area of benchmarking the performance of two open source cloud platforms , OpenStack, and CloudStack. This article used VirtualBox as a hypervisor and the benchmark used consisted of the time cloud platforms take to create and delete instances of different sizes. In addition, they evaluated the use of hard disk and CPU during management operations (creating and deleting instances). The result can clearly identify that OpenStack performance outperforms CloudStack when measured against benchmarks. This is evidenced and summarized throughout the paper tests where not only OpenStack performs tasks in less time than the CloudStack in all tasks, but also showed a more relative correlation in its results, particularly in relation to hard disk sizes.

The article presented by Zhang, Lu and Panda (2016), provides a container-based virtualization performance (Docker) and KVM hypervisor with PCI passthrough and SR-IOV for HPC on InfiniteBand clusters. In this evaluation, the Graph500, NPB, LAMMPS and SPEC MPI 2007 benchmarks were used. The methodology used in the evaluation considered that the KVM was used as the Virtual Machine Monitor (VMM). A single VM was deployed per node because the PCI passthrough can only have one VM with the IB device. Each VM has 24 cores, 32 GB of memory and the same operating system as the host operating system. HCA was attached to VM by PCI passthrough and SR-IOV technology, respectively, which are presented as 'VM-PT' and 'VM-SR-

IOV'. On the container-based virtualization side, Docker 1.8.2 was deployed as the mechanism to build and run the Docker containers. To have a fair comparison, we also performed single container at each node. The results indicate that VM with PCI passthrought outperforms VM with SR-IOV, while SR-IOV allows efficient sharing of resources. In general, the container-based solution can deliver better performance than the hypervisor-based solution (KVM). Compared to native performance, the PCI passthrough container only incurs up to 9% overhead in HPC applications.

The article presented by Chakthranont et al. (2014), provides a comparison of performance in a private cloud with CloudStack, specifically measured in a physical cluster against a virtual cluster. This evaluation used the KVM hypervisor and the Intel Micro Benchmark (IMB), HPC Challenge (HPCC), OpenMX and Graph 500 benchmarks. The results shows that almost all MPI collective functions suffer from scalability. Fortunately, the negative impact is limited to benchmark application-level programs where the virtualization overhead is around 5%, even as the number of nodes grows to 128.

The paper presented by Varghese et al. (2016) explores lightweight cloud benchmarking techniques - processes that run quickly and can be used in near real-time to collect metrics from cloud providers and VMs. The exploration of lightweight benchmarking techniques is facilitated by the development of DocLite - Docker Container-based Lightweight Benchmarking. DocLite is built with Docker container technology, which allows a user-defined portion of the VM to be compared. DocLite operates in two modes, in the first mode, containers are used to compare a small part of the VM to generate performance ratings. In the second mode, the reference historical data is used in conjunction with the first mode as a hybrid to generate VM rows. The classifications generated were evaluated against three high-performance scientific computing applications. Lmbench was deployed as the benchmarking tool and the cloud platform was the Amazon EC2. The results indicate that the proposed techniques are up to 91 times faster than the heavyweight technique, which marks the entire VM. It is observed that the first mode can generate scales with more than 90% and 86% accuracy for sequential and parallel execution of an application. The hybrid mode improves the correlation slightly, but the first mode is sufficient for benchmarking cloud VMs. The paper of Huang et al. (2013), compared a variety of features, technologies and performance, as well as analyzing the performance of virtual machines for geoscience applications. The cloud platform chosen by the authors were OpenNebula, Eucalyptus, and Cloudstack. In addition, XEN and KVM virtualization were used to performing the benchmarks UBench, Bonnie++, LLCbench, Iperf, to stress CPU, HD, memory and network. The tests show that there are no significant differences in computing resources between the platforms analyzed, but OpenNebula has the fastest network, while Eucalyptus and CloudStack perform better in VM isolation. Moreover, CloudStack has the fastest cloud management operation (VM deployment, images, snapshots and network). The tests also reveal that the cloud platform is not a good choice for intensive large-scale communication applications, requiring traditional HPC to be improved for virtualization environment, in order to achieve a better performance.

The authors Gupta et al. (2013), designed and implemented an HPC-aware scheduler in Nova Compute, a component of OpenStack, to simultaneously deploy VMs and also incorporate CloudSim to simulate the cloud environment. The study benchmarked NPB, NAMD, ChaNGa and Jacobi2d, using KVM hypervisor, to characterize the applications and identify the similarity behavior of the applications. The studied has characterized the challenges involving VMs and HPC in the cloud, demonstrating the interference between the platform, and also contributed to performance improvements to such an environment.

On the other hand, the study of Ghoshal, Canon and Ramakrishnan (2011a), focused on benchmarking I/O performance over different cloud and HPC platforms, identifying bottlenecks and performance issues. Therefore, two cloud platforms (Amazon and Magellan test bed), with KVM hypervisor and IOR Timed Benchmarks were used to perform the tests. The results showed that I/O performance can be one of the main causes for performance issues in virtualized cloud environments, evidencing that the abstraction layer between the hardware and the VM induces these bottlenecks. Moreover, the tests made on local storage performs better in comparison with block storage. The authors appointed that this happens because block storage volumes can be multi-instance mounted and used as a shared file system, which degrades the performance, especially for latency sensitive applications such as MPI.

The paper of Gupta et al. (2013) proposes a dynamic load balancer for HPC applications in the cloud, focusing on the interference introduced by multi-tenancy. The cloud platform used was the OpenStack with KVM hypervisor, to run Stencil2D, Wave2D and Mol3D benchmarks. Thus, 64 VMs were used to simulate multi-tenancy and the results discuss the effect of load balancing frequency and different problem sizes that affect scalability and performance. The load balancer presented by the authors adopts different dynamic variations to use cloud features and presents a proposal that demonstrates that the performance for such operations can be improved up to 45%.

The study made by Ruiz, Jeanvoine and Nussbaum (2015) evaluates the isolation and performance using container technologies in different Linux kernels, running HPC workloads (NPB and TAU). Two different tests were performed, one using containers hosted on the same machines and others hosted on different machines, both to verify communication between containers. The study does not run the benchmarks on the cloud platform, focusing only on containers and HPC workloads. The results show the container usage limits, revealing that oversubscription does not significantly affect application performance. In addition, container overhead exists, but newer versions perform better than old ones, revealing that container technology is getting mature as new versions are released.

The paper Gupta et al. (2013) presented a holistic viewpoint to answer the question “Why and who should choose cloud for HPC, for what applications, and how should cloud be used for HPC?". The authors used the Jacobi2, NAMD, ChaNGa, Sweep 3D, Nqueens benchmarks in KVM, LXC and Bare metal virtualization. For this, different cloud platforms were used, such as, Eucaliptus, Open Cirrus, Taub, Ranger, and a public cloud provider. Therefore, the presented test-bed allows to simulate workloads on physical machines, containers, multi-tenancy and also network performance test. The results reveals that interference from virtualization and multi-tenancy can significantly affect performance-related HPC applications, including those that are network-sensitive, highlighting the need for low latency requirements for such workloads, to avoid performance degradation. Also, in terms of isolation, public clouds are profitable only on small-scale HPC workloads.

Assuming that the cloud computing is suitable for some HPC applications, the study made by Gupta and Kale (2013), compares the cost-benefit ratio of the cloud to typical dedicated HPC platforms to propose a technique for cloud providers to improve their business. Therefore, the tests were done on the Open Cirrus testbed to simulate the cloud environment, in the HP Lab Site and scientific applications NPB, ChaNGa, Charm++ were benchmarked on KVM virtualization. Through the characterization of the applications running, a workload was modified for better adapt to cloud computing (HPC-aware) and also the opposite, that is, modifications were made in the cloud to better suit HPC (HPC cloud-aware). It has used heuristics to analyze and select a suitable platform for each specific workload, and propose a dynamic load balancing. The results demonstrate that techniques improves performance, throughput and reduces the cost. Thus, demonstrating that HPC performance can be significantly improved to run in the cloud environment.

The authors Zhou et al. (2013) identify and analyze CPU scheduling problems in hypervisors, and propose a scheduling algorithm (Poris), and also consider synchronization problems and real-time constraints. They used two machines with different configurations, to evaluate STREAM (Non-real-time workloads), PARSEC, Media Player and MyConnection server (VOIP). These benchmarks were run on the Xen hypervisor to design parallel soft real-time scheduling algorithm, and Poris that treats VMs as non-real-time. The paper identified and analyzed scheduling problems in virtual machine monitor (VMM), and the results reveals that Poris can improve the performance of parallel soft real-time applications, media player and shortens the execution time of PARSEC.

The paper of Mulerikkal and Sastri (2015) made a comparative study, focusing on the complexity of implementation, stability, and performance of OpenStack and CloudStack, using benchmarks Unixbench and Bonnie++. Clouds were deployed in the same machine configurations to characterize the complexities of implementations and conduct a performance evaluation. The results showed that OpenStack outperforms CloudStack in most scenarios in a single node environment, has overall better stability, but installation is more complex. In addition, the authors have suggested that OpenStack is more attractive because it has the most community support and business initiatives.

The paper of Xavier et al. (2015) analyses the performance interference suffered by disk-intensive workloads on container-based clouds. To run the experiments, Swingbench and the Sysbench benchmarks were used in Oracle and MySQL databases. Tests were conducted on the OpenStack platform, with LXC and KVM virtualization technologies, on two identical Dell PowerEdge R810 machines. The results showed that LXC does not provide complete isolation of resources like KVM, inducing degradation of performance in certain disk-intensive workloads. In addition, the authors mentioned that the container-based system is note mature yet to this type of intensive workloads, and it is a good alternative implement different workloads types, especially I/O and CPU-intensive to reduces the interferences, instead I/O and memory intensive workloads.

The authors Evoy, Mury and Schulze (2014) made a study of scientific applications in virtual clusters by analyzing how different VM deployments affects the performance of parallel applications that uses distributed memory. It was utilized a software for definition and installation of virtual clusters, called VALPA, which provide the necessary environment to simulate the effect of the topology and VM size, and perform the systematic execution of the virtual clusters. The benchmark used was the Parpac responsible for the characterization of parallel MPI applications, in KVM virtualization. The main findings are that virtual core mappings and VM spreading can have significant impact on performance, and also the study provided a deployment guideline that can increase the performance about 6-11% of one VM per node.

An extended examination of the feasibility, performance and scalability of the scientific and engineering applications was made by the authors Saini et al. (2012). The study compares Nebula performance using KVM virtualization with HPC NASA's Pleiades, using the Network Benchmark (NUTTCP), HPC Challenge Benchmarks (HPCC), MPI Function Benchmarks (MFB), I/O Benchmark Sequential Read/Write (SRW), NPB, Overflow, CART3D, USM3D, MITgcm (MIT General Circulation Model) and also investigating the performance of virtIO and jumbo frames. The results indicate that in Nebula, virtIO and jumbo frames improve network bandwidth in 5 times, but there is a significant overhead of virtualization layer (10-20%), and the write performance is 25 times inferior compared to the native environment. In addition, latency for short MPI messages is very high, and the overall performance is 15% to 48% lower than HPC Pleiades. The study also contributes to quantify virtualization overhead, analyze the impact of virtualization, I/O and jumbo frames on performance, helping to identify the limiting factors in the cloud platform.

The paper of Kerbyson et al. (2014) made a performance analysis of Cray XT/XE, IBM Blue Gene and systems using InfiniBand, therefore three architectures for high performance computing were used. The impact of the operating system was analyzed in each core using the P-SNAP benchmark as reference, for memory was used the STREAM benchmark and also four applications (DNS3D, MiLC, GTC, and Nek-Bone) to examine the network performance. The analysis brings an individual characterization of each system using the proposed applications and benchmarks, and also made a quantification of the performance lost in each system caused by inter-job interference.

The study performed by Mehrotra et al. (2012), compared the performance characteristics of Amazon EC2 HPC instances with NASA's Pleiades supercomputer. The NPB (NAS Parallel Benchmarks) was used along with four scale applications to represent that are currently used by NASA scientists and engineers. The results reveals that while Amazon has made great strides in offering HPC resources, the results were much lower compared to the Pleiades. The authors highlights that for scientists and engineers, the cloud systems such as presented, do not provide the environment they require.

A performance evaluation of NPB on intel Xeon Phi processor was performed by Ramachandran et al. (2013), comparing the execution time with the traditional Intel Xeon processor, using the Intel Parallel Studio XE 2013 and V Tune Amplifier. The results indicate high memory latency of Xeon Phi, caused by the inefficiency of Gather-Scatter instructions. In addition, some common issues that can lead to bottlenecks from both processors have been identified.

The authors Leite et al. (2012) present a study on technologies for cloud computing focused on virtual machines performance. The tests were conducted using the Phoronix Test Suite to verify the virtualization overhead and performance issues between Xen and KVM. The results reveals that KVM achieves the best responsive time and performance is better.

The study of Campos et al. (2015) characterizes the performance instantiation of VMs in the Eucalyptus private cloud, thus focusing on the elasticity. The authors developed a script to instantiate the VMs repeatedly one time after another and measure the times of each instantiation phase, considering different sizes of virtual machine images as well as cache. The experiments were only created on the front-end node with the argument that VM instantiation is a process involving this particular node where the VM is allocated. The results show that the cache has a significant impact on instantiation performance (45,07%), followed by machine image (26,54%) and VM type with 1,05%.

A theoretical performance model for predicting the performance of parallel applications was presented by Hong et al. (2014), and tests were done using different virtual machines scheduling on KVM, Xen and VMware. The authors developed a micro benchmark to measure the overhead and to evaluate the predicted execution time of a parallel program was used the NPB. Thus, for experiments a target VM executing each child benchmark was performed individually on hypervisors. The results help to characterize the performance of individual NAS workloads and specially the proposed model could predict the performance of parallel applications in various scheduling policies.

A performance evaluation was performed by Hashimoto and Aida (2012) focusing on applications programs running on virtual machines. The authors benchmarked NPB, MySQL and Sysbench by running two application programs simultaneously to evaluate performance in each application. The experiments results indicate interference among VMs executing two HPC applications on the physical server. However, they can mitigate the interference effect by selecting HPC applications that require low resources, such as LU with EP or IS.

An in-depth study was conducted to analyze performance degradation in virtualized environment by Nikounia and Mohammadi (2015). The authors have configured four VMs using KVM virtualization, and also considering overcommitment, which is a practice adopted by IaaS cloud providers to allocate more than one vCPU per each physical core. Thus, VMs ran 12 Parsec benchmark applications simulating multi-tenancy to verify interference. The results reveals that performance degradation in the virtualized environment presented can be up to 16x in some cases. The main factors that led to poor performance were: blindness of hypervisors scheduler, cache contentions, context switch decisions and the lack of special instructions such as SSE, which is not ported to the guest OS by the hypervisor.

The main goal of the authors Coutinho, Paillard and Souza (2014) was to demonstrate the feasibility of using a local cluster built in a private cloud for HPC, comparing with applications running in real HPC environment. For this, the private cloud chosen for the experiments was OpenNebula and the KVM virtualization was used to instantiate the VMs. The experiments utilized the Intel MPI benchmark as well as MPI-based bioinformatics workloads, such as mpiBlast and ClustalW-MPI. The results were favorable to the utilization of scientific applications analyzed in cloud computing, and also the study helps to characterize the environment.

The aim of study conducted by Zhang, Lu and K. (2016) is to characterize the performance of two virtualization solutions (KVM and Docker) and two virtualized I/O technologies (PCI passthrough and SR-IOV) using infiniband. For this, it was utilized the Graph500, NPB, LAMMPS and SPEC MPI 2007 benchmarks, running on Chameleon cloud technology. The tests results showed that the container-based solution and PCI passthrough, performs better than the hypervisor-based solution and SR-IOV.

The paper of Beserra, Endo and Barreto (2016) investigates which scenarios LXC can perform better than KVM in relation to I/O operations. The fs test was used as a benchmark and executed in a scenary with and without communication between processes to make a comparison. The results reveals that LXC offers a better performance than KVM in I/O-bound applications, and also in shared resources among multiple abstractions hosted on the same host.

NAS Parallel Benchmarks was used by the authors Okada, Goldman and Cavalheiro (2016) to estimate the performance of actual HPC applications in the cloud, based on their communication characteristics. They compared execution behavior in Google Compute Engine, OpenStack using KVM virtualization and a NUMA multiprocessor system using LXC container. The authors conclude that HPC users should use the appropriate number of vCPUs on each VM, avoiding the overcommitment of resources because application performance may be affected by scheduler and Hyper-Threading issues.

3.2.5 Analysis and Comparison of Related Work

Name	Cloud Platform				Virtualization			Benchmarks		
	CloudStack	OpenStack	OpenNebula	Other	KVM	LXC	Other	NPB	PARSEC	Other
Vogel et al. (2016)	✓	✓	✓		✓			✓		
Felter et al. (2015)					✓		✓			
Steinmetz et al. (2012)		✓		✓						✓
Barker et al. (2010)				✓						✓
Huber et al. (2011)				✓			✓			✓
Reddy and Rajamani (2014)	✓				✓		✓			✓
Gupta and Milojicic (2011)				✓				✓		
Mauch, Kunze and Hillenbrand (2013)				✓						
Roloff et al. (2012)				✓			✓	✓		
Morabito, Kjällman and Komu (2015)					✓	✓	✓			✓
Xu et al. (2014)				✓						
Jiang et al. (2012)				✓			✓		✓	
Xavier et al. (2013)						✓		✓		✓
Vogel et al. (2016)	✓	✓	✓		✓			✓		✓
Ogrizović, Car and Kovačić (2014)										
Jayasinghe et al. (2014)				✓	✓		✓			✓
Scheepers (2014)						✓	✓			✓
Sadooghi et al. (2015)				✓			✓			✓
Li, Xie and Zhang (2013)	✓	✓		✓	✓					✓
Kudryavtsev et al. (2012)					✓		✓	✓		✓
Beserra et al. (2015b)					✓	✓				✓
Pflanzner et al. (2016)	✓				✓					✓
Expósito et al. (2013)				✓			✓	✓		
Paradowski, Liu and Yuan (2014)	✓	✓					✓			✓
Zhang, Lu and Panda (2016)					✓		✓	✓		✓
Chakthranont et al. (2014)	✓				✓					✓
Varghese et al. (2016)				✓			✓	✓		✓

Source: Baum, Maliszewski, Griebler, 2017.

Table 3.1: Comparison and contrast with Existing Studies 1.

Name	**Cloud Platform**				**Virtualization**			**Benchmarks**		
	CloudStack	**OpenStack**	**OpenNebula**	**Other**	**KVM**	**LXC**	**Other**	**NPB**	**PARSEC**	**Other**
Huang et al. (2013)	✓		✓	✓	✓		✓			✓
Gupta et al. (2013)		✓			✓			✓		✓
Ghoshal, Canon and Ramakrishnan (2011a)				✓	✓					✓
Gupta et al. (2013)		✓			✓					✓
Ruiz, Jeanvoine and Nussbaum (2015)				✓			✓	✓		✓
Gupta et al. (2013)				✓	✓	✓	✓			✓
Gupta and Kale (2013)				✓	✓			✓		✓
Zhou et al. (2013)							✓		✓	✓
Mulerikkal and Sastri (2015)	✓	✓								✓
Xavier et al. (2015)		✓			✓	✓				✓
Evoy, Mury and Schulze (2014)					✓					✓
Saini et al. (2012)				✓	✓			✓		✓
Kerbyson et al. (2014)				✓			✓			✓
Mehrotra et al. (2012)				✓				✓		
Ramachandran et al. (2013)								✓		
Leite et al. (2012)					✓		✓			✓
Campos et al. (2015)				✓						✓
Hong et al. (2014)					✓		✓	✓		✓
Hashimoto and Aida (2012)								✓		✓
Nikounia and Mohammadi (2015)					✓				✓	
Coutinho, Paillard and Souza (2014)			✓		✓					✓
Zhang, Lu and K. (2016)					✓		✓	✓		✓
Beserra, Endo and Barreto (2016)					✓	✓				✓
Okada, Goldman and Cavalheiro (2016)		✓			✓	✓		✓		
Our work	✓	✓	✓		✓	✓		✓	✓	✓

Source: Baum, Maliszewski, Griebler, 2017.

Table 3.2: Comparison and contrast with Existing Studies 2.

3.2.6 Related Works Discussion

Through the related works presented, it is possible to realize that there is an effort from the scientific community involved in researching and evaluating the different technologies that compose the cloud computing. Not all technologies are entirely new, but they are the result of combinations of features and resources that can provide this deployment, as is the case of containers. In this case, it is hot topic and it is a technology that is still evolving and requires more research because it can produce evidentially good results, but at the cost of the others. This lightweight virtualization brought by the containers, has made companies like VMWare and Microsoft improve their technology to compete fairly in this fast-growing market, forcing those companies to launch their own container technologies. With one eye in this market too, the Canonical improves the LXC releasing the LXD, which is a daemon that brings new features to the LXC, including the possibility to live migrate VMs and more scalable[12].

This is a multi-millionaire market, and by all means, probably more improvements in the current technologies will appear sooner or later. In addition, there is a major community effort dedicated to characterize and search for the best technology for each cloud scenario. Moreover, the domain of HPC applications frequently used in the scientific and enterprise fields, often requires prohibitive financial investments to deploy a supercomputer, so these fields demand further research to prove the viability required for applications to be ported to the cloud without compromising performance, representing a financially sustainable option.

However, as it can be seen in the Tables 3.1 and 3.2, there is still research needed in this area to meet scientific and enterprise needs. The majority of the studies only comprehends the cloud deployment using one instance and focus to evaluate the differences between virtualization technologies by applying a limited number of workloads. This is harmful because cloud computing is a complex environment that focuses

[12]https://linuxcontainers.org/lxd/introduction/

on shared resources and ignoring this fact and especially, adding more instances, can increase the concurrency for resources between users and lead to a poor experience. This is the case of the proposed multi-tenancy environment, where instances are deployed and workloads are used to simulate noisy neighbors that can affect the applications behavior in others tenants. Unfortunately, there is a limited number of papers focused on this needs, which is crucial to the success of cloud deployment.

In addition, another point that is usually overlooked is the fact that cloud computing uses some underlying technologies that can also affect the applications performance. One example is the utilization of compute node responsible for hosting the instantiated users, which could introduce, some performance issues in relation to the communication with the front-end node. Therefore, an extended research is necessary to avoid such problems.

In this work, we seek to characterize a diverse number of applications of the scientific and enterprise fields, which we believe best represent these domains. By exposing these workloads into the virtualized cloud environment using two different virtualization technologies, we contrast to the others works by adding the multi-tenancy environment, as well as running applications in the multi-node cloud configuration, which differently characterize the environment. Thus, by creating this complex cloud environment, we expect to contribute to the others studies in the area.

3.3 EXPERIMENT PLAN

This research intends to investigate the applications performance in the virtualized cloud computing environment, to do this, is important to configure the experiments that are able to simulate the production environment. This is necessary because many different technologies and features are involved, which can lead to different results. Thus, in order to deliver more accurate results, the experiment plan is designed to define parameters to set up the cloud environment, resource provisioning and Instantiable

VMs that will host the applications.

Therefore, the tests were performed in the LARCC at SETREM, using CloudStack 4.8 as cloud platform tool, the main technologies and features are best described in Section 2.2.9.1. The cloud used is composed of four server computers with the same configuration, running Ubuntu 14 OS on Intel Xeon X5560 processor, 24GB RAM (1333MHz), Sata II storage disk and gigabit network on each node. The cloud structure is composed of the front-end node (responsible for cloud administration) and three computing nodes (responsible for running the experiments). The primary and secondary storages are mounted on the front-end node and use the NFS protocol to communicate and share resources between nodes, being responsible for storing the VM images, templates and SO images.

As mentioned before, the native environment uses Linux Ubuntu 14 installed on respective compute node as well as the VMs offered by virtualization technologies managed by CloudStack. In addition, it is important to highlight that, among other things, CloudStack is responsible for providing the environment conductive to virtualization technologies, managing the allocation of resources used by LXC and KVM based-clouds, that is, whenever a cloud is deployed, a compute node will be designated to provide such virtualization technology. This means that a computer host can only provide one type of virtualization at the moment, so the tests were performed using one virtualization technology and upon completion of the tests, the host became available to deploy another virtualization technology.

To characterize the applications performance, the first step is to know how they behave in the native environment by benchmarking them. Making possible to compare with different virtualization technologies. Therefore, in order to not interfere with the proposed test scenario, we used a program[13] to trace the use of hardware, submitting the execution of the applications in a specific scenario mounted for monitoring

[13]https://github.com/dalvangriebler/upl.git

purposes only, collecting relevant information for analyses. Thus, the memory usage, cache miss, core load and disk I/O data were collected from the 22 applications (9 from NAS and 13 from PARSEC). Moreover, it serves as a baseline not only to compare with the virtualized cloud environments, but also to characterize performance by scheduling threads.

Although the intention was to represent IaaS cloud behavior in the production environment, we avoid adopting the overcommitment practice adopted by some cloud providers. Overcommitment in according with Nikounia and Mohammadi (2015) and Huber et al. (2011), is a practice used to provide more resources to users than actually exists, such as a cloud provider that offer 16 vCPUs among the users but in fact it has only 8 vCPUs. In according with Mukhedkar, Vettathu and Chirammal (2016), there is nothing wrong with this practice, even though it may provide better use of resources, but in certain cases performance could be significantly compromised. Therefore, we assume that the cloud provider has the exact service provision at disposal.

Different users configuration was used to create an even more realistic environment. In fact, we perform experiments combining a total of four users, running the same and different applications among them, in order to simulate the concurrency on the multi-tenancy environment. This was made possible by configuring different service offerings in CloudStack, so the hardware resources from the compute node were splited between running users. However, in the case of native environment the hardware resources were not divided as was done in virtualized, instead we created four users, then the performance could serve as a baseline, facilitating the comparison among the studied environments.

The execution time of an application may vary for many reasons, for example, one instance can complete its tests after another, and thereby release hardware resources before the another. In order to mitigate the variation of runtime and give more confidence to the experiments, the tests were performed 15 times on each instance,

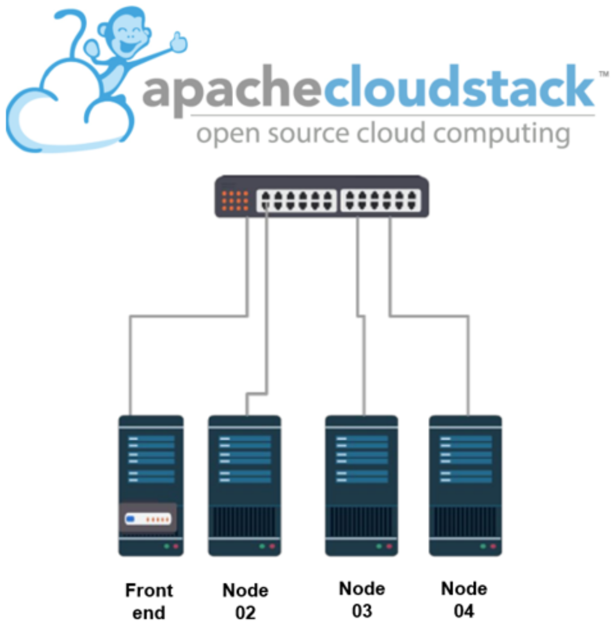


Figure 3.3: Physical environment

and average running time of the first 10 executions were extracted. Another care has been taken in the procedures, we avoid repeatedly executing the same applications to prevent the benefit of caching. Therefore, we have developed a script to execute the workloads in a sequence by scheduling from one to eight threads, thus all applications initially run with one thread and, after the one-thread execution is finished, the second-thread execution is started, and so until it reaches eight threads.

In all VMs and containers versions, as well as the native environment, the latest stable versions of the benchmarks were downloaded from the official project developer pages. So, PARSEC 3.0 and NPB 3.3.1 versions were used. Thus, node 2 was responsible for running the PARSEC experiments and the node 4 was chosen to perform NPB experiments. Unfortunately, the compute node 3 has been deactivated by mal functioning due to sudden power loss during the initial stages of the tests and cannot be repaired. The physical environment is also shown in the Figure 3.3.

Finally, in order to better characterize and discuss the results, a deep investigation was conducted based on cited paper and books that served as a reference.

However, it is important to highlight that, despite the fact that many referenced papers use the same applications suites, they did not perform their tests using dedicated computing nodes, neither multi-tenancy users that are typically offered by a cloud provider. Therefore, not all answers were found in the researched literature and many conclusions are derived from our experience and knowledge acquired so far.

3.3.1 Enterprise Applications

The PARSEC benchmarks suite was used to represent real-world applications used in many areas, such as computer vision, backup, similarity search and data mining, which are best described in Section 2.3.1. Then, in order to have a more realistic experiment, all applications ran using their native input, which has larger working sets and represents the full workload operation. Thus, we benchmarked the applications individually in the native environment by increasing the threads, what makes it possible to analyze the behavior of each individual application. Therefore, graphs with performance characterization of the main hardware resources used were plotted.

All applications, except for Freqmine, were run using their PThreads implementation because it is a frequently used parallel library in the HPC domain and offers a good level of parallelism with relatively low overhead. Freqmine requires the OpenMP parallelization model due to the algorithmic structure present in this application.

In the case of application Fluidanimate, we decided to omit it from our tests because, according to Southern and Renau (2016) and Bhadauria, Weaver and McKee (2009) the non-power-of-two threads configurations are unsupported. Similarly, it occurs with the Facesim workload on the fifth and seventh threads, but in this case we have decided to keep this application and only omitting the problematics threads.

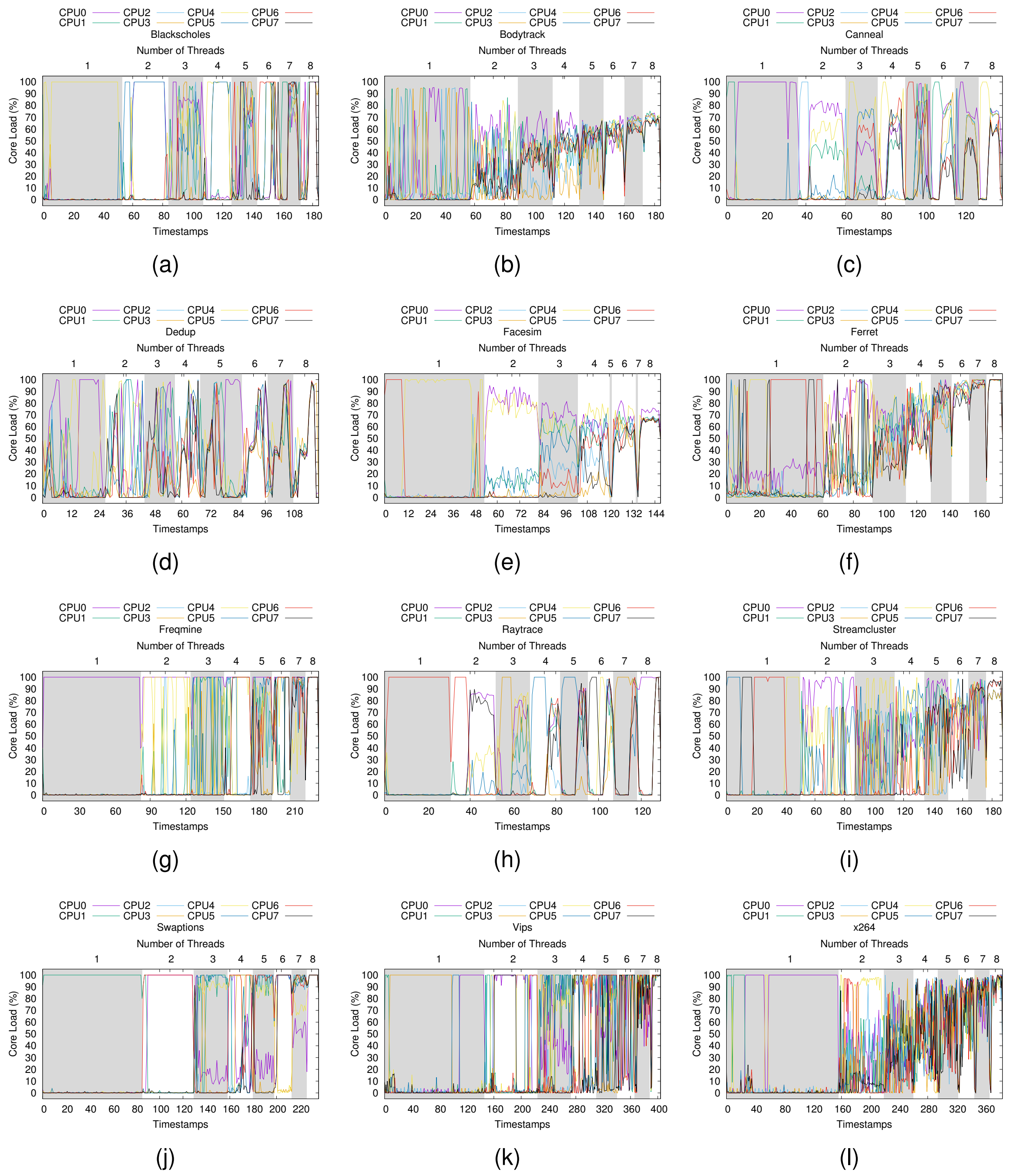
3.3.2 Performance Characterization

The workloads presented in the Parsec benchmark suite are from different areas, therefore they have different algorithmic structure and also different hardware needs, so they are expected to behave differently from each other. Therefore, to better discuss the results, a trace application was used for monitoring purposes in a separated native scenario to characterize the performance without the virtualization overhead. The results will be discussed below.

3.3.2.1 Core Load

The Figure 3.4 represent the core load utilization of eight CPU cores (four physical and four logical) offered by the Intel Xeon X5560 processor. This is an extremely important test so we can better understand the applications stages and correlate with the CPU load. As a consequence, therefore, we can also realize the specific characteristic of the applications, scheduling in eight threads. The first detail we can realize is that not all applications are CPU-bound, this became clear when all 8 threads are used. Not all core load is used as in the case of Bodytrack (3.4(b)), Canneal (3.4(c)), Dedup(3.4(d)) and Facesim (3.4(e)). The other applications have high CPU demand and have spent almost the entire application time using full processor capacity.

It is also important to note that in all applications as the number of threads increases, the execution time is also improved. However, in some cases, a small difference is noticed, such as the case of Bodytrack (3.4(b)), Canneal (3.4(c)), Dedup (3.4(d)), Facesim (3.4(e)) and Raytrace (3.4(h)), this is because not all stages of these applications are fully parallelized and adding more threads does not improve performance, especially when the application has some serial stages, incurring dependency on a segment (BIENIA, 2011). In addition, threads are mainly affected due to memory latency or synchronization, causing them to stall, which leads to performance bottlenecks (BHADAURIA; WEAVER; MCKEE, 2009). Dedup (3.4(d)) has an interesting case in the fifth thread, we can notice that the CPU0 line acts on serial part of the program and the others cores are waiting because they are dependent for this job, occurring a bottleneck (NAVARRO; ASENJO; TABIK; CASCAVAL, 2009).



Source: Baum, Maliszewski, Griebler, 2017.

Figure 3.4: Performance Characterization of Enterprise Applications (Core Load).

3.3.2.2 Memory Usage

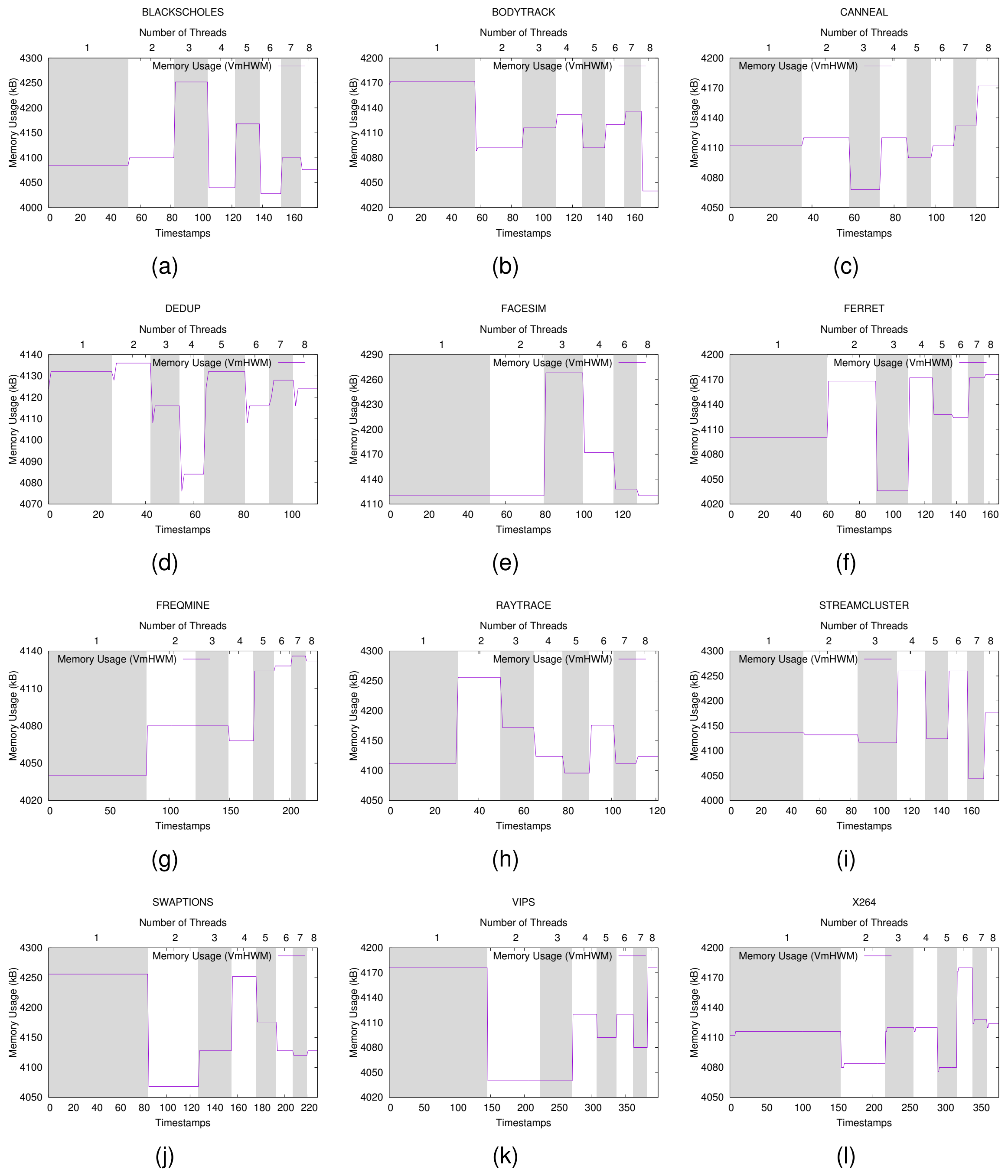
The node configuration has 24GB of RAM, so a fact that requires attention is that all applications use almost the same amount of memory, varying from 4 to 4,2GB. Although the expressive variations in the plotted graphs (Figure 3.5), the scale is represented in order of Kilobytes, which in this case the variance is irrelevant, being little significant for the overall performance. In addition, Bienia (2011) and Bhadauria, Weaver and McKee (2009) highlights that Parsec applications are more sensitive to memory latency and bandwidth than the amount of memory.

However, it is expected in memory usage, that workloads that demonstrate fluctuations, the more threads are added, the greater the level of parallelism and as a consequence, will also increase the memory utilization. As we can see in the Figure 3.5, some applications reduce the memory usage when scaling, so a further investigation it is necessary to comprehend this behavior.

3.3.2.3 Disk I/O

The Figure 3.6 shows the I/O disk usage of the twelve Parsec applications in the native environment. The trace application was used to capture the read and write disk operations. It gives a better insight into identifying the behavior of the workloads in their nature, especially in the case of I/O disk intense applications. This is necessary because, in general, the I/O-bound applications frequently presented a performance bottleneck, especially in the case of virtualization, which is known to not provide good performance portability over the native environment (HUBER; QUAST; HAUCK; KOUNEV, 2011).

It is expected that time spent in disk write operations increases as a measure that more threads are giving to the process. This is observed in all workloads and occurs because writing operations are slower than read operations, so in this case,



Source: Baum, Maliszewski, Griebler, 2017.

Figure 3.5: Performance Characterization of Enterprise Applications (Memory Usage).

when more worker threads are added, more concurrency occurs for write operations.

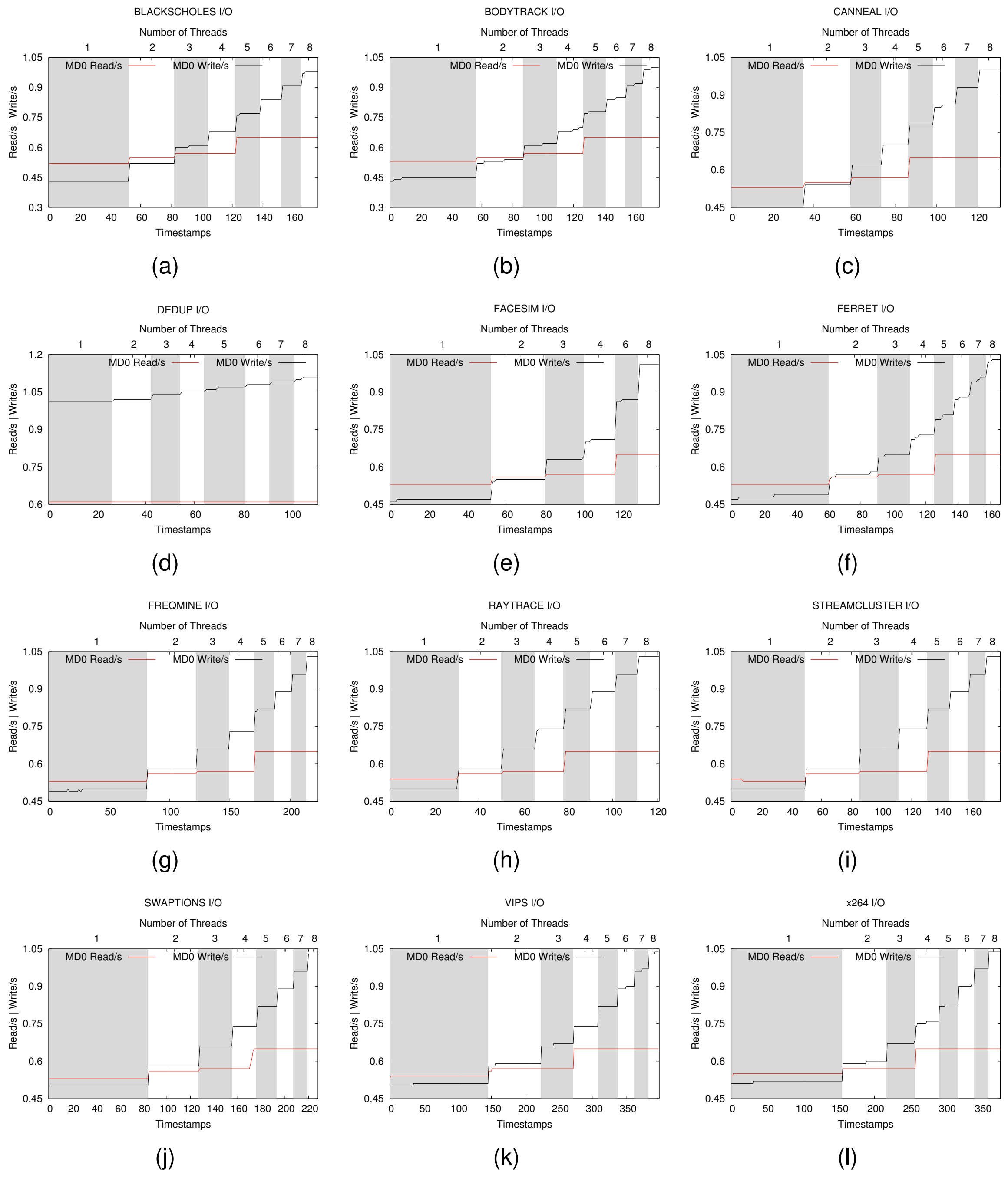
An interesting case occurs in Dedup (3.6(d)), which is a backup application that use pipeline parallelism and is bounded by the output I/O stage (BIENIA, 2011). Figure 3.6(d) shows that it demonstrates different behavior compared to others workloads, using high rates of disk writing. This essentially due to the algorithmic structure of this application, which loads the data from the disk, applies algorithmic methods to compress and mount the serial output stream that will be written back to the disk in the final stage.

It is also important to note that in virtualized and the multi-tenancy environments, I/O disk operations are expected to present significant overhead. It can occur due to the sharing of underline resources, as presented by Ghoshal, Canon and Ramakrishnan (2011b). This performance degradation, may also be significant for instances that do not implement disk-buffering techniques and in applications that works with small files, generating high latency and performance variation, leaving an uncontrolled competition for shared computing resources (BESERRA; ENDO; BARRETO, 2016), (SILVA; RYU; DA SILVA, 2012).

3.3.2.4 Cache Miss

The CPU cache-miss measured in the native environment (Figures 3.7(a), 3.7(b)), represents the number of data request not found in the cache memory of the CPU. Thus, when a cache miss occurs, a CPU fetch request is required for another cache level, such as L1, L2, L3, RAM or even the disk. As a consequence, the process latency is increased as the number of misses increases. This is also a side effect of the processor context-switch, caused by the thread scheduling (NIKOUNIA et al., 2015).

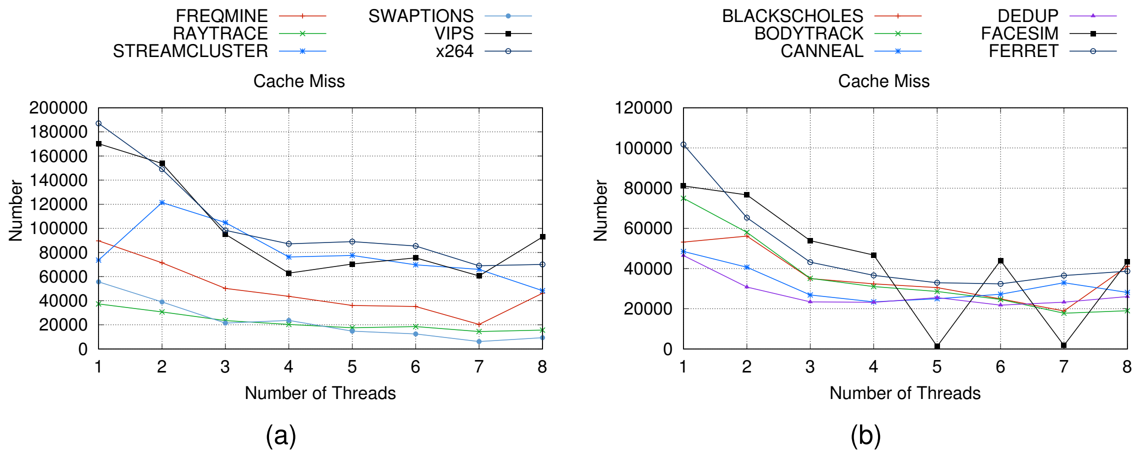
As we might expect applications that require an intense use of context-switching, such as Streamcluster, Ferret, Facesim, Canneal, Vips and Streamcluster, have high rates of cache miss, especially in the first threads, a similar result is presented by Bienia (2011) and Jiang et al. (2012), which indicates that it happens because the programs have a significantly larger working sets, so a small portion of it fits into the



Source: Baum, Maliszewski, Griebler, 2017.

Figure 3.6: Performance Characterization of Enterprise Applications (I/O).

cache. However, we can note that the cache miss also increases in the last thread for some applications, this possibly happens because as more working threads are added the cache is replaced quickly by new instructions, so this competition for more cache causes more misses.



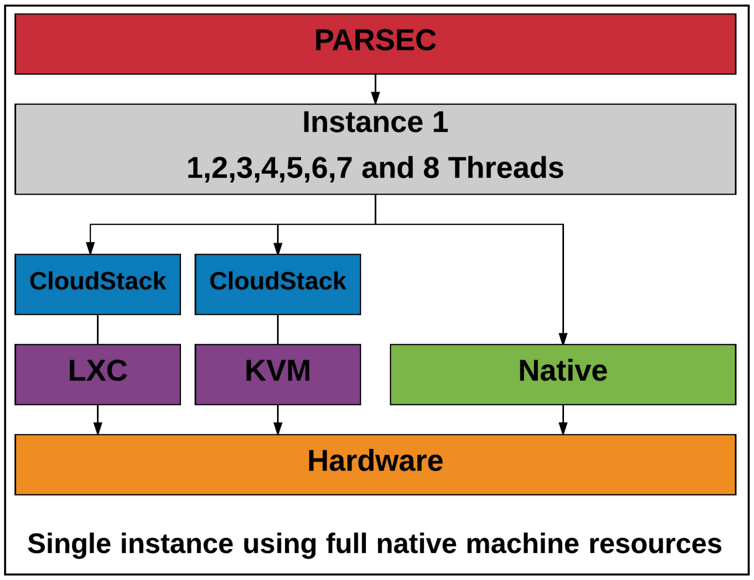
Source: Baum, Maliszewski, Griebler, 2017.

Figure 3.7: Performance Characterization of Enterprise Applications (Cache Miss).

3.3.3 HPC Scenario

The HPC environment is responsible for characterizing and comparing the performance of applications exposed in the native, KVM and LXC environments. The tests were performed using full node capacity in each individual environment without concurrency between the technologies. Therefore, the same service offer for hardware resources was used, this guarantee the test efficiency.

The Figure 3.8 describes the methodology employed to perform the HPC experiments. Thus, an instance was configured to scheduling from one to eight threads the Parsec workloads in three individual environments with the same hardware configuration. Hence, the average execution time is plotted on Figure 3.9. In addition, it is also important to highlight that the main individual Parsec application characteristics are described in Section 3.3.2.


PARSEC
Instance 1
1,2,3,4,5,6,7 and 8 Threads
CloudStack
CloudStack
LXC
KVM
Native
Hardware
Single instance using full native machine resources

Source: Baum, Maliszewski, Griebler, 2017.

Figure 3.8: Methodology followed in the evaluation of high performance scenario in enterprise applications.

The Blackscholes application (Figure 3.9(a)) represents the domain of financial analysis, being characterized to be data-parallel, coarse granulation, low sharing and exchange data usage. Bienia (2011) suggests that Blackscholes has the lowest cache miss rate of all Parsec programs with almost all of the shared data being accessed by two threads. This can be seen on Core load (Figure 3.4(a)) and cache miss (Figure 3.7(b)), where Blackscholes have relatively low cache miss despite the fact that it demonstrates being CPU intense (BHADAURIA; WEAVER; MCKEE, 2009). It possible occurs because the workload is small enough so that memory bandwidth and cache used in the experiments is not an issue. In addition, as mentioned by Chasapis et al. (2016) Blackscholes is a relatively simple application that can achieve high degrees of parallelism. Thus, virtualizations technologies demonstrate, in this type of implementation, to have little degradation effect in this application, revealing similar results on three environments.

Bodytrack (Figure 3.9(b)) is a data-parallel application that simulates the computer vision, which is responsible for tracking a human body with multiple cameras in a sequence image. It is characterized to have medium working set and high sharing data usage. Bodytrack also uses pipeline to perform I/O asynchronously, but Bienia (2011) suggests to treat it as a data-parallel because it does not take advantage of pipeline parallelism in the computationally intensive parts, which is responsible to compute floating points too. Thus, the image is loaded using a persistent serial thread pool, responsible for the intensive use of asynchronous disk I/O. Another characteristic appointed by Bienia (2011) and Bhadauria, Weaver and McKee (2009) is that the Bodytrack has working sets no larger than 16 MB and low L1 cache miss rates, this can explain the good scalability in our three environments showed in the Figure 3.9(b). However, as mentioned by the authors the serial parts of this programs will bottleneck the performance if we add more cores.

The Canneal application (Figure 3.9(c)) represents the field of engineering. It is characterized by having an unstructured model of parallelization with fine granularity, in addition, there is a high sharing and data exchange. Bienia (2011) describes it as having a very aggressive synchronization strategy based on data race and, in some cases, the processor context-switching can reduce cache performance, which explains the fact that it has the worst cache behavior of all Parsec benchmarks. It is also important to notice that it has a large working sets, which means that a small portion of it can fit on the processor cache, as a consequence, high levels of cache are used, increasing the latency. However, as we can see in Figure 3.9(c), performance is getting better as a measure of cores are added. Bienia (2011) argues that is caused by improved data sharing between threads. It is also realized that from one to four threads, that the KVM performance is worse than the container and native, this can be explained because Canneal is sensitive to memory latency, and the elevated number of context switches executed by the scheduler when migrated to vCPUs used in virtualization technology, is not fully ported compared to the native environment, resulting in a high cache miss rates, especially in first threads. A similar result is presented by Nikounia et al. (2015).

Dedup is an application (Figure 3.9(d)) that employs deduplication technique to eliminate redundant data, it represents the enterprise storage, which is commonly used in backup systems. It has pipeline parallelism composed of five stages, medium granularity with unbounded working set, high data sharing and exchange. This workload has some similarity to vips, both have serial stages (first and last) responsible for reading and writing the output format and also use small chunks to read and write to disk I/O (NAVARRO; ASENJO; TABIK; CASCAVAL, 2009), (BIENIA, 2011).

Dedup is defined by Bienia (2011) as a complex workload, with many threads executing different functions with different characteristics, in addition, it has a very large working set, having some issues with cache memory and, as a consequence, demands high levels of memory at disposal. In our tests represented in the Figure 3.9(d), dedup shows an interesting behavior especially on the fifth thread, which has a peak value compared to the others, this behavior is repeated in all three environments. This is easily explained if we analyze the CPU load (Figure 3.4(d)), we can realize that the serial part of the program causes the others threads to get stall while it writes the final stages to the disk.

Running Dedup in KVM based-cloud has demonstrated some overhead, probably caused by the extensive need to reorder the cache memory and the synchronizations points characteristic of this workload. The performance gets worse at the eighth thread, this happens because as we add more threads, the instructions that require the integer processing units responsible for hashing, checksums and compression gets full, and this queue of instructions causes the processor to stalled (NIKOUNIA; MOHAMMADI, 2015). In addition, some instructions can be forwarded from the third stage to the last, thus inducing more concurrency to the serial I/O stage, thereby increasing the bottleneck. It seems that KVM is not able to fully port the instructions presented in native OS, also in this case, because it uses QCOW2 disk images, which is known to have some limitations (REGOLA; DUCOM, 2010). However, what brings us surprise was the poor performance of the LXC based-cloud, the expected result would be some resemblance to the native environment, but actually we have a completely different one, being much worse than KVM. This result led us to re-run this test, with different deployment to investigate this issue, this is best described at the end of this subsection.

Facesim (Figure 3.9(e)) represents the field of realistic animation applying physical effects to simulating a muscular activation of a human face. It uses a data-parallel application with coarse granularity, large working set with low sharing and medium data exchange. Facesim is an intensive task parallel application with streaming behavior, and it is among applications with high rates of cache miss (BIENIA, 2011). This factor is also explained by the large working set used by the application, and it spends most of the time updating the human face. In addition, Facesim is knows for poor scheduling with more threads, this explanation is given by the fact that the main function of the program expects read requests from other threads. In our tests (Figure 3.9(e)), some overhead is noticed when KVM is used, a similar result is presented by the authors Nikounia and Mohammadi (2015). This is expected since the virtualization layer affects this type of application, which requires significant memory operations (BHADAURIA; WEAVER; MCKEE, 2009).

Ferret is a streaming application (Figure 3.9(f)) that uses pipeline parallelism to employ a similarity search. Basically the program segments an image and searches a database and sorts it in order of similarity. The process involves six stages, so the first (input) and the last stage (output) are serials, which is typical in this type of parallelism, and the four intermediate stages are responsible for query image segmentation (NAVARRO; ASENJO; TABIK; CASCAVAL, 2009), (BIENIA, 2011). Another characteristic of this application, is that the cache is intensively used for inter-thread communication, so that its capacity and latency is a fundamental property and could affect the behavior. In our experiments, we can perceive at the Figure 3.9(f) that the KVM suffers a little overhead on three and four threads. Probably the complexity of this application coupled with the queue buffers and high cache miss ratio causes some threads to stall, waiting the I/O serial stages in the pipeline. Thus, virtualization is incapable to efficiently coordinate the virtual processor (NIKOUNIA; MOHAMMADI, 2015), (BARROW-WILLIAMS; NICK; FENSCH; CHRISTIAN; MOORE, 2009), (NAVARRO; ASENJO; TABIK; CASCAVAL, 2009).

Freqmine behavior is represented in Figure 3.9(g) and is a data mining application parallelized with OpenMP. It applies a data-parallel model with medium granularity and also has high data sharing with medium exchange. It has a large working set, which easily exceed the amount of hundreds of megabytes (BIENIA, 2011). However, this application actively uses inter-thread communications and it is known to scale well across different CPU platforms, but for this, they must have a high FSB rates for communication among cores (BHADAURIA; WEAVER; MCKEE, 2009). In the performed experiments represented by the Figure 3.9(g), little impact due to virtualization can be seen on this application, therefore, this scenario without competition among users, represents an environment conducive to this type of application.

Raytrace performance is plotted in Figure 3.9(h). It is a rendering domain application, which uses a ray technique to create realistic images by applying lights and shadows effects. It has a data-parallel model with medium granularity, it shares high amounts of data with low exchange. Raytrace simulates the use of a camera by moving an object in front of it, and this view is shown on the screen. Bienia (2011) argues that this representation forms a large working set and also creates a significant reuse of it. This probably explains the relatively low cache miss rate (Figure 3.7(a)) in our experiments and also the high L1 cache requests presented by Nikounia and Mohammadi (2015). However, in the figure 3.9(h) we can see in our KVM-based cloud, Raytrace performance differently when compared to native and LXC. The overhead experienced may be due to the fact that Raytrace has some significant points of threads synchronization, so in this case the hypervisor adds more cycles to this task. A similar result is described by Nikounia and Mohammadi (2015), which also correlate this behavior by comparing perceived in Ferret, Facesim and Bodytrack, which also has a similar structure model.

Streamcluster is characterized in the Figure 3.9(i). It is a data-parallel mining application with medium granularity. It also has a medium working set with low data sharing and medium exchange. Bienia (2011) characterizes it as being similar to Blackscholes because they both make intense use of floating-point operations, and also with canneal, facesim and ferret, having high levels of cache miss rates. This can also be observated in the cache-miss Figure 3.7(a). However, Bhadauria, Weaver and McKee (2009) increases L2 cache sizes and finds a minimal reduction in miss rates, this is due to the fact that a larger working set does not fit into the cache area, in addition, it cannot take advantage of a larger L3 cache either, because some instructions do not extend to L3, bypassing it to the main memory (BIENIA; KUMAR; SINGH; LI, 2008). Thus, the authors Bhadauria, Weaver and McKee (2009) have identified that by increasing the FSB and giving the memory more channels, the performance of Streamcluster is improved. Another important fact is demonstrated by Nikounia and Mohammadi (2015), is that Streamcluster has some cache contentions and its many synchronization points can reduce the performance. It is noticed in our tests that in the first thread KVM has a worse performance initially, this can be explained by the high context-switching presented in the first thread used and the extended number of thread synchronization, thus spending more cycles. Moreover, KVM improves performance with more threads, being similar to that experienced by native and LXC. Possibly because adding more threads makes better memory usage and parallelization level is increased.

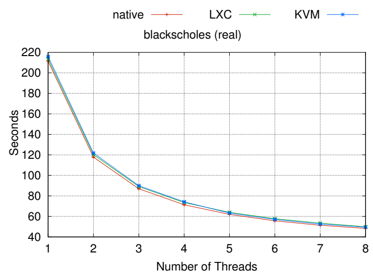
Swaptions is represented in Figure 3.9(j). It is an application of financial analysis, which is data-parallel with coarse granularity. It also has medium working set, both low sharing and data exchange. Swaptions basically breaks the input into small chunks and stores it in the portfolios array, so portions of the array are equally distributed among the working threads. It has a small working set that almost entirely fits into processor cache structure, which explain the best cache-miss rate of the Parsec applications (Figure 3.7(a)). In addition, the authors Bhadauria, Weaver and McKee (2009) and Chasapis et al. (2016) described that Swaptions and Blackcholes are relatively simple applications that scale well, moreover, they mentioned that benchmarks Blackscholes, Bodytrack, Fluidanimate, Freqmine, x264 and Swaptions achieves some performance gains using virtual cores, indicating that this can happen by the utilization of simultaneous multi-treading, which is capable to efficiently coordinate the threads and increase the level of parallelization. Thus, it is almost impossible to see the difference in three environments presented for such application.

Vips performance can be analyzed in Figure 3.9(k). It is a media processing application that applies a technique to perform an image transformation. It uses data-parallel model with coarse granularity and also has a medium working set with low data sharing and medium data exchange. Vips is composed by 18 pipeline stages and initially it loads parts of an input image on demand and uses memory-mapped I/O to perform the operations and re-write to disk (BIENIA, 2011). The application is described by Bhadauria, Weaver and McKee (2009) as having good scalability and also because it is sensitive to the number of memory channels. In our experiments, KVM performance was expected due to the fact that the virtualization layer causes some significant overhead especially in the case of applications that demands disk I/O operations and synchronization points, is a similar result presented in the experiments of Nikounia and Mohammadi (2015). However, what was not expected was the poor performance of LXC containers, which was worse than KVM in almost all threads. This results requires further investigation because it is known that LXC performs near-native and the virtualization layer does not interfere as it does in KVM.

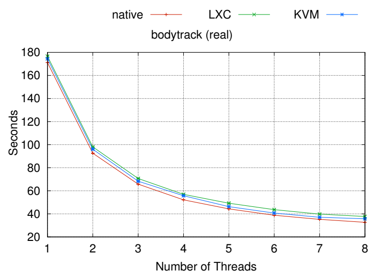
X264 is a media processor based on H.264 video encoder standard that uses a short scene from an uncompressed movie to encode. This workload (Figure 3.9(l)) uses pipeline parallelism with coarse granularity and has a medium working set using high amounts for sharing and exchange data. In native input, scenes from an uncompressed movie in HDTV resolution are encoded. Initially x264 build a complex pipeline model to compute the dependency between the frames, where each frame corresponds to a pipeline stage, then the frames are divided across thread using fluctuating points units (BIENIA, 2011), (BHADAURIA; WEAVER; MCKEE, 2009). In our tests, x264 demonstrates good scalability up to eight threads, presenting a similar result in three environments, revealing that the communication patterns employed by this type of parallelization method coupled with the intense use of floating point units is not considerably affected by virtualization.

The LXC containers are well known to be a light-weight virtualization with little virtualization overhead because they share the same kernel that the OS, employing cgroups and namespace for hardware and software isolation. Its features are well documented and several papers describes that it performs better than full virtualization, but at the cost of less resources isolation. However, two specific applications demonstrated some unpredictable results using Linux Containers, they are, Dedup and Vips. The performance of such applications was dramatically lower than KVM, so the results made us suspect that something interfered in the experiments, so we ran it again and the same results were experienced.

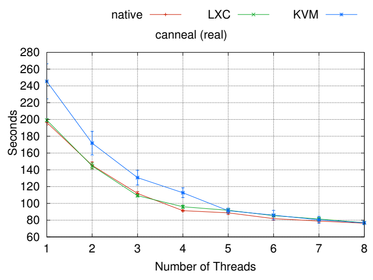
A deeper investigation was conducted to answer the unsolved question and we discovered that two things are influencing our results. The first is about the workloads characteristics. Two applications use pipeline parallelism, so it is typical to find in this model a large number of threads bounded to serial I/O stage, which is responsible for reading and writing data to disk. Therefore, both applications compute their data in the intermediate stages and require to write the final data on the disk, followed by a reordering of the stages. This causes a well-known overhead in the KVM, some papers estimate to be about 6 to 10%. But in the case of LXC, the explanation goes further and relies on the way that cloud manages the containers. Cloudstack uses a primary storage for running VMs and attached disks, and a secondary storage for system images, ISOs, and snapshots, both storages are mounted on the front-end node and use the NFS protocol. The front-end node was not used by the experiments because it was responsible for managing the cloud infrastructure and its utilization could distort the results. Therefore, we essentially used the compute node two and



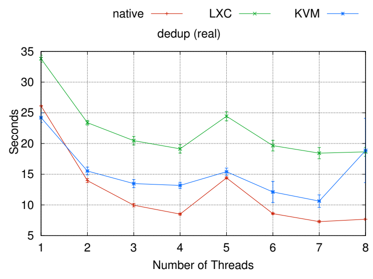
(a)



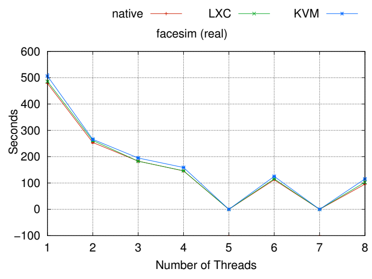
(b)



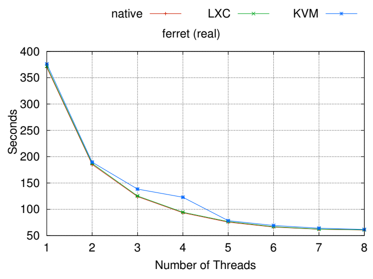
(c)



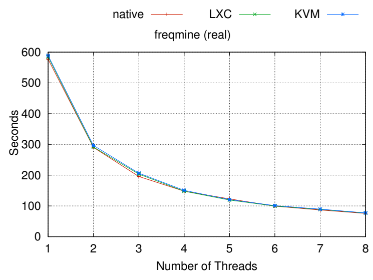
(d)



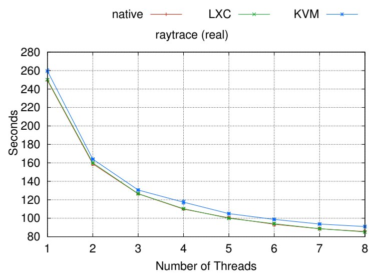
(e)



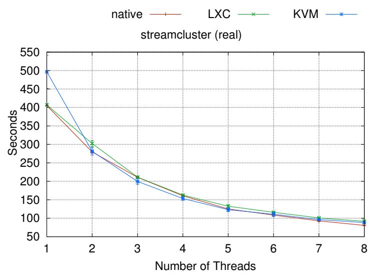
(f)



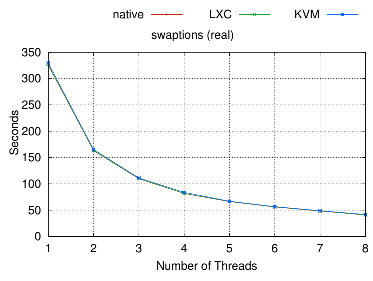
(g)



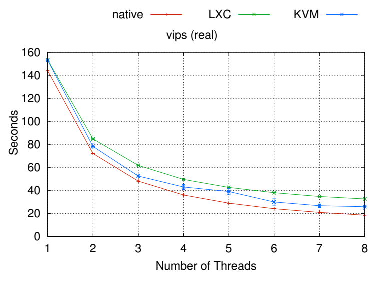
(h)



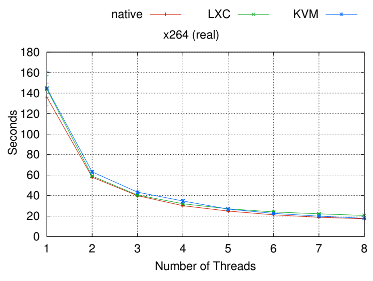
(i)



(j)



(k)



(l)

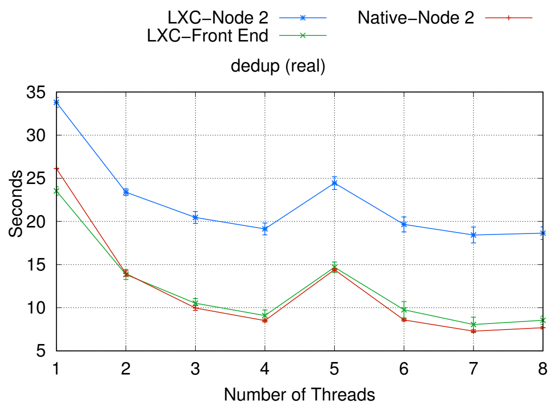
Source: Baum, Maliszewski, Griebler, 2017.

Figure 3.9: High Performance Scenario.

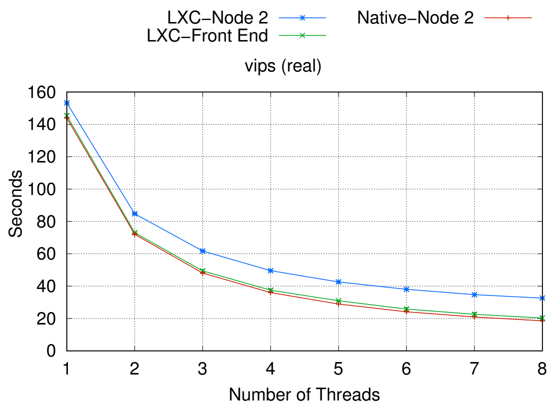
four (node three was deactivated) to performs the experiments. However, we suspect that this could might contain the response and we decided to perform the same test by running the instance on the front-end node where the mounted NFS storage is located.

Thus, Figures 3.10(a) and 3.10(b) contains the results of the tests performed on the front-end node and compute node 2.

Therefore, the second thing that we discover is, that the workloads performed differently in LXC executed in front-end in comparison to LXC executed in compute node 2. The reason for the poor performance presented by the compute node (Figure 3.10(a), 3.10(b)) is because Cloudstack accesses shared disk of primary storage over the network via NFS, so when the underlying storage uses NFS for disk I/O operations, the data is broken into smaller, costly and unstable NFS I/O operations that require windows sizes that are not sufficiently large, increasing latency. So, this intensely repeated behavior by applications that demand high disk usage has its performance massively degraded (AL-MUKHTAR; MARDAN, 2014), (FELTER; FERREIRA; RAJAMONY; RUBIO, 2015). In the case of front-end, the disk is accessed locally, without relying NFS, this explains the good performance.



(a)



(b)

Source: Baum, Maliszewski, Griebler, 2017.

Figure 3.10: Deeper investigation (Dedup and Vips).

So even if we setting in the Cloudstack dashboard to instantiate the VM on compute node 2, the container uses its hardware (memory and processor) resources, but manipulates the I/O data that is mounted on the front-end node using NFS, causing the bottleneck (UEDA; NAKATANI, 2010). This is more expressive in these two workloads, because according to Bienia (2011), both use I/O to load parts of an input image and then re-write to the disk, and this operation using the network increases the latency. Therefore, the LXC based-cloud using this deployment is not favorable to this type of workload. However, a further investigation is necessary to predict whether a Cloudstack issue or a cloud deployment configuration.

3.3.4 Multi-tenancy Scenario

The multi-tenancy environment represents concurrency among users who use the same service offering. This is the closest scenario to represent a IaaS cloud production environment. Because it adds more users, it is instigated to compete for hardware resources and different results between virtualization technologies are expected as more users are added.

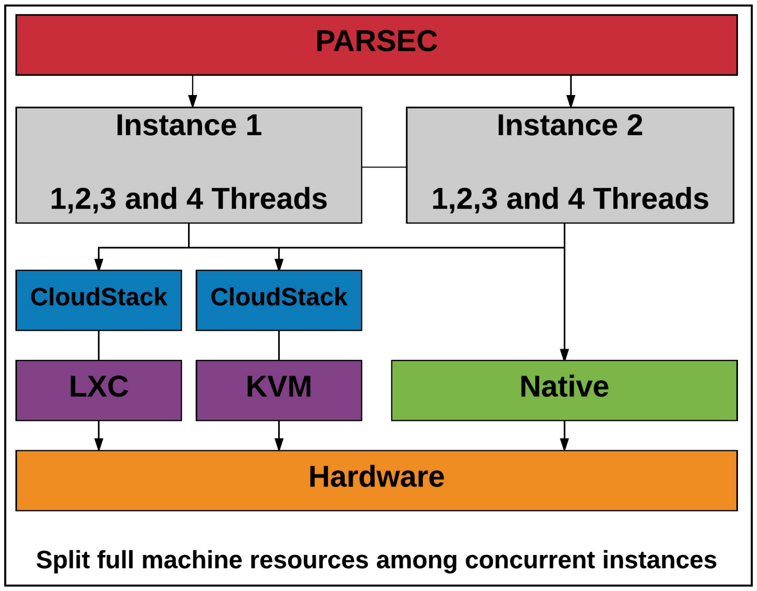
Figure 3.11 represents the methodology used to perform the experiments. Thus, two instantiated users share the same computing node with split hardware resources, running Parsec workloads in three different environments.

3.3.4.1 Configuration 1

The first configuration made, used all the parsec applications in two concurrent instances (Figure 3.12). With this test we expect to find if there is statistically difference between the environments in deployed scenario.

3.3.4.1.1 Same Applications

Some applications show insignificant variation using two users when compared to the native environment, are blackscholes (Figures 3.13(a), 3.14(a), bodytrack (Figures 3.13(b), 3.14(b)), facesim (Figures 3.13(f), 3.14(e)), ferret (Figures 3.13(e), 3.14(f)), raytrace (Figures 3.13(h), 3.14(h)) and swaptions (Figures 3.13(j), 3.14(j)). Canneal (Figures 3.13(c), 3.14(c)) is an interesting case despite the fact that it has a

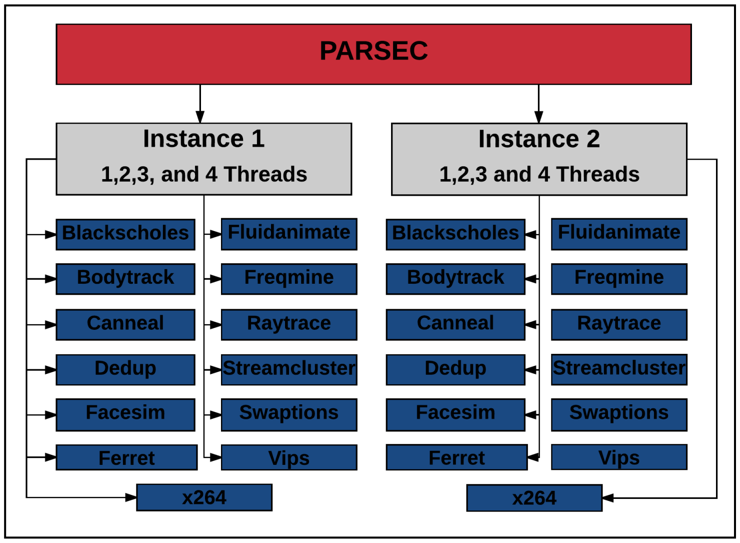


Source: Baum, Maliszewski, Griebler, 2017.

Figure 3.11: Conf1: Methodology followed in the evaluation of enterprise applications in multi-tenancy scenario.

high cache miss ratio, both KVM users has performed significantly better than KVM HPC in all threads, while the native and LXC outperforms the HPC from the second thread. This can be explained by the nature of this application, which shares large amounts of cache for read-only data and it is unbounded, that is, it does not have limited memory usage. (BIENIA, 2011), (NIKOUNIA; MOHAMMADI, 2015).

Dedup (Figures 3.13(d), 3.14(d)) and Vips (Figures 3.13(k), 3.14(k)) instead, demonstrated a proportional performance in all users and environments, with LXC-users being even worse than HPC, caused by manipulation of container on another host. Freqmine demonstrated in LXC and native has performed poorly in the first thread, demonstrating that this unbounded workload gets better as it is scalonated and present negligible overhead in the next threads due to the high degrees of data cache sharing. Streamcluster is a stream data mining application, sensitive to memory latency. Tests have shown that KVM, LXC, and native users outperform the HPC environment in all environments, despite the fact that computing resources have been split. This is possible explained to be because Streamcluster can achieve high degrees of parallelism, and in the Pthreads version spends up to 90% of total execution time in the same function, as a consequence running the same applications in the neighbor makes it favorable to cached data sharing. The last of the thirteen applications is x264, the native user2 is initially better than the others, but stabilizes after that, a possible reason is that it could benefit from the data shared by the user1, which started the workload first.



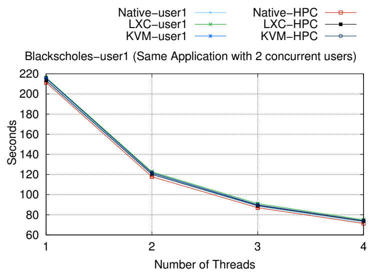
Source: Baum, Maliszewski, Griebler, 2017.

Figure 3.12: Description of the Parsec multi-tenancy scenario with same applications on 2 concurrent users.

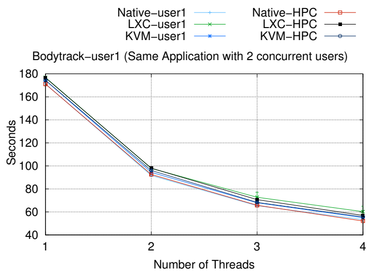
3.3.4.1.2 Different Applications

To perform this deployment, the Parsec applications was splited in two instances, that is, each instance with six different applications (Figure 3.15). A care was taken to avoid similar domain applications (e.g. data mining with data mining) to be executed one after another in the same instance, this to simulate a differentiated environment and avoid the cache sharing.

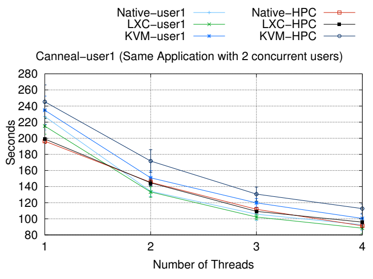
Blackscholes (Figure 3.16(a)), Facesim (Figure 3.17(c)), Freqmine (Figure 3.16(d)), Raytrace (Figure 3.16(e)), Swaptions (Figure 3.17(e)), x264 (Figure 3.16(f)), had negli-



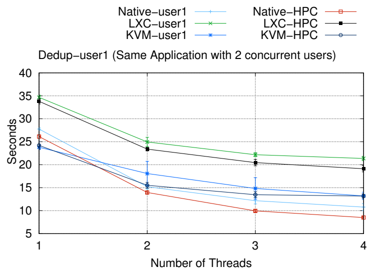
(a)



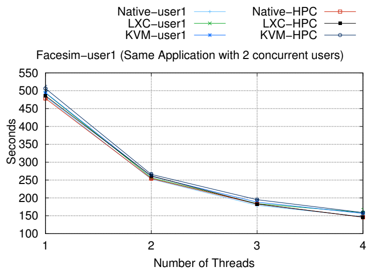
(b)



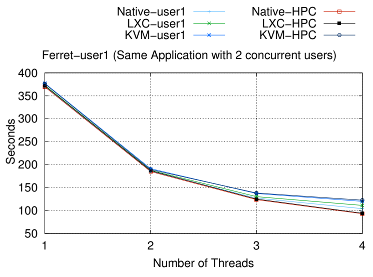
(c)



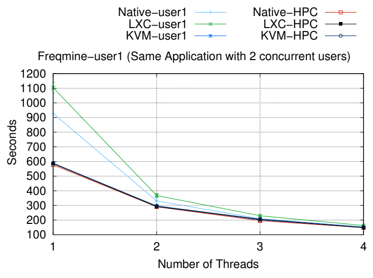
(d)



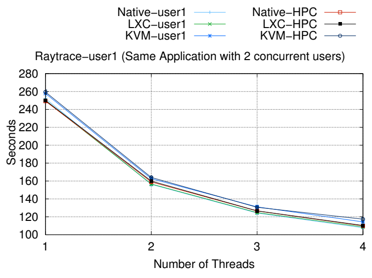
(e)



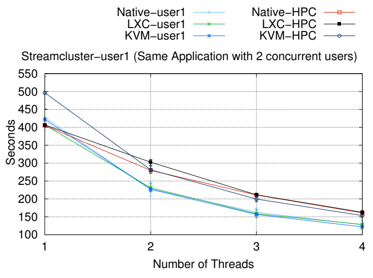
(f)



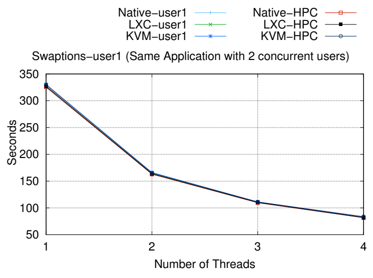
(g)



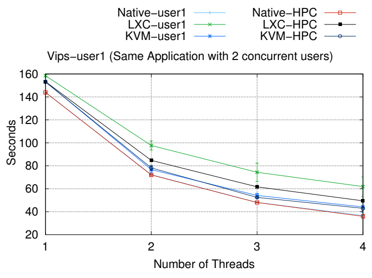
(h)



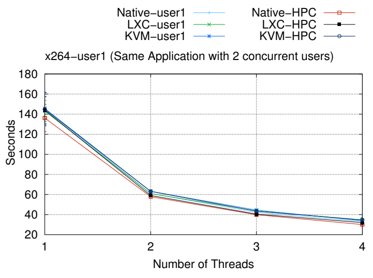
(i)



(j)



(k)

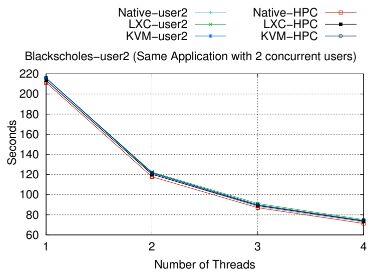


(l)

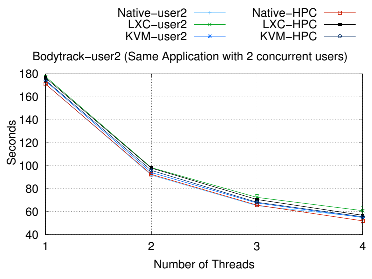
Source: Baum, Maliszewski, Griebler, 2017.

Figure 3.13: MultiTenancy-User1 Scenario (Same Applications with 2 concurrent users).

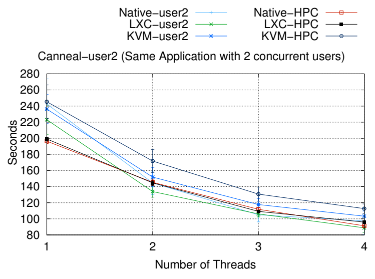
gible difference among users, with little overhead due to virtualization. Bodytrack (Figure 3.16(b)), shows little difference in the first thread with LXC, indicating some interference from the neighbor user2. Canneal (Figure 3.17(a)) and Streamcluster (Figure



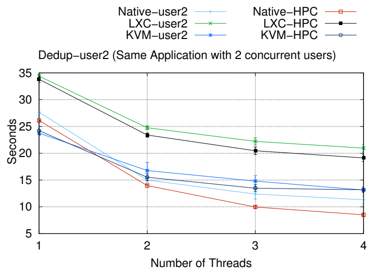
(a)



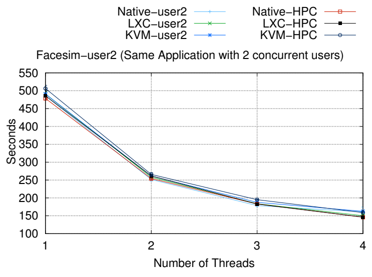
(b)



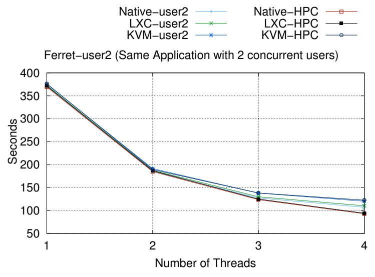
(c)



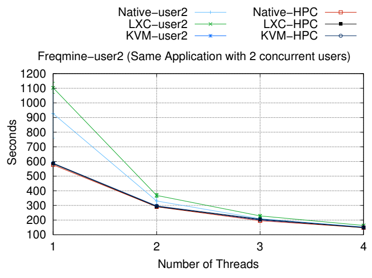
(d)



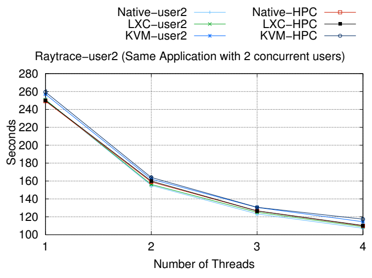
(e)



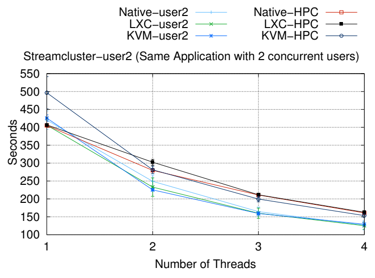
(f)



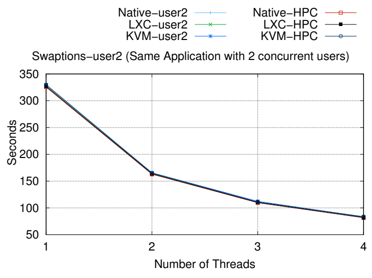
(g)



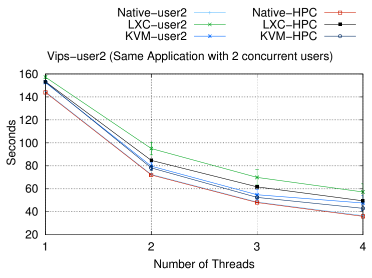
(h)



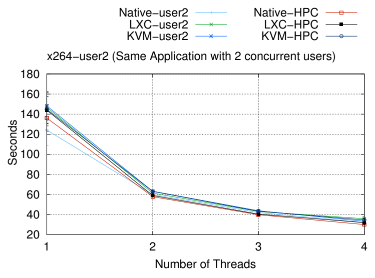
(i)



(j)



(k)

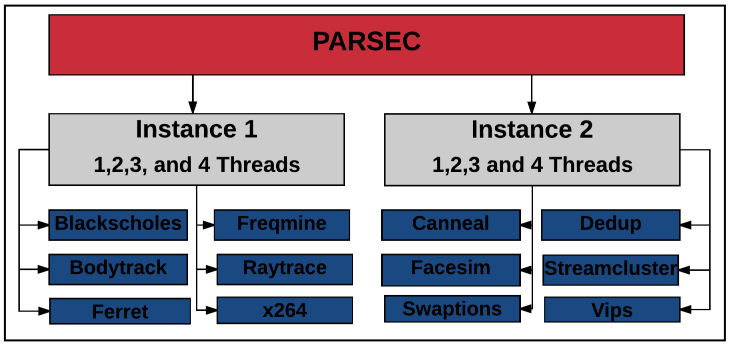


(l)

Source: Baum, Maliszewski, Griebler, 2017.

Figure 3.14: MultiTenancy-User2 Scenario (Same Applications with 2 concurrent users).

3.17(d)) that are similar in sensitive to main memory latency (BHADAURIA; WEAVER; MCKEE, 2009), (NIKOUNIA; MOHAMMADI, 2015), have some performance boost from the second thread on all users. Two possible factors explain this behavior, mainly



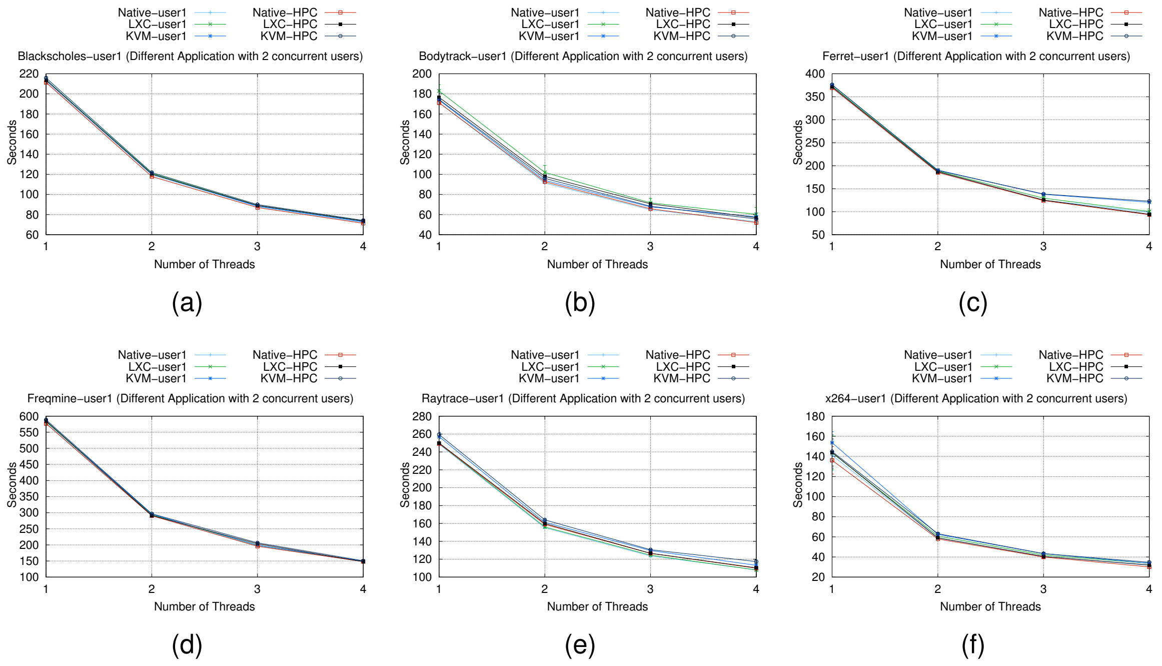
Source: Baum, Maliszewski, Griebler, 2017.

Figure 3.15: Description of the Parsec multi-tenancy scenario with different applications on 2 concurrent users.

because the working set is larger and requires more access to main memory, so in this case, the transactional lookaside buffer (TLB) that is responsible for mapping regular memory access works very well reducing memory misses, (FOUNDATION, 2016), (FELTER; FERREIRA; RAJAMONY; RUBIO, 2015) and also the data prefetch, which loads the data or instructions from the main memory to a lower-level cache that is faster before they are needed, predicting the block addresses that will be referenced (CHEVERESAN; RAMSAY; FEUCHT; SHARAPOV, 2007), (FERDMAN; ADILEH; KOCBERBER; VOLOS; ALISAFAEE; JEVDJIC; KAYNAK; POPESCU; AILAMAKI; FALSAFI, 2012). Dedup (Figure 3.17(b)) and Vips (Figure 3.17(f)), has an expected performance degradation with LXC-user2, due to the manipulation of the container on another host, and the similar performance in KVM and native has been realized. Ferret (Figure 3.16(c)), demonstrates a growing virtualization overhead in the fourth thread, which is expected due to the serialized stages, especially the I/O input presented in this workload.

3.3.4.2 Configuration 2

To perform this configuration, three instances were used to simulate the multi-tenancy environment (Figure 3.18). Furthermore, to have a total distribution from the eight threads from the processors the service offer was configured for the first and



Source: Baum, Maliszewski, Griebler, 2017.

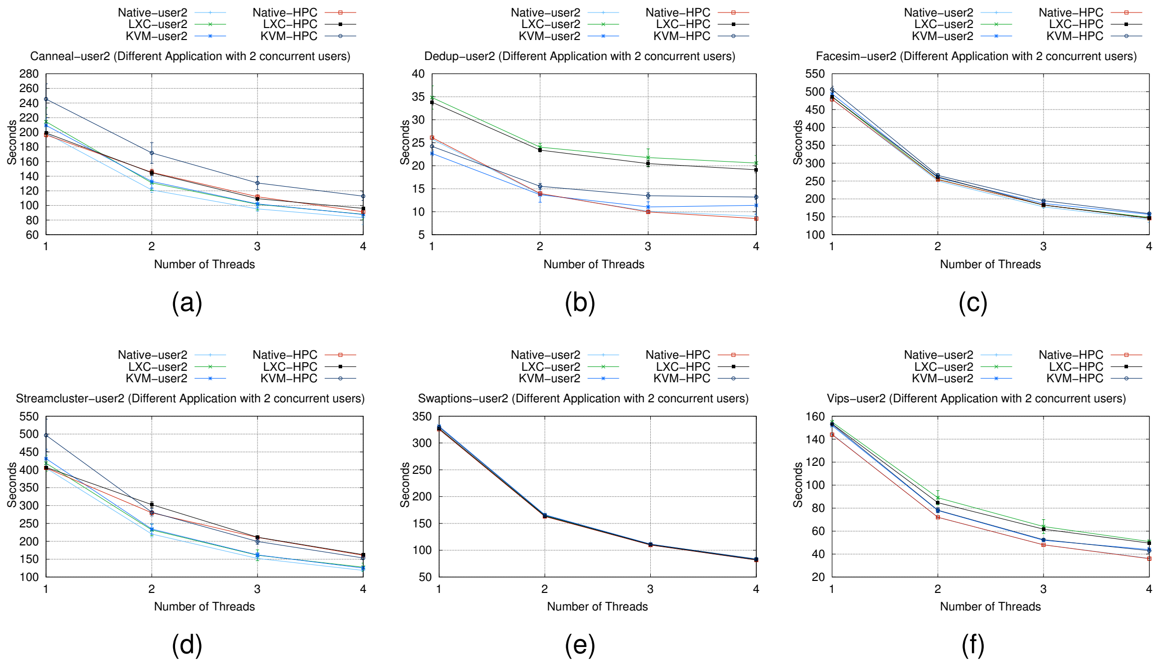
Figure 3.16: Multi-tenancy-User1 Scenario (Different Applications with 2 concurrent users).

second users to use three threads and the third uses two threads.

3.3.4.2.1 Same Applications

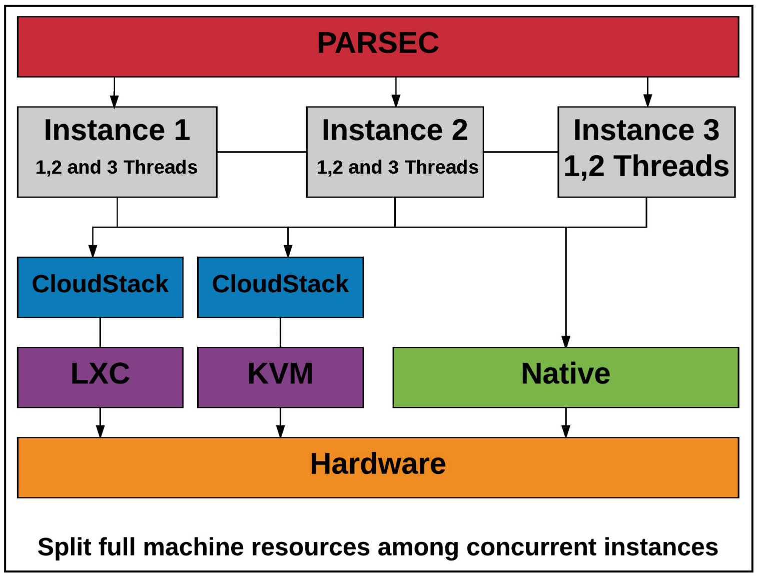
The same applications equally distribute all the workloads on the instances, as is shown in the Figure 3.19.

In the case of Blackscholes (Figures 3.20(a), 3.21(a), 3.22(a)), Facesim (Figures 3.20(e), 3.21(e), 3.22(e)), Raytrace (Figures 3.20(h), 3.21(h), 3.22(h)) and Swaptions (Figures 3.20(j), 3.21(j), 3.22(j)) workloads, all users showed little difference between virtualization technologies, with similar variations that occurred in the past test. The Bodytrack LXC workload on user 1 (Figure 3.20(b)) and 2 (Figure 3.21(b))starts to behave worse than KVM and user 3 (Figure 3.22(b)) overcomes KVM on the second thread, we suspect that simultaneity in I/O competition induces these results. Canneal (Figures 3.20(c), 3.21(c), 3.22(c)) and Streamcluster (Figures 3.20(i), 3.21(i), 3.22(i))



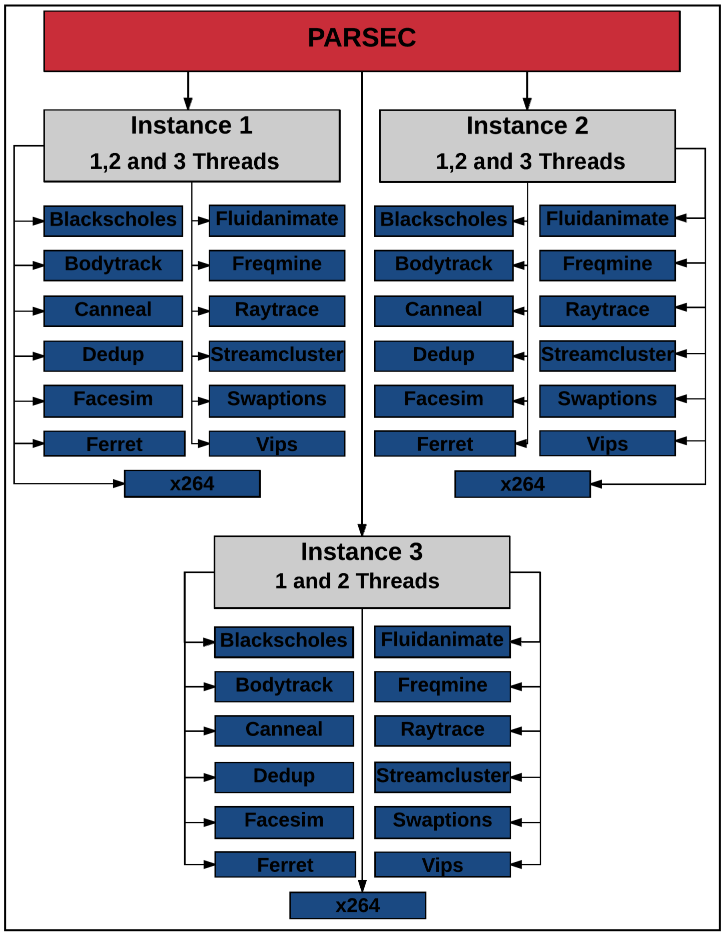
Source: Baum, Maliszewski, Griebler, 2017.

Figure 3.17: Multi-tenancy-User2 Scenario (Different Applications with 2 concurrent users).



Source: Baum, Maliszewski, Griebler, 2017.

Figure 3.18: Conf2: Methodology followed in the evaluation of enterprise applications in multi-tenancy scenario.



Source: Baum, Maliszewski, Griebler, 2017.

Figure 3.19: Description of the Parsec multi-tenancy scenario with same applications on 3 concurrent users.

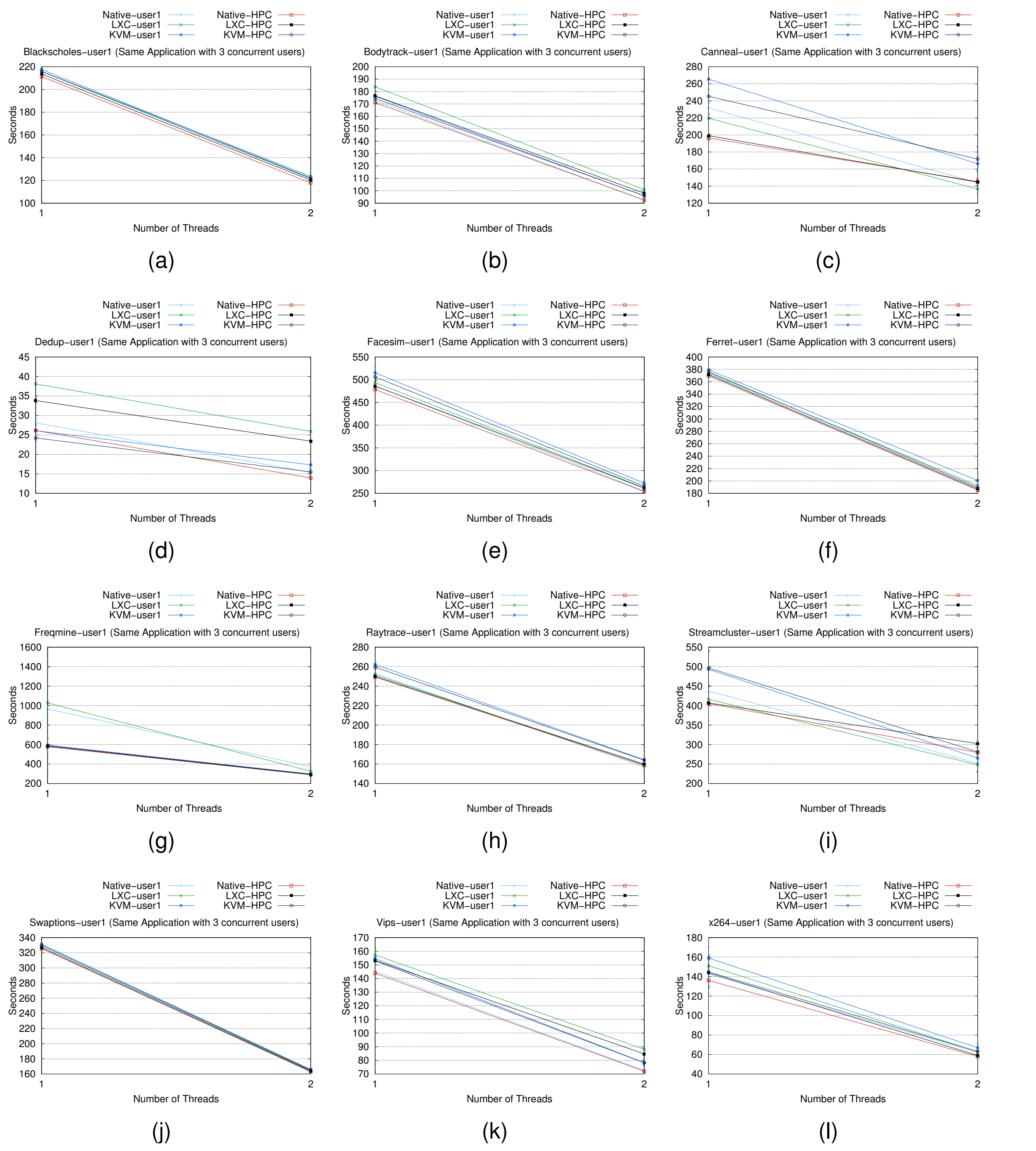
have demonstrated some erratic behavior in all LXC users, performing initially better than native, this is expected, as these workloads were described by Nikounia and Mohammadi (2015) by suffering more cache contention, in addition, container resource coordination proves to be more efficient than native users who were competing for resources aggressively, which elevates the cache miss rate.

Dedup (Figures 3.20(d), 3.21(d), 3.22(d)) and Vips (Figures 3.20(k), 3.21(k), 3.22(k)) as expected, had poor LXC performance, with KVM being similar to native in all users. Ferret and x264 in third user with KVM presented high levels of overhead in the second thread, it could have been caused by the fact that the instance was mounted with different thread configuration. Freqmine (Figures 3.20(g), 3.21(g), 3.22(g))is interesting because KVM gain some boost in the first thread and LXC performs better than native in user 2 and 3. In fact, in this case an opposite result was expected but a closer analyze indicates that using native and LXC the OS apparently does not coordinate efficiently the resource isolation between users, inducing high rates of context switching and cache miss.

3.3.4.2.2 Different Applications

The proposed environment divides the applications in three users, as shown in Figure 3.23.

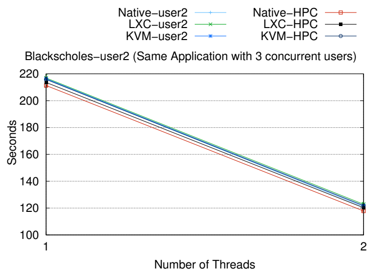
Blackcholes (Figure 3.24(a)) demonstrates little difference between virtualization technologies in this environment as well, which proves to be favorable to this application especially due to the fact that almost all shared data is accessed by two threads, where the main thread is not communication-bound in relation to the worker's threads (BIENIA, 2011). And similarly occur in Facesim (Figure 3.25(a)), Freqmine (Figure 3.25(b)), Swaptions (Figure 3.26(d)) workloads, where different levels of parallelization can be achieved, but using two threads the result is negligible difference among virtualizations. Bodytrack (Figure 3.24(b)), LXC repeats behavior presented in the same



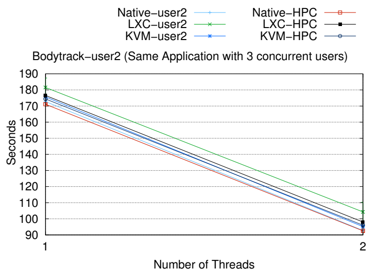
Source: Baum, Maliszewski, Griebler, 2017.

Figure 3.20: Multi-tenancy-User1 Scenario (Same Applications with 3 concurrent users).

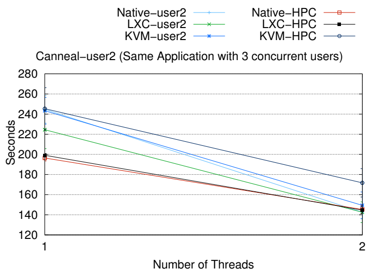
application environment, where performance is below than expected, this is a result of the use of asynchronous disk I/O where a pool of threads loads the input images to be computed using floating point operations (BIENIA, 2011).



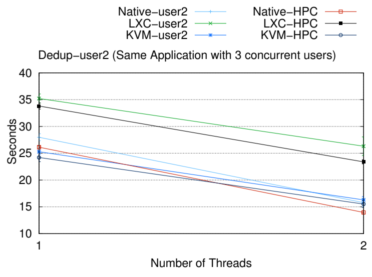
(a)



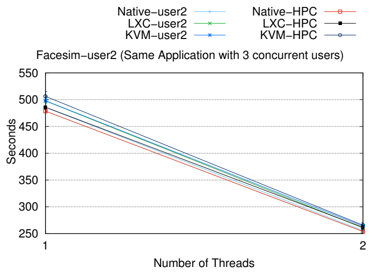
(b)



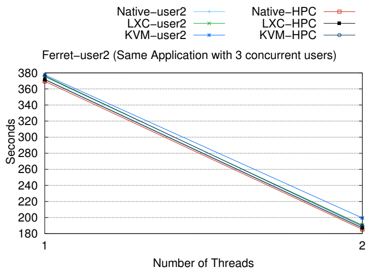
(c)



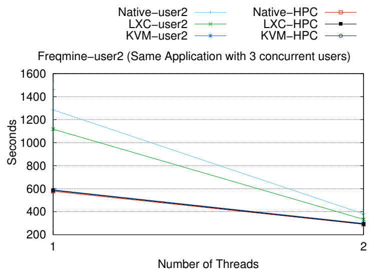
(d)



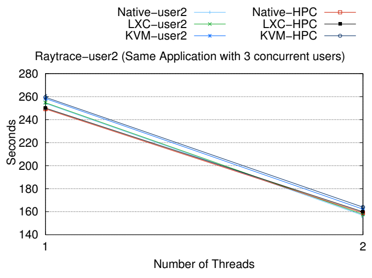
(e)



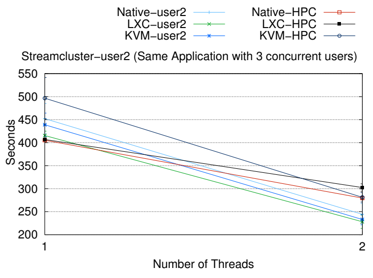
(f)



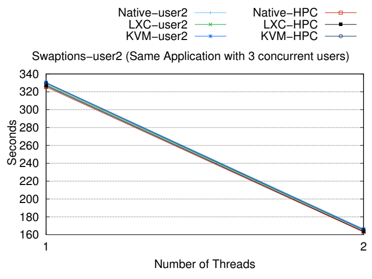
(g)



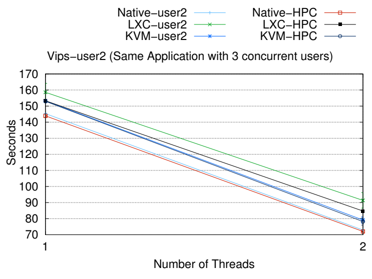
(h)



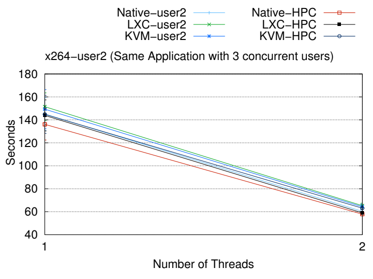
(i)



(j)



(k)

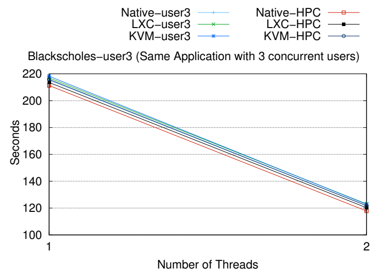


(l)

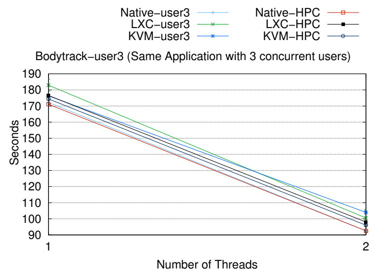
Source: Baum, Maliszewski, Griebler, 2017.

Figure 3.21: Multi-tenancy-User2 Scenario (Same Applications with 3 concurrent users).

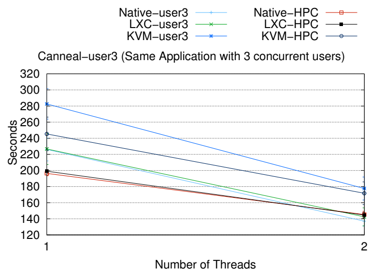
Canneal (Figure 3.24(c)) now exhibits similar performance between native and KVM, and LXC has a worse performance, indicating that the race for more resources in the multi-tenancy induces this result, a similar behavior is presented by Nikounia and



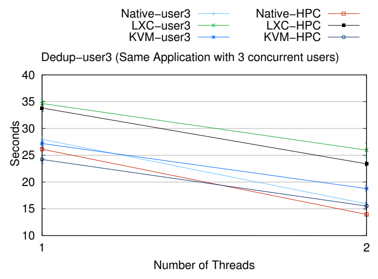
(a)



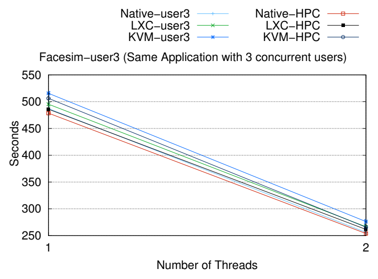
(b)



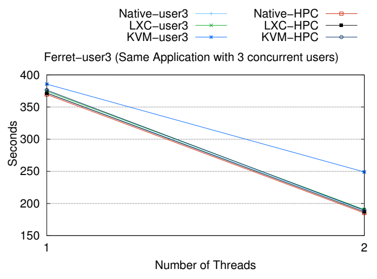
(c)



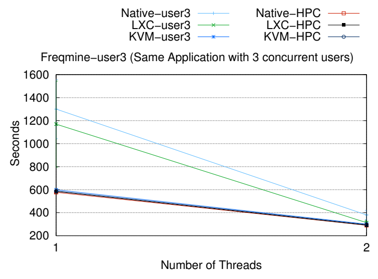
(d)



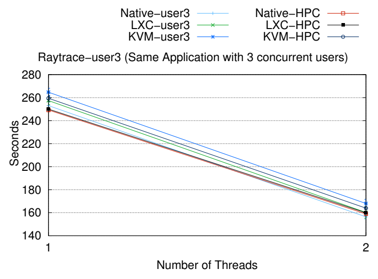
(e)



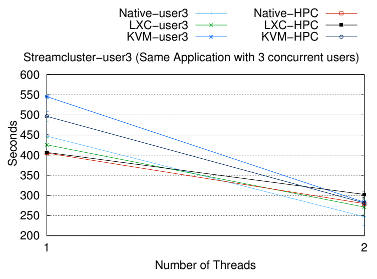
(f)



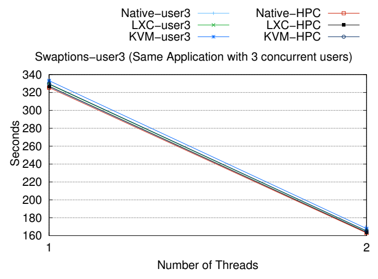
(g)



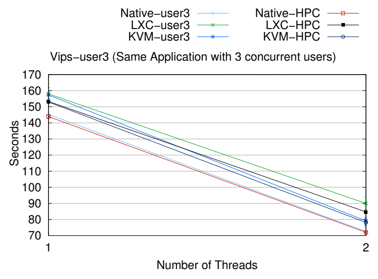
(h)



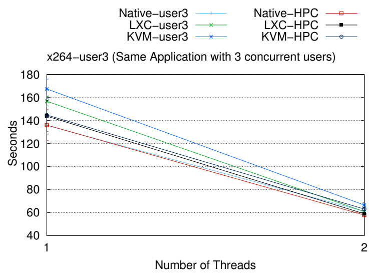
(i)



(j)



(k)

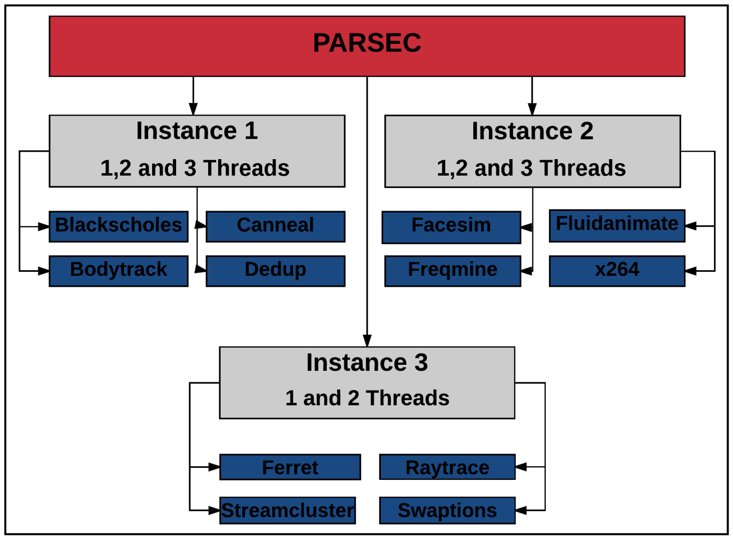


(l)

Source: Baum, Maliszewski, Griebler, 2017.

Figure 3.22: Multi-tenancy-User3 Scenario (Same Applications with 3 concurrent users).

Mohammadi (2015) and Bhadauria, Weaver and McKee (2009) where this workload exhibits high rates of context switching and is also sensitive to memory latency. In the case of Dedup (Figure 3.24(d)) and Vips (Figures 3.26(e), LXC again proves to be

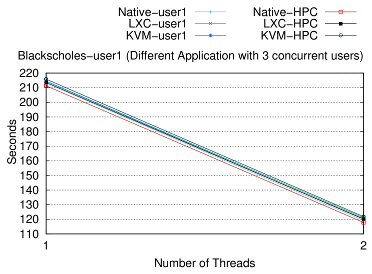


Source: Baum, Maliszewski, Griebler, 2017.

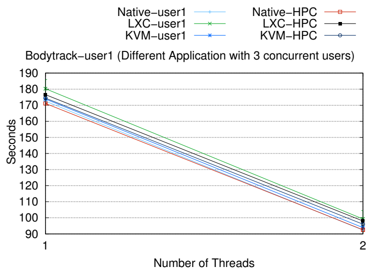
Figure 3.23: Description of the Parsec multi-tenancy scenario with different applications on 3 concurrent users.

inefficient for this deployment. KVM is better than native, which is not surprising due the fact that the race for resources in the native environment between users can increase the cache miss, especially in this case, and this workload is described by Bienia (2011) for being dependent on effectiveness of cache management. Ferret (Figure 3.26(a)), reveals increasingly KVM overhead in the second thread, a similar result is presented by Nikounia and Mohammadi (2015), possibly caused due to the restriction of threads used by the virtualized processor, negatively affecting the resources required to satisfy the strong communication required in this workload.

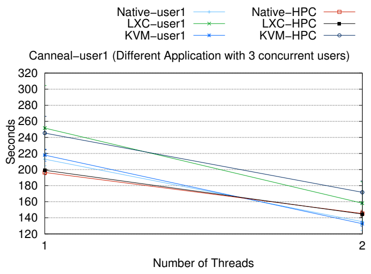
The workload Raytrace (Figure 3.26(b)) renders an image and according to Nikounia and Mohammadi (2015), although the working set is large, many parts of its computation can be reused achieving less L1 cache misses, which leads to LXC performs a slightly better than KVM in our experiments. Streamcluster (Figure 3.26(c)) it is mentioned by Nikounia and Mohammadi (2015) and Bhadauria, Weaver and McKee (2009) by having largest memory demands and also has many synchronization points, inducing more instructions to the schedulers presented in the VMs, consequently KVM and LXC have similar overhead in comparison to native, especially in this case, where only two threads were giving to the processor. In the case of X264 (Figure 3.25(c)), apparently the serial sections of the code and also the context switching between the initial thread adds some overhead initially to the KVM, where it starts slower but in the second thread the result is similar to LXC.



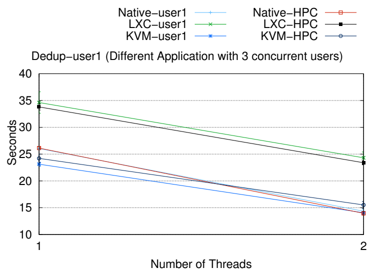
(a)



(b)



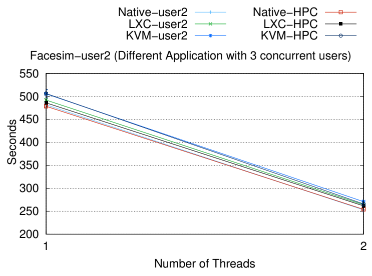
(c)



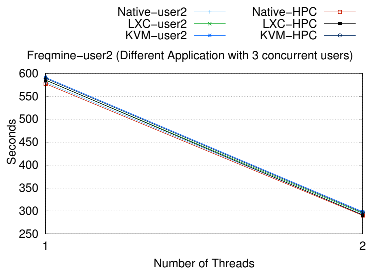
(d)

Source: Baum, Maliszewski, Griebler, 2017.

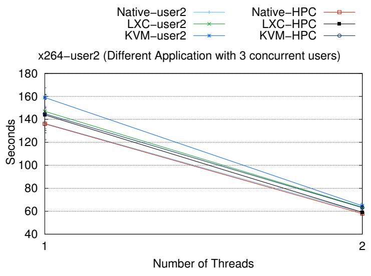
Figure 3.24: Multi-tenancy-User1 Scenario (Different Applications with 3 concurrent users).



(a)



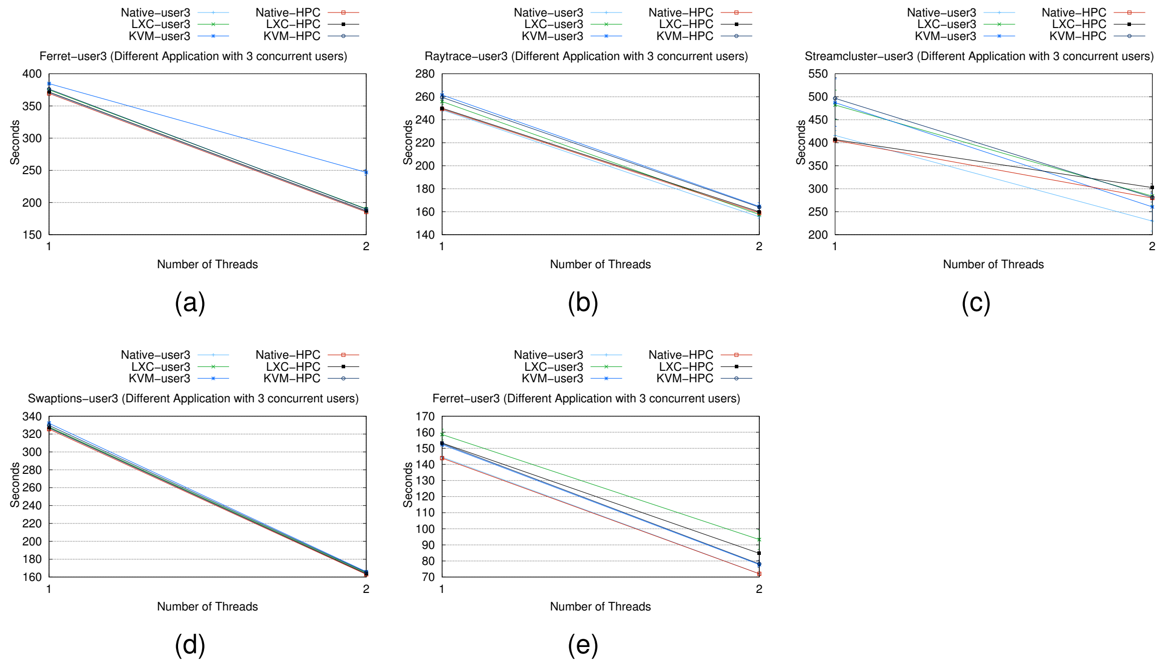
(b)



(c)

Source: Baum, Maliszewski, Griebler, 2017.

Figure 3.25: Multi-tenancy-User2 Scenario (Different Applications with 3 concurrent users).



Source: Baum, Maliszewski, Griebler, 2017.

Figure 3.26: Multi-tenancy-User3 Scenario (Different Applications with 3 concurrent users).

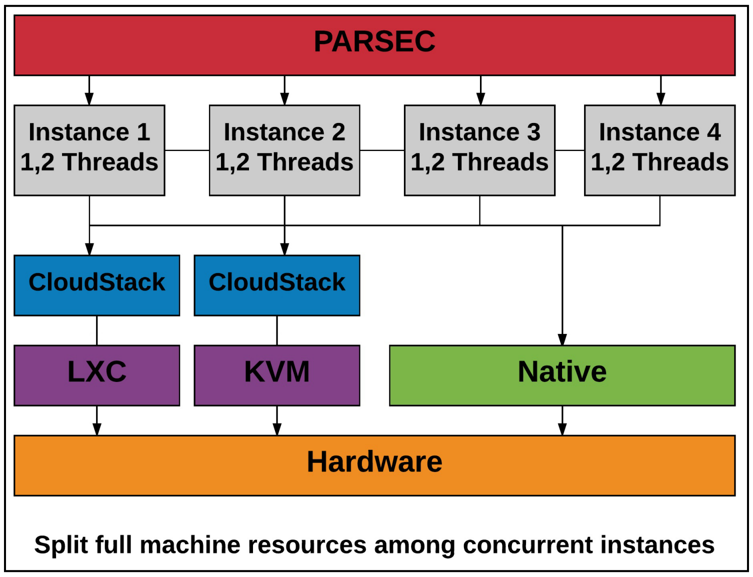
3.3.4.3 Configuration 3

The last configuration made, simulates the multi-tenancy environment using four users (Figure 3.27). In the case of cloud users, service offerings have been configured to split computing resources according to the available processor threads and main memory.

3.3.4.3.1 Same Applications

This methodology focuses on running all applications on the four users simultaneously (Figure 3.28) to investigate the performance of parsec workloads. Therefore, four instances were deployed in the case of cloud, and four root users were created in the case of native environment.

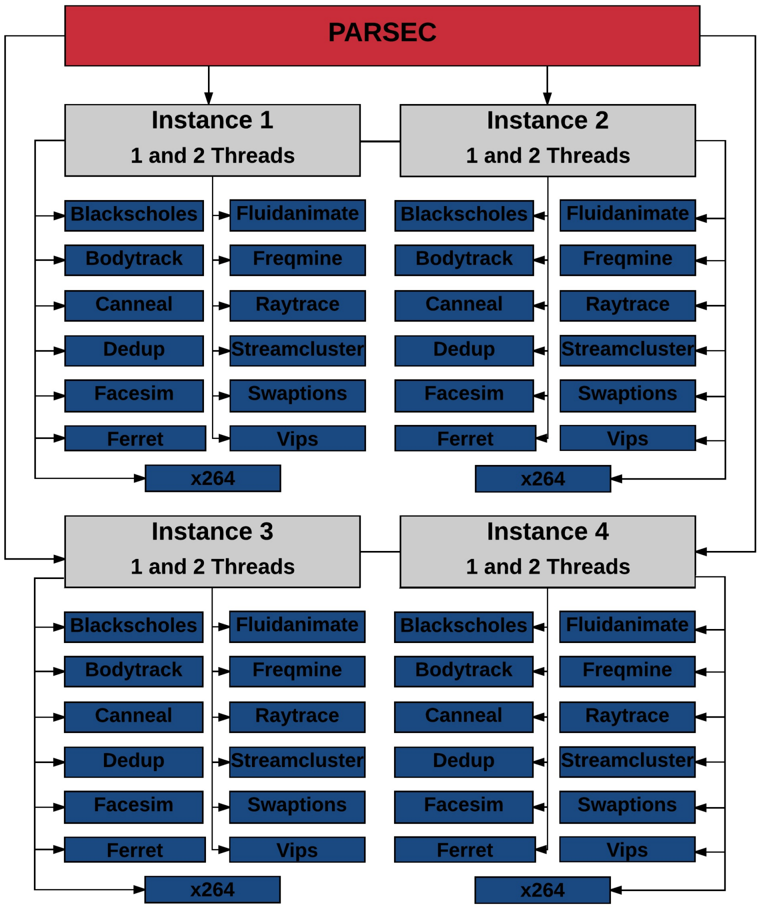
The Blackscholes (Figures 3.29(a), 3.30(a), 3.31(a), 3.32(a)) and Swaptions



Source: Baum, Maliszewski, Griebler, 2017.

Figure 3.27: Conf3: Methodology followed in the evaluation of enterprise applications in multi-tenancy scenario.

(Figures 3.29(j), 3.30(j), 3.31(j), 3.32(j)) workloads are mentioned by Chasapis et al. (2016) as simple applications that contain a single parallel loop, so running in all users, showed little difference between the virtualization even in the multi-tenancy. Facesim (Figures 3.29(e), 3.30(e), 3.31(e), 3.32(e)) also has a little variation, especially in this scenario where the bus contention does not impact at low treads count (BHADAURIA; WEAVER; MCKEE, 2009). Bodytrack (Figures 3.29(b), 3.30(b), 3.31(b), 3.32(b)), Dedup (Figures 3.29(d), 3.30(d), 3.31(d), 3.32(d))and Vips (Figures 3.29(k), 3.30(k), 3.31(k), 3.32(k)) executing with containers exhibits notorious poor performance compared to native and KVM, caused by the container manipulation on another host, it is also interesting to note that in Vips (Figures 3.29(k), 3.30(k), 3.31(k), 3.32(k)), the first LXC user apparently gains some form of preference having a better performance in relation to other tenants. Canneal (Figures 3.29(c), 3.30(c), 3.31(c), 3.32(c)) and Freqmine (Figures 3.29(g), 3.30(g), 3.31(g), 3.32(g)) in this environment have demonstrated overhead on KVM virtualization for all users and few variations between native and LXC users, which is expected due the fact that these workloads are CPU intensive and show aggressively data race. This could produce different results with native



Source: Baum, Maliszewski, Griebler, 2017.

Figure 3.28: Description of Parsec multi-tenancy scenario with same applications on 4 concurrent users.

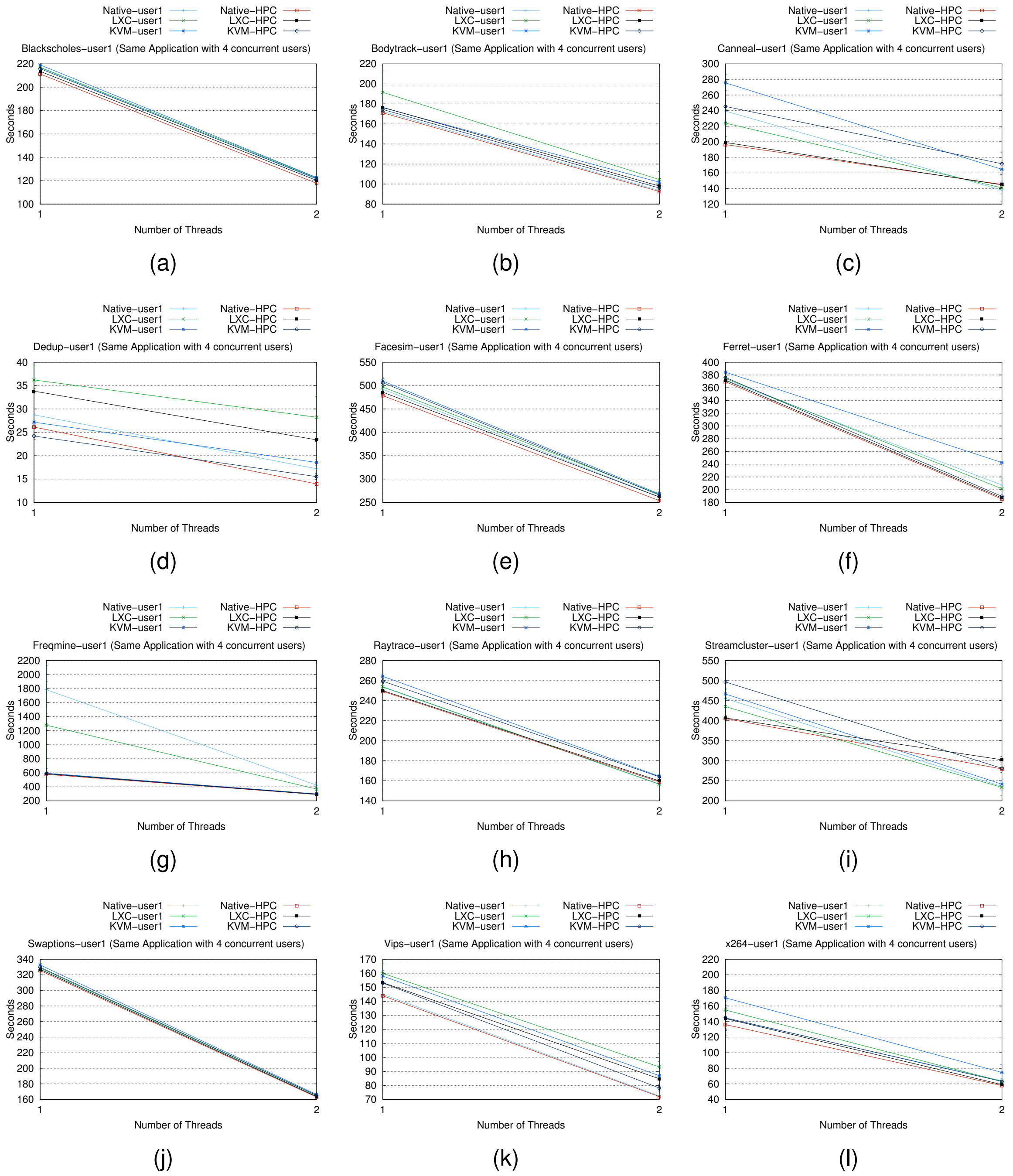
users and with LXC virtualization, especially with the level of isolation offered by it, which seems to be conducive to the same application in the tenant environment.

Freqmine also demonstrates a performance boost using KVM virtualization in all users, a similar result is shown by the authors Bhadauria, Weaver and McKee (2009), which explains that this can happen since the threads can be swapped into the core when one thread is stalled waiting data. Although Ferret (Figures 3.29(f), 3.30(f), 3.31(f), 3.32(f)) uses a pipeline parallelism similar to use in dedup, with serial portions in the input and output stages, it demonstrated a different result with increasing KVM overhead. Essentially this is because Ferret keeps the database of images loaded in memory, not the disk like Dedup (BIENIA, 2011).

Raytrace (Figures 3.29(h), 3.30(h), 3.31(h), 3.32(h)) and x264 (Figures 3.29(l), 3.30(l), 3.31(l), 3.32(l)) representing the rendering and media processing application domains, and demonstrate some KVM overhead resulting from the necessity for these workloads to have many data computing, using synchronizations points and context switching, a well-known weakness of the KVM (MUKHEDKAR; VETTATHU; CHIRAMMAL, 2016). In according with Bhadauria, Weaver and McKee (2009) Streamcluster (Figures 3.29(i), 3.30(i), 3.31(i), 3.32(i)) uses 52% of all its operation to compute data using floating point operations, this is similar to Blackscholes, so the tests in this environment show to be a case with low distinction between the virtualizations.

3.3.4.3.2 Different Applications

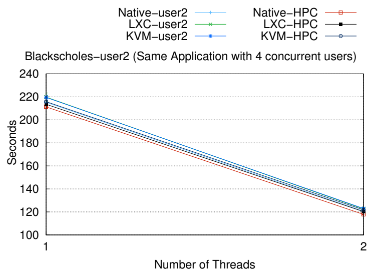
In this test, the workloads presented in the Parsec suite are distributed among the four users to represent another form of multi-tenancy environment, where applications are different (Figure 3.33). Thus, different results are expected due to the variety of this environment and, as a consequence, a performance characterization can be achieved.



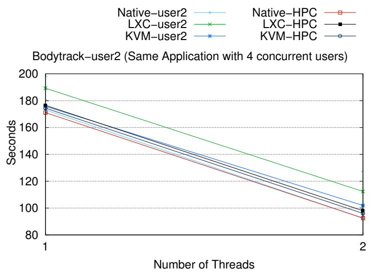
Source: Baum, Maliszewski, Griebler, 2017.

Figure 3.29: Multi-tenancy-User1 Scenario (Same Applications with 4 concurrent users).

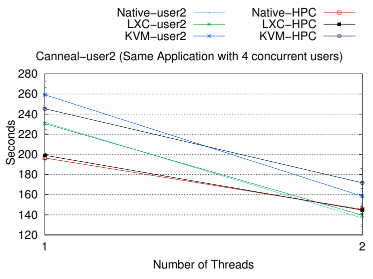
The Blackscholes (Figure 3.34(a)), Facesim (Figure 3.35(b)) and x264 (Figure 3.37(c)), showed an insignificant difference in the presented multi-tenancy environment, proving that in this case, these applications suffer little interference and can be



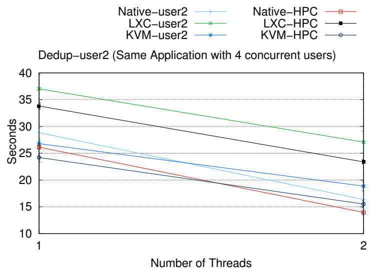
(a)



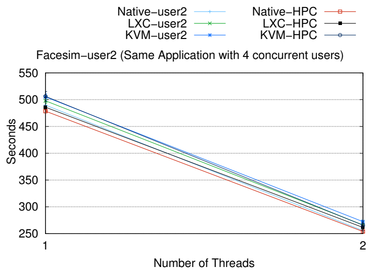
(b)



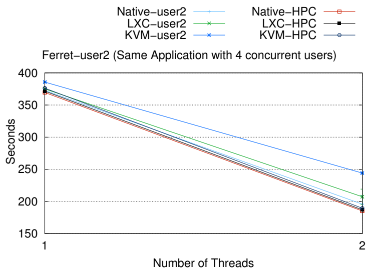
(c)



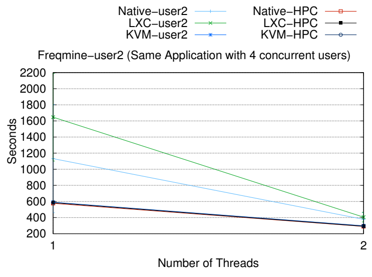
(d)



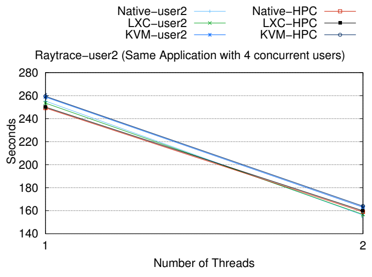
(e)



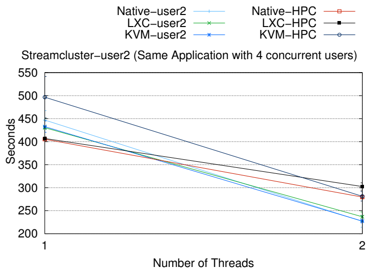
(f)



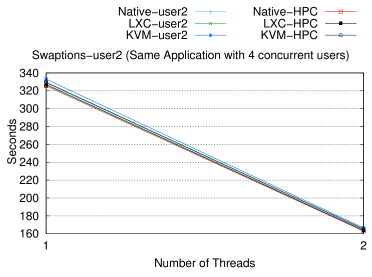
(g)



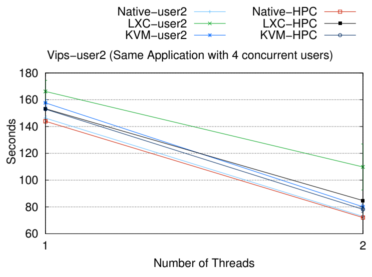
(h)



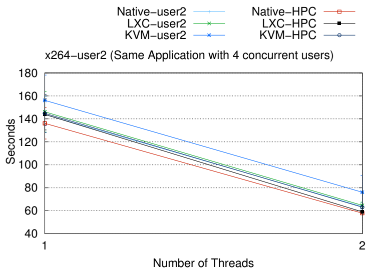
(i)



(j)



(k)

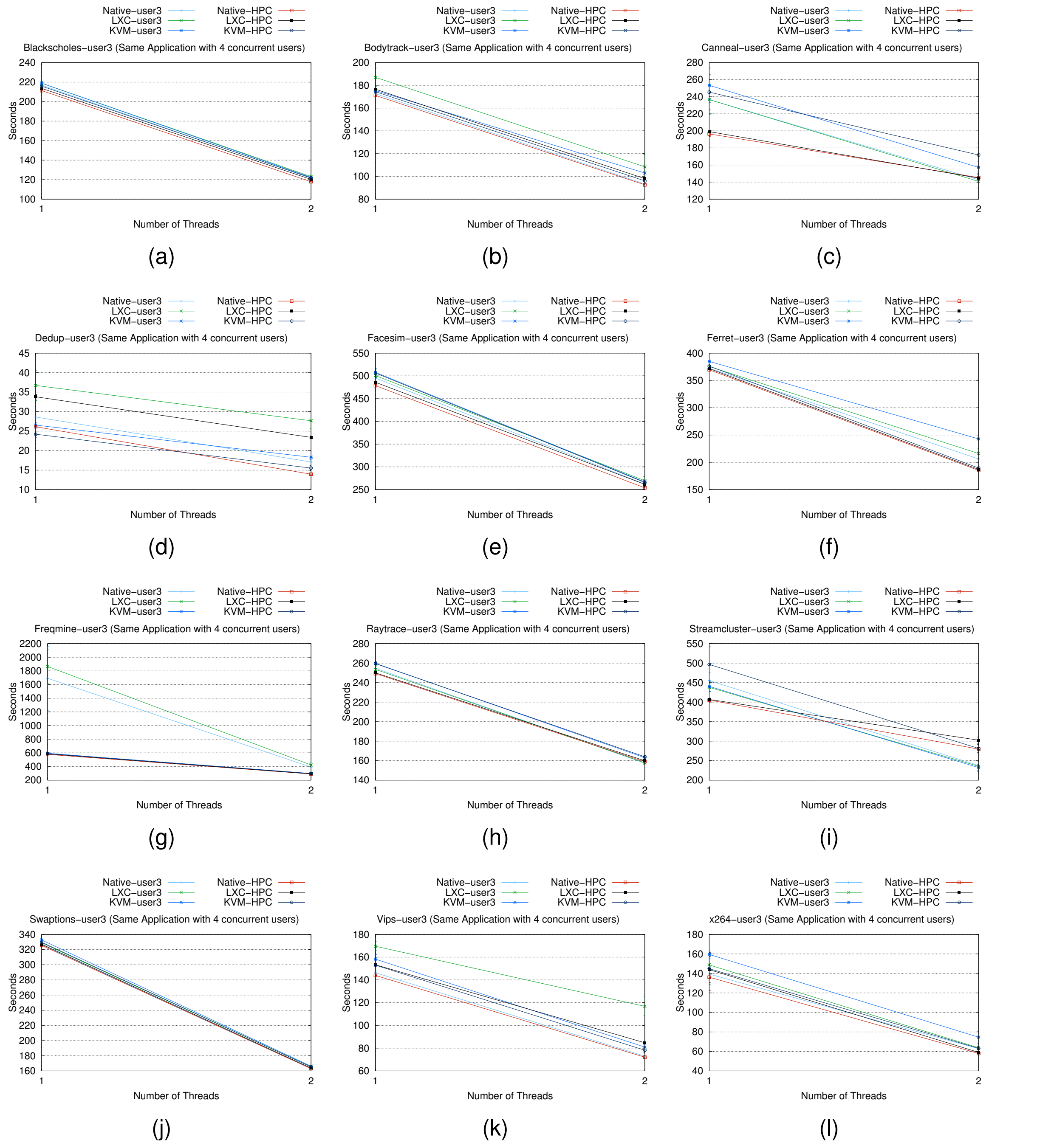


(l)

Source: Baum, Maliszewski, Griebler, 2017.

Figure 3.30: Multi-tenancy-User2 Scenario (Same Applications with 4 concurrent users).

effectively used. The Bodytrack (Figure 3.35(a)), Dedup (Figure 3.34(c)) and Vips (Figure 3.36(b)) applications in KVM virtualization perform similarly to the native environment, and LXC containers are significantly different, especially in the case of Dedup,

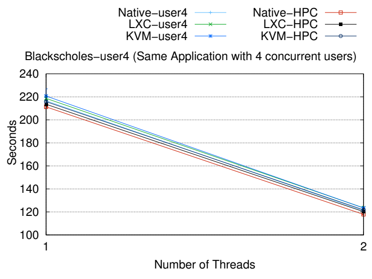


Source: Baum, Maliszewski, Griebler, 2017.

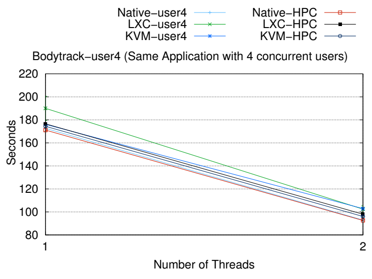
Figure 3.31: Multi-tenancy-User3 Scenario (Same Applications with 4 concurrent users).

where the deployed cloud is not favorable to such workload.

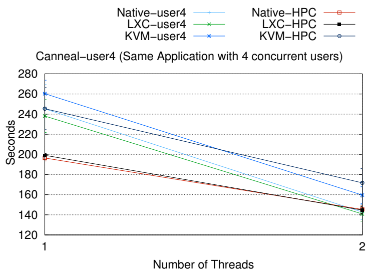
In the case of the Canneal (Figure 3.34(b)) and Freqmine (Figure 3.35(c)) ap-



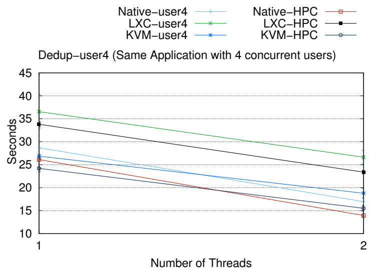
(a)



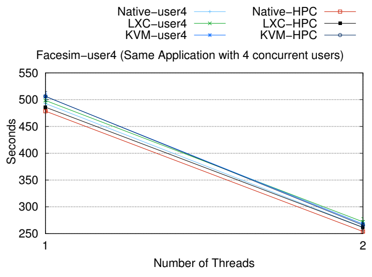
(b)



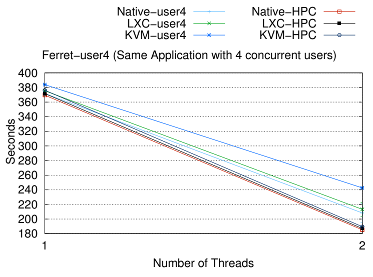
(c)



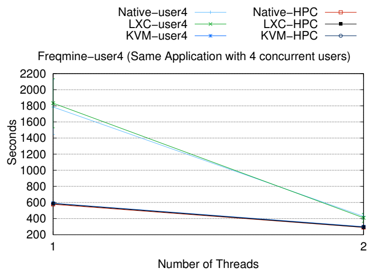
(d)



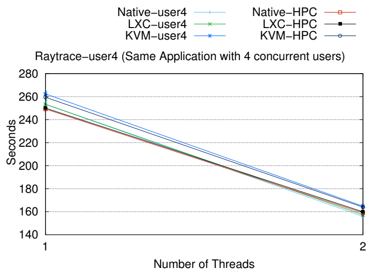
(e)



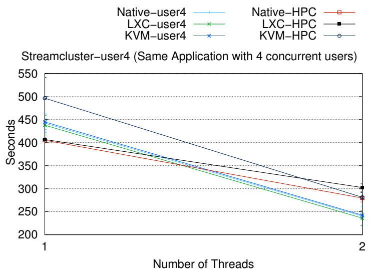
(f)



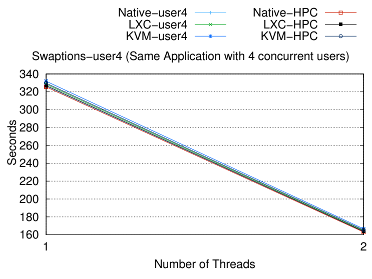
(g)



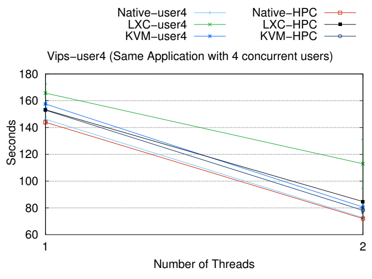
(h)



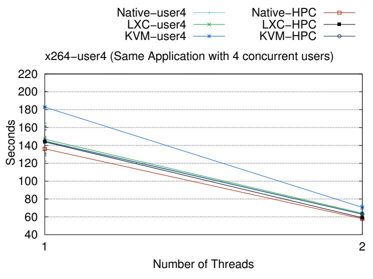
(i)



(j)



(k)

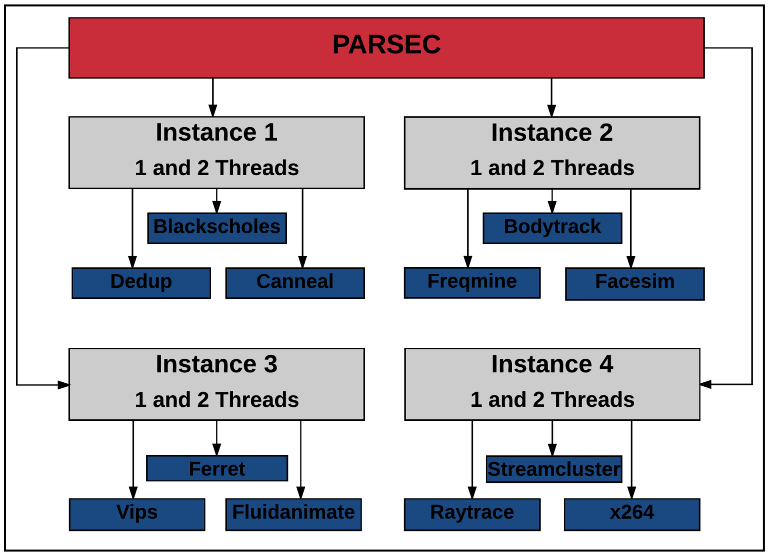


(l)

Source: Baum, Maliszewski, Griebler, 2017.

Figure 3.32: Multi-tenancy-User4 Scenario (Same Applications with 4 concurrent users).

plications running on KVM-based cloud performs quite better than the others, it is very interesting performance in these workloads, mainly because LXC and native are not as different as shown in the another environment, which reveals that in the same ap-



Source: Baum, Maliszewski, Griebler, 2017.

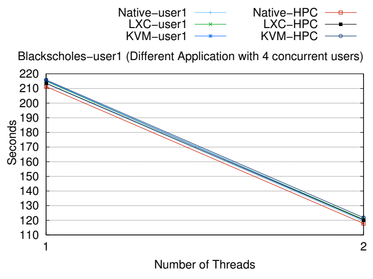
Figure 3.33: Description of Parsec multi-tenancy scenario with different applications on 4 concurrent users.

plications, four workloads compete aggressively for machine resources, deteriorating performance and runtime for all users. Thus, in this environment with different tenants this has not occurred, especially in the KVM virtualization, which has a better resource isolation.

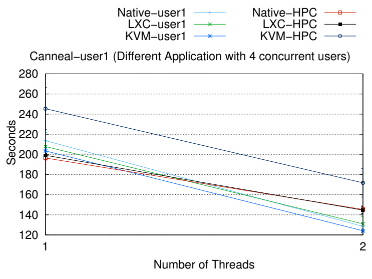
Ferret (Figure 3.36(a)) demonstrated a crescent overhead using KVM, where the two threads induce significant overhead due to the previously discussed characteristics of this workload. The KVM-based cloud executing the workloads Raytrace (Figure 3.37(a)) and Streamcluster (Figure 3.37(b)) has demonstrated that it does not be able to port all the necessary processors resources and memory management for these workloads, so the applications do not perform as well as LXC.

3.4 SCIENTIFIC APPLICATIONS

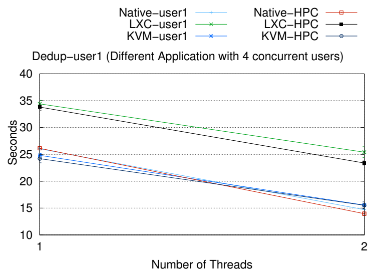
The scientific field was represented by the NAS Parallel Benchmark suite, which contains a small set of applications. These applications are described in the



(a)



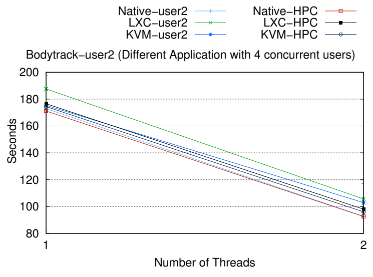
(b)



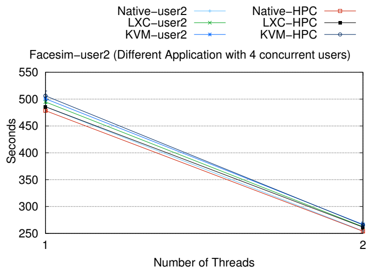
(c)

Source: Baum, Maliszewski, Griebler, 2017.

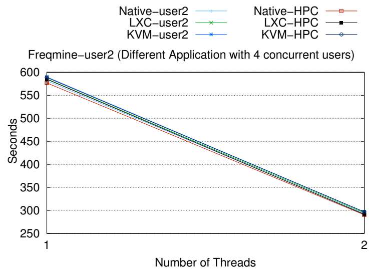
Figure 3.34: Multi-tenancy-User1 Scenario (Different Applications with 4 concurrent users).



(a)



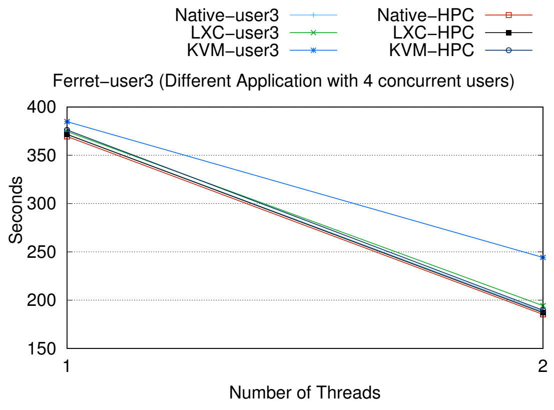
(b)



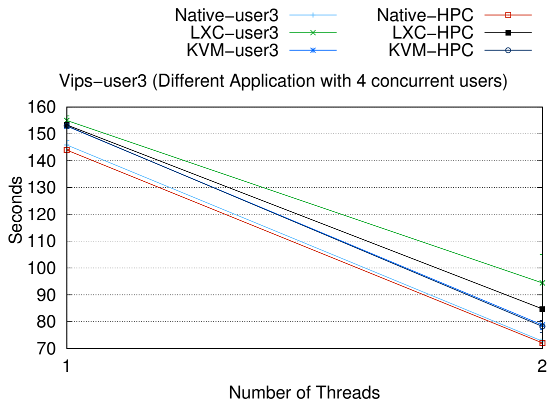
(c)

Source: Baum, Maliszewski, Griebler, 2017.

Figure 3.35: Multi-tenancy-User2 Scenario (Different Applications with 4 concurrent users).



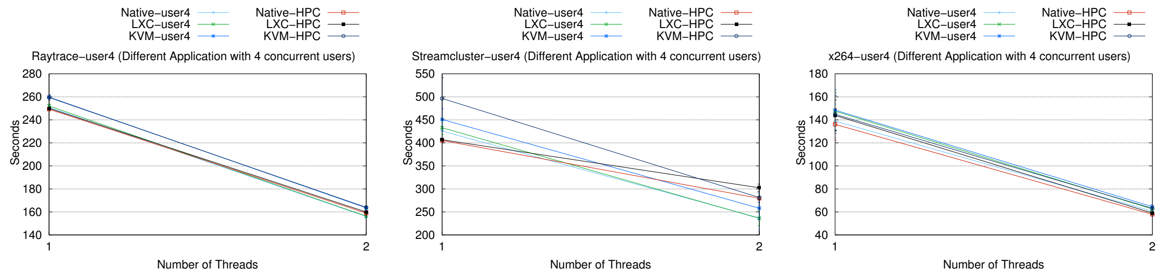
(a)



(b)

Source: Baum, Maliszewski, Griebler, 2017.

Figure 3.36: Multi-tenancy-User3 Scenario (Different Applications with 4 concurrent users).



(a) (b) (c)

Source: Baum, Maliszewski, Griebler, 2017.

Figure 3.37: Multi-tenancy-User4 Scenario (Different Applications with 4 concurrent users).

Section 2.3.2. In these test scenario, OMP programming model was used for the NPB benchmarks BT, CG, EP, FT, IS, LU, MG, SP and UA. In addition, it was employed the class B, which represent the intensity of the workloads. A better description of the functionalities of these applications can be seen in 2.3.2.

3.4.1 Performance Characterization

To discover and characterize how applications react to increasing threads, all 9 applications were monitored in the native environment, creating a baseline. The collected data refers to memory usage, core load, cache miss, and I/O operations. They are described in the following subsections.

3.4.1.1 Core Load

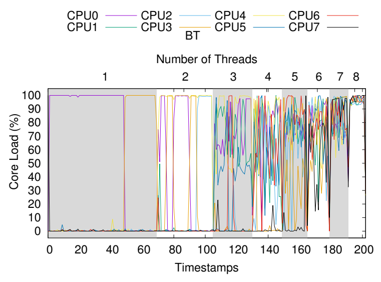
The core load utilization of the workloads confirms an important description of the NPB suite addressed in the literature. The NPB suite is high performance computing oriented and explore its workloads parallelism. As we can see in Figure 3.38, all applications have increased core utilization when more threads are scheduled. In addition, when we look at the eighth thread of all applications, the maximum utilization of the processor is noticeable and, due to this, a reduction in execution time. Another relevant aspect is that when we have an increase of threads, the use of the core has a quickly reduction and soon thereafter a quickly increase in its use. This suggests that when the threads are scheduling, a small loss of performance occurs, almost imperceptibly, and soon after, when the other thread replaces, there is an increase in performance. One of the main contributions is that all applications behave differently.

The IS application is an important case. It differs from other applications because of its faster execution time, represented in some threads in milliseconds. Because of this, to create it graph, a small data collection could be made, and we can note in Figure 3.38(e) that, as the number of threads increases, the use of the core load also increases, a reaction already expected. However, here it is clear that when threads are scheduling, there is a minimal loss of performance, even in milliseconds, which is known as context switching.

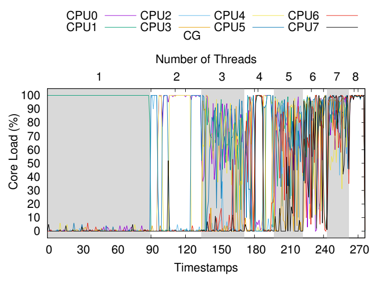
3.4.1.2 Memory Usage

The workloads memory usage of the NPB suite (Figure 3.39) had a specific variance in each application. It is important to point out that the memory scale represented in each application is in Kilobytes, to evidence the increase or decrease of memory usage even in small amounts. The BT (Figure 3.39(a)) is an application that focus in the floating point performance, which uses less than 200 Megabytes during its execution along 8 threads. We note that in the second thread there is a large increase in memory usage, but in fact this is not representative because the scale in this application is sized in 50 to 50 kilobytes, that is, 0.1 Megabytes. In addition, it is noticed that there is a small increase in memory usage which shows that it follows the increase of performance presented by the application in the next threads.

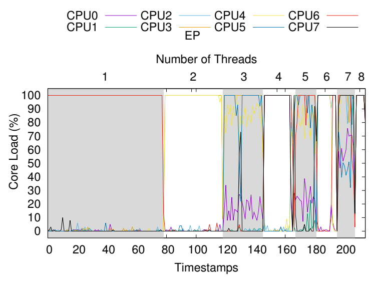
The CG (Figure 3.39(b)) is an intensive memory application, which uses a linear amount of memory in any of the 8 threads. The average memory usage in this application is about 350 Megabytes. In addition, we can note that when threads are



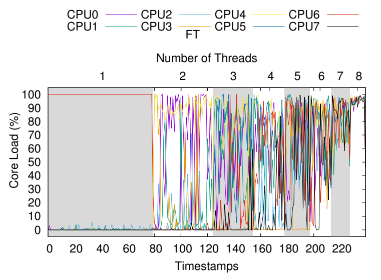
(a)



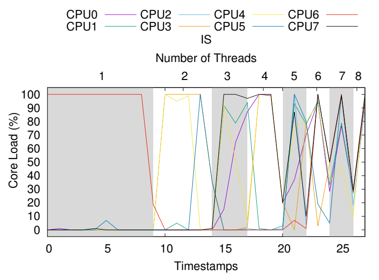
(b)



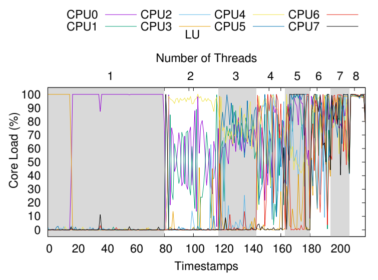
(c)



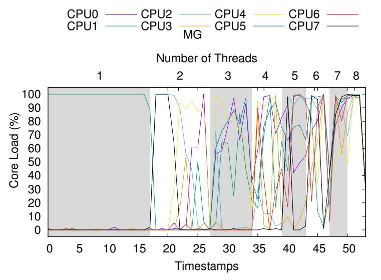
(d)



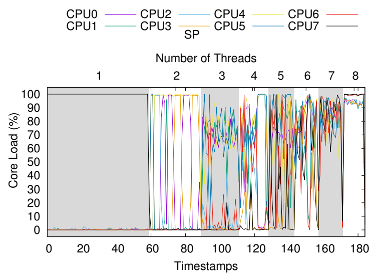
(e)



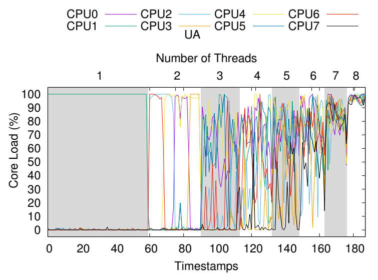
(f)



(g)



(h)



(i)

Source: Baum, Maliszewski, Griebler, 2017.

Figure 3.38: Performance Characterization of Scientific Applications (Core Load).

scheduling, there is low memory usage, probably resulting in performance loss. Also, when the next thread replaces, memory usage returns to the linear quantity.

The EP (Figure 3.39(c)) is a non-intensive memory application, which exhibits a variation in memory usage taking into account the scale in Kilobytes. It uses about 100 megabytes. It is noticeable that memory usage increases while the number of threads increases, resulting in shorter execution time and performance gain.

The FT (Figure 3.39(d)), is characterized as a intensive memory application, which uses a linear amount of memory along the 8 threads. The average memory usage is about 13 Gigabytes, the highest usage among all NPB suite applications. Also, when threads are scheduling a small decrease in usage is noticeable, but returns quickly to their linear consumption of memory.

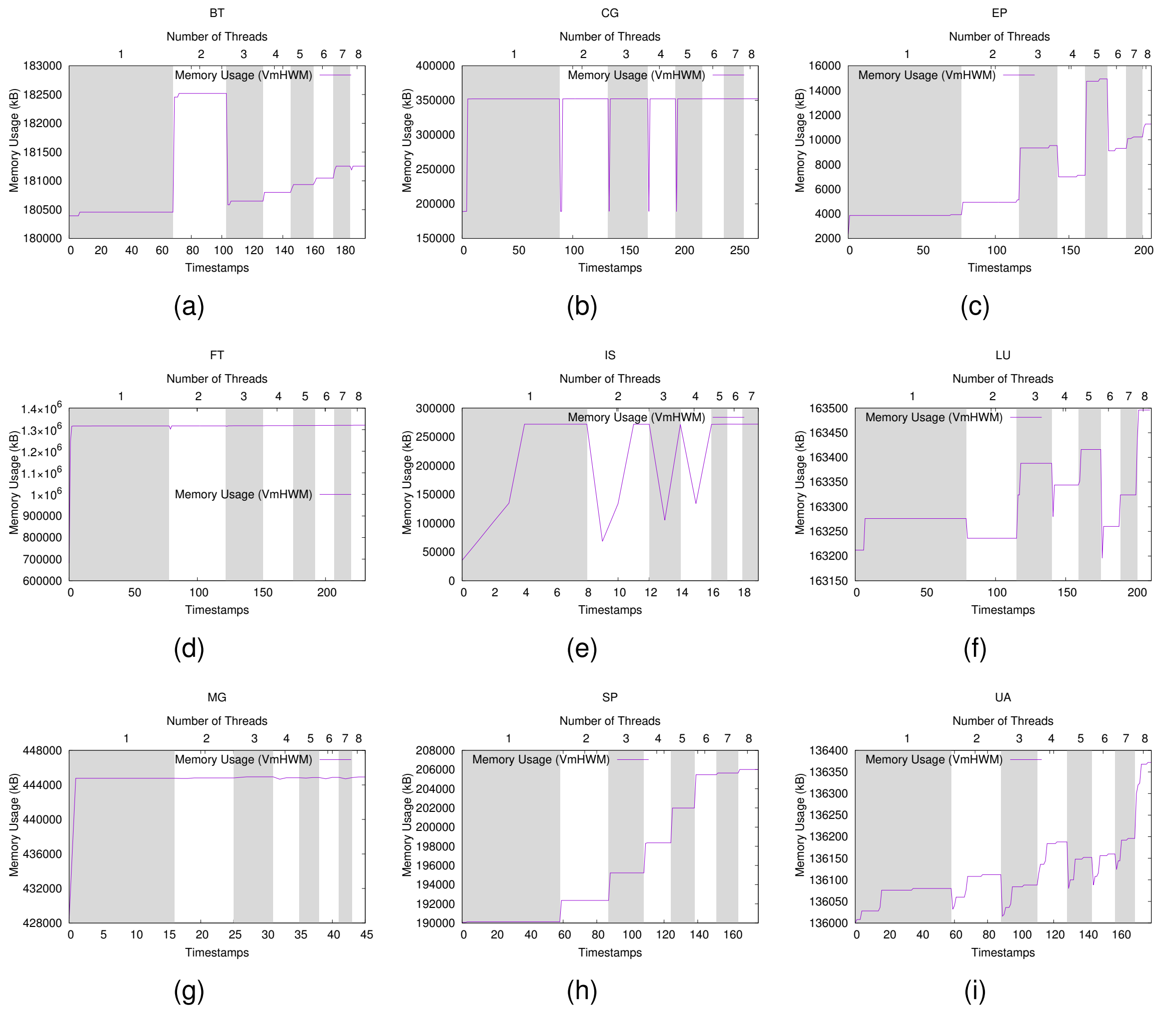
The IS (Figure 3.39(e)) is an intensive memory workload with focus in integer performance, that has significant variation in memory usage and precisely a reduction in memory usage while the threads were scheduling. The average memory usage is approximately 250 Megabytes. Due to its fast execution time, IS suffers more intensely when threads are scheduling because it needs to do this in milliseconds. Figure 3.39(e), show only the first seven threads because the eighth thread did not have enough data collection to form the line graph.

The LU (Figure 3.39(f) shows a strange variance in memory usage during its execution, but if we take into consideration the scale of the graph that is sized from 50 to 50 Kilobytes, that variation becomes insignificant. The average memory usage is 163 Megabytes. We can observe a use of scalar memory, which emphasizes a performance gain. Also, when threads are scheduling, a reduction in memory usage is noticeable.

The MG (Figure 3.39(g)) is an intensive workload with a small working set. The average usage is about 444 megabytes. It presents a linear use of memory and can be seen a small variation when the threads are scheduling.

The SP (Figure 3.39(h) is a memory intensive workload and has the characteristic of a scalar use of memory. The average memory usage in this application is approximately 202 Megabytes. It is clearly perceptible that while the number of threads is increasing, memory usage shows a scaling increase in its use, what suggest a gain of performance.

Finally, the UA (Figure 3.39(i)) is an memory-intensive workload that uses an average of 136 Megabytes. In this application it is visible the increase in memory usage, almost being a scalar increase. Additionally, a decrease in memory usage is also evident when the threads are scheduling. The scale of the graph should also be highlighted, which uses 50 to 50 kilobytes, so small variations do not represent anything significant.



Source: Baum, Maliszewski, Griebler, 2017.

Figure 3.39: Performance Characterization of Scientific Applications (Memory Usage).

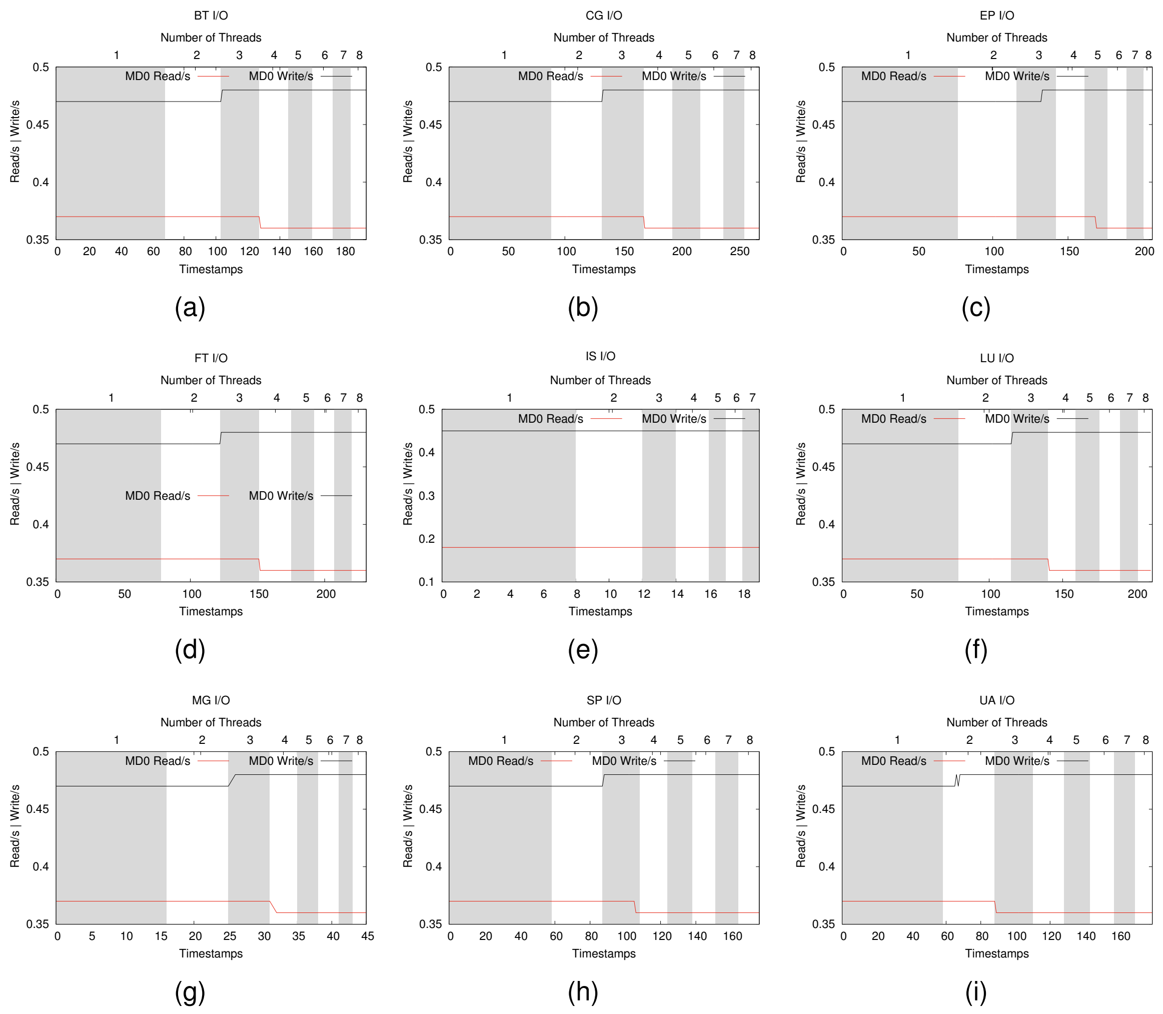
3.4.1.3 Disk I/O

The I/O disk is represented in this evaluation by read and write per second. As explained before, the NPB suite using the OMP implementation are processor and memory intensive applications, so all programs make sporadic disk usage (GUPTA; MILOJICIC, 2011). Therefore it was expected that the I/O disk would not be significantly used as well as demonstrated by the authors Cheveresan et al. (2007) and Antoniou (2012). In fact, as we can see in Figure 3.40, disk I/O does not provide significant variations. Moreover, in this evaluation, the scale of the figure is small to represent the differences and, as we can see, the readings and the writings have almost linear results.

3.4.1.4 Cache Miss

Cache miss has an important role in the NPB suite, as it is can significantly affects applications behavior, especially in high-performance of applications. Accordingly, the authors Cheveresan et al. (2007) suggest that HPC applications are governed by the cache miss latency and the ability to transmit data to the processor. Thus, the Figures 3.41(a) and 3.41(b) shows the cache miss of the NPB applications, which were benchmarked in the native environment. As we might expect, CPU-intensive applications such as CG, SP and UA, and which have large working sets as LU, have high cache miss rates (CHENG; CHEN, 2013a), especially on the first and second threads. Thus, these memory-intensive applications (Figures 3.39(b), 3.39(h), 3.40(i)), and having significantly larger working sets (3.39(f)), frequently require to others levels of cache instructions, causing an increase of cache misses (FERDMAN; ADILEH; KOCBERBER; VOLOS; ALISAFAEE; JEVDJIC; KAYNAK; POPESCU; AILAMAKI; FALSAFI, 2012).

However, as we can see, in the second thread, almost all applications have a significant increase in the number of cache misses. This can occur because of the ad-



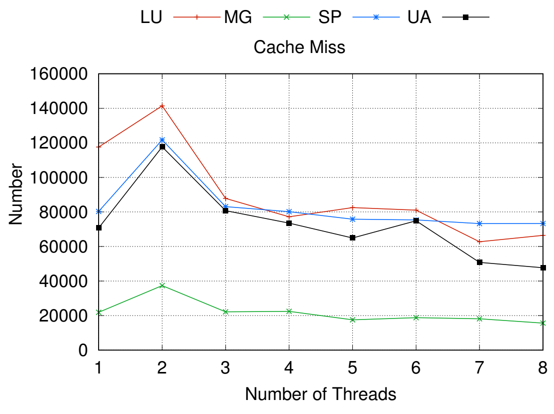
Source: Baum, Maliszewski, Griebler, 2017.

Figure 3.40: Performance Characterization of Scientific Applications (I/O).

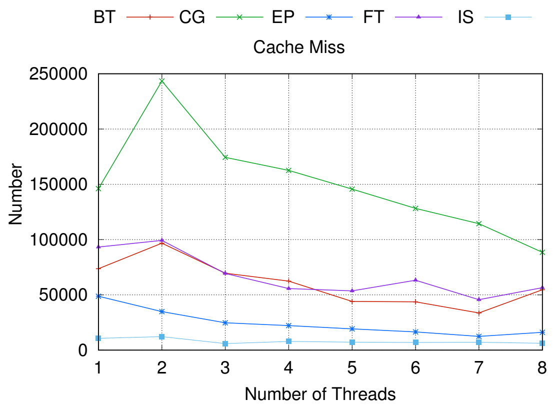
dition of threads, so the cache must be quickly replaced with new instructions (data) of the workloads, this concurrency for cache causes more cache misses (BIENIA, 2011). In contrast, MG, IS, and EP applications that have small working sets show only a small number of cache misses.

3.4.2 HPC Scenario

In this environment, benchmarks were performed using full native machine resources in only one instance. A script with a loop of repetitions was used to execute



(a)



(b)

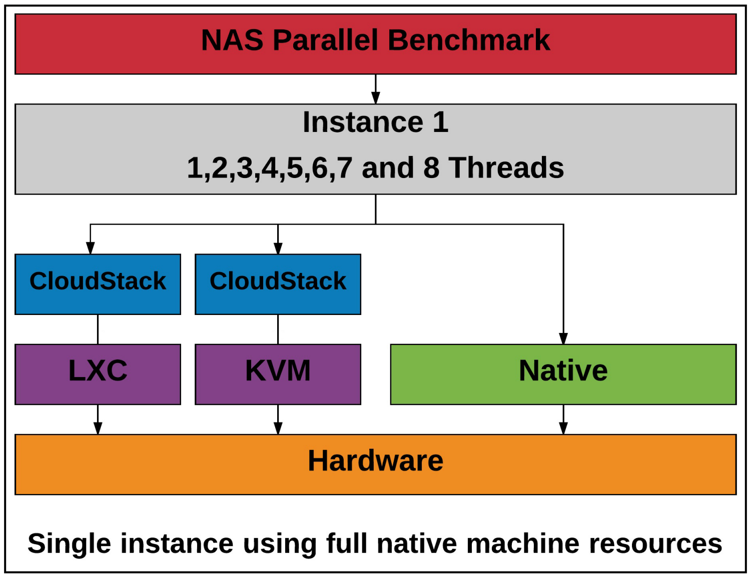
Source: Baum, Maliszewski, Griebler, 2017.

Figure 3.41: Performance Characterization of Scientific Applications (Cache Miss).

the benchmarks one after another, that is, the benchmarks were never run at the same time.

The NAS Parallel Benchmark was executed in each instance, scheduling from 1 to 8 threads, totaling eight separate runs with 15 executions each. In addition, two cloud-based were deployed, one with the CloudStack LXC-based cloud (container virtualization) and other with the CloudStack KVM-based cloud (full virtualization). Finally, the Native environment was benchmarked to compare it with the two based clouds. This methodology can also be seen in the Figure 3.42.

BT is an application that focuses on floating point performance. Represents a computation in a regular 3D grid and executes a solution of a tridiagonal system of equations (CHEVERESAN; RAMSAY; FEUCHT; SHARAPOV, 2007). This application experiences a loss of performance over CPU usage (Figure 3.38(a)), especially in odd-numbered threads. A thorough study of this problem is necessary to justify its occurrence. Referring to memory (Figure 3.39(a)), we can see that BT has a growth in its use (not considered representative, given the size of its scale) as the number of threads increases. The results of the cache miss (Figure 3.41(b)) show a loss considered medium, which decreases as the number of threads increases. The execution


NAS Parallel Benchmark
Instance 1
1,2,3,4,5,6,7 and 8 Threads
CloudStack
CloudStack
LXC
KVM
Native
Hardware
Single instance using full native machine resources

Source: Baum, Maliszewski, Griebler, 2017.

Figure 3.42: Methodology followed in the evaluation of scientific applications in high performance scenario.

time represented in the Figure3.43(a) shows that the LXC-based cloud (container) got results very close to the native environment, the same result of Xavier et al. (2013) and Cheng and Chen (2013b). In addition, a overhead is perceived in the KVM-based cloud and this was already expected due to the fact that the virtualization layer adds more instructions that need to be managed by the CPU. This involves more information to treat and bufferization by the CPU that cause a degradation of the receive throughput rate compared to native performances Reddy and Rajamani (2014), and can also be targeted to context switch of the core load.

CG concentrates on irregular communication and is an intensive memory application (Figure 3.39(b)). This application has performance loss related to the core load (Figure 3.38(b)), specifically in the third thread. What caught our attention is the lack of cache (Figure 3.41(b)) where CG presents the highest rates among all applications in the NPB suite. Another relevant feature is that the CG uses a large number of small messages Saini et al. (2012), and has a large set of work, this may be the reason for the high number of cache miss. In Figure 3.43(b) it is shown that performance in the native environment and in the LXC-based cloud is very close and a overhead is noticed in the KVM-based cloud. This overhead can be directed among other factors for the high number of cache misses and also the context switch presented. In addition, the authors Regola and Ducom (2010) emphasized that virtualization penalty is most significant in benchmarks that requires large amounts of communication or memory access.

SP is an application focused on floating point performance. Wijngaart, Sridharan and Lee (2012) suggests that SP and BT are functionally identical, and any derived analysis for BT could be applied to SP. This application also presents problems related to its use of core load (Figure 3.38(i)) mainly in threads with odd numbers, a problem that has already been presented and needs to be better studied. A noticeable characteristic is the scalar use of memory (Figure 3.39(h)) that follows the increase of threads. In addition, the SP has high cache miss rates (Figure 3.41(a)) that had a considerable increase in the second thread and soon afterwards shows a stabilization, but still high. As we can see in Figure 3.43(c), the LXC cloud-based performance is almost identical to the native environment. In addition, an overhead is displayed in the KVM based cloud and this can be directed among other factors to the cache miss and its CPU usage. In addition, the authors Regola and Ducom (2010) emphasize that SP is an intensive workload in memory, so it requires large amounts of memory access and ends up suffering virtualization penalties.

EP is a floating-point performance application. As suggested by Vogel et al. (2016), the EP has no task dependency. The main characteristic of this application can be seen in its use of core load (Figure 3.38(c)) which it presents a high performance gain, using CPU resources intelligently. In addition, the cache miss (Figure 3.41(b)) and memory usage (Figure 3.39(c)) demonstrates that this application requires less memory capacity and has one of the lowest cache miss rates, respectively. As we can see in Figure 3.43(d), the native environment and the LXC and KVM cloud based are identical. This application presents the actual context of parallel processing. It is noticed that in each increase of threads, the execution time has decreased considerably. This means that the performance gain follows at levels close to the recommended one, with small differences between the environments. This performance can be directed primarily to the way this application uses processor resources. In addition, EP have short execution times (HONG; KIM; KIM; PARK; YOO, 2014) and small set of work, so overhead is reduced. Moreover, the authors Hashimoto and Aida (2012), says that the performance degradation in EP is minimum, because EP requires less memory capacity.

FT is a memory-intensive application and is focused on "everything for all communication". The Figure 3.39(d) shows that the FT uses about 13 Gigabytes for its execution, confirming to be the application that has the highest use of memory in the NAS suite. Regarding to core load (Figure 3.38(d)), this application presents some loss in performance, mainly if we observe the context switch that happens. In addition, FT presents medium cache miss rates 3.41(b). The execution time (Figure 3.43(e)) shows that FT has good gain of performance in the virtual environment, with results identical to the native environment. A overhead is noticed in the KVM based cloud, especially in the first threads. This overhead is reduced as the number of threads increases. This behavior is also displayed in its cache miss evaluation (Figure 3.41(b)), where it has a higher number of cache misses in the first threads and decreases as the number of threads increases. This suggests among other factors that FT in the full virtualization (KVM) environment suffers penalties in the first threads due to its cache miss and unbalanced use of cores.

UA is an application that focuses on irregular communication. It is characterized as memory intensive (Figure 3.39(i)) and has high rates of cache miss (Figure 3.41(a), mainly in the second thread. Regarding to its core load, UA demonstrates problem in the use of cores, especially in odd threads and is also noticeable the context switch. Its execution time presents results very close in the three environments. However, a overhead is perceptible in LXC and KVM based cloud, mainly in second thread. This can be addressed among other factors to its high rates of cash misses and mainly to the context switch that is presented in the core load usage.

IS is an intensive memory application with a focus on integer performance. This application has some specific characteristics, such as the smallest working set, the fastest execution time, and the lowest cache miss rate (Figure 3.41(b)) among all applications in the NPB suite. In addition, the evaluation of the core load (Figure 3.38(e)) shows problems in the direction of loads to specific CPU's, a contest switch problem. As we can see in Figure 3.43(g), the LXC-based cloud and the native environment are practilly identical. We can perceive a overhead in the KVM based-cloud. As suggested by Huang et al. (2006) applications with intensive communication, such as IS a CG, has worse performance in the virtualized environment. In addition, the author Strazdins et al. (2012) also suggests that IS does not scale well. This result is seen in many articles in the literature, and overhead can be addressed among other factors for the short execution time of IS (WALTERS; CHAUDHARY; CHA; GUERCIO JR; GALLO, 2008), and especially for the use of the processor, demonstrating that this application does not have an efficient scheduling.

LU is an application with focus on irregular communication and also it has a large working set. As we can see in Figure 3.38(f), LU does not have a high gain in performance in the first few threads regarding core load. In addition, it has a strange variance in memory usage (Figure 3.39(f)), but shows an increase in use as the number of threads increases. The cache miss of LU (Figure 3.41(a)) is one of the highest in NPB suite, reaching its peak in the second thread. The LU execution time (Figure 3.43(h)) shows nearly equal results in the LXC-based cloud with the native environment. A overhead is noticed in the fully virtualized environment (KVM-based cloud), especially in the first 3 threads. It can be explained, among other variants, to its high cache miss rate, which reaches its peak in the first 3 threads. In addition, as explained by Regola and Ducom (2010) LU requires large amounts of memory access so it suffers from virtualization penalties.

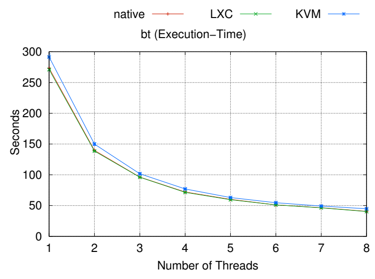
MG is a workload with focus on regular communication. It does not present a good balance in the use of the core (Figure 3.38(g)) occurring several context switch in its execution. The memory results (Figure 3.39(g)) shows that this application is a memory intensive. In addition, it presents good results in cache miss (Figure 3.41(a)). In its execution time (Figure 3.43(i)) MG presents a considerable gain in performance as the threads increase. The native and the LXC-based cloud shows the same result and a overhead is noticed in KVM-based cloud. This can be addressed, among a set of factors, to the balance in utilization of the cores processor, whats suggests that this application have problems in load balancing regarding to CPU usage. In addition, as this benchmark requires large amounts of memory access Regola and Ducom (2010), the virtualization penalty is most significant.

3.4.3 Multi-tenancy Scenario

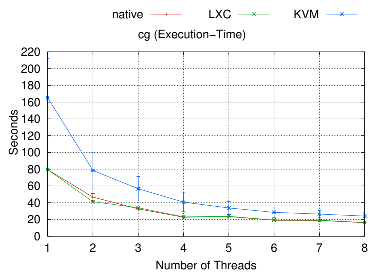
The multi-tenacy environment was build thinking on running applications on concurrent users. Its definition can be seen in the section 2.2.11. We create a methodology of six configurations. Three of them executed concurrent same applications and the other three executed concurrent different applications. The next sections will describe this six configurations.

3.4.3.1 Configuration 1

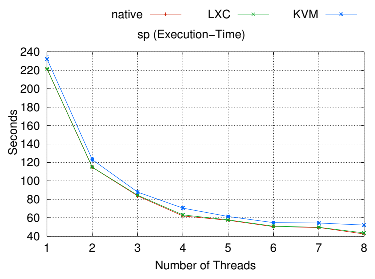
This configuration can be seen in Figure 3.44, which NAS Parallel Benchmark was executed in two concurrent instances. This instances used 1,2,3 and 4 threads totalizing 8 threads because the applications were executed at same time. In addition, in the cloud environment the service offer was splitted between the two instances.



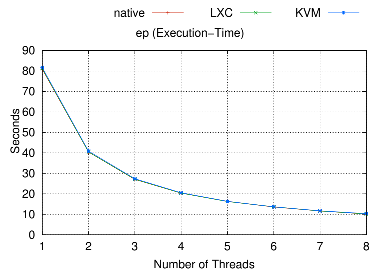
(a)



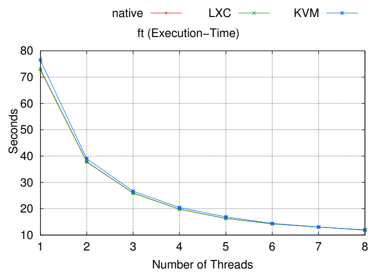
(b)



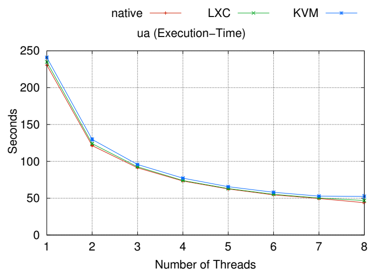
(c)



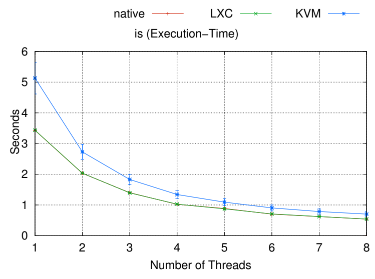
(d)



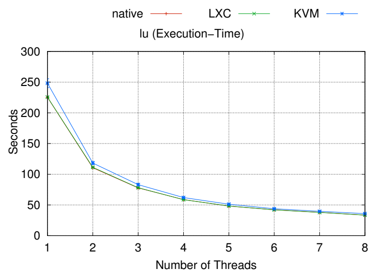
(e)



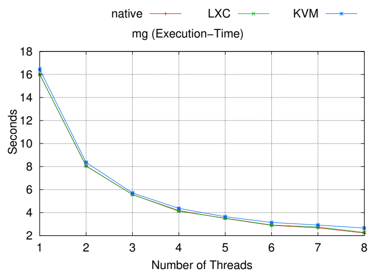
(f)



(g)



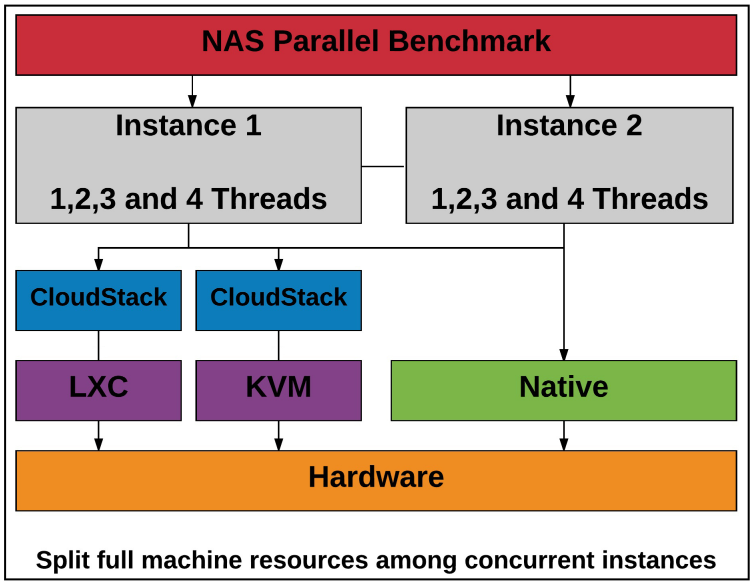
(h)



(i)

Source: Baum, Maliszewski, Griebler, 2017.

Figure 3.43: High Performance Scenario.

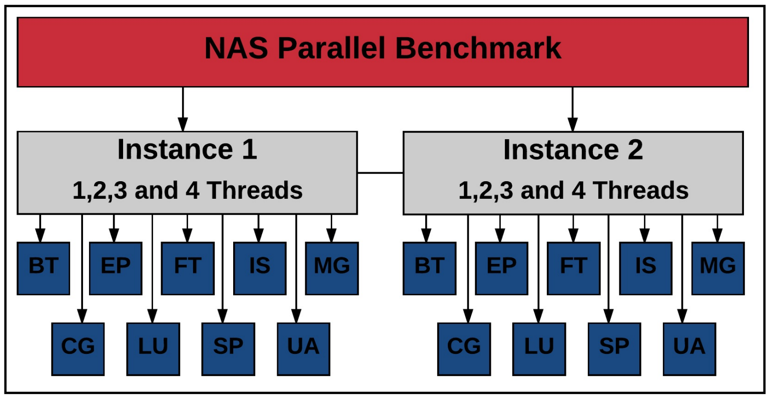


Source: Baum, Maliszewski, Griebler, 2017.

Figure 3.44: Conf1: Methodology followed in the evaluation of scientific applications in multi-tenancy scenario.

3.4.3.1.1 Same Applications

In this methodology (Figure 3.45), same applications were splitted over two concurrent instances in the LXC and KVM based clouds and in the Native environment over two users.



Source: Baum, Maliszewski, Griebler, 2017.

Figure 3.45: Description of NPB multi-tenancy scenario with same applications on 2 concurrent users.

The applications BT, EP, FT and MG show small differences results in their execution time, being practically identical in the three environments (Native, LXC and KVM based clouds) and even very similar to HPC scenario. CG shows an overhead in the KVM and also overhead in the LXC. This result demonstrates that this application with high cache miss rates (Figure 3.41(b)) and context switching (Figure 3.38(b)) have it execution even more degraded in the concurrency environment.

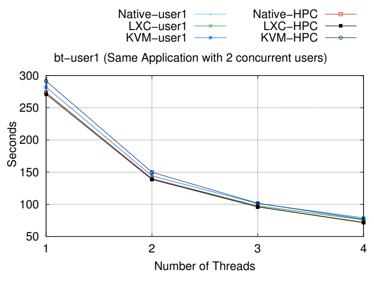
IS (Figure 3.46(g) and 3.47(g)) shows a KVM overhead and also an LXC obtaining better results than the native environment. The overhead presented in the KVM-based cloud demonstrates that this application does not have an efficient scheduling Gupta et al. (2013) and this can be seen in the use of the core load (Figure 3.38(e)). As the native environment does not provide isolation for its users, they compete for the use of resources, which may explain the result that shows the native environment achieving worse results than those presented by the LXC-based cloud.

On the other hand, LU (Figure 3.46(h) and 3.47(h), SP (Figure 3.46(c) and 3.47(c)) and UA (Figure 3.46(f) and 3.47(f)) applications present KVM overhead and LXC with results close to the native environment. These applications have similar problems regarding to high cache miss rates (Figure 3.41(a) and 3.41(b)) and also problems with high rates of contest switching. Thus, in the multi-tenancy scenario where there is competition among users for resources, the overhead shown in the HPC environment is enhanced.

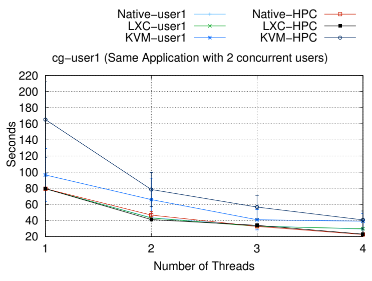
3.4.3.1.2 Different Applications

In this methodology, the multi-tenancy environment is represented by the execution of different applications of the NPB suite that are splitted into two users as shown in Figure 3.48.

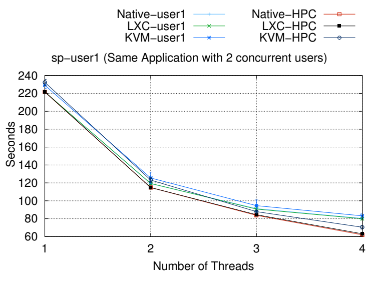
The FT (Figure 3.50(b)), EP (Figure 3.49(b)) and MG (Figure 3.49(e)) applica-



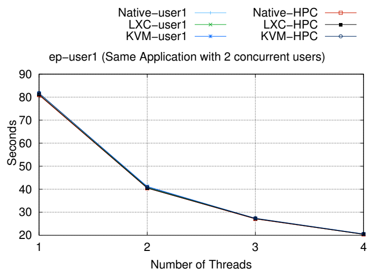
(a)



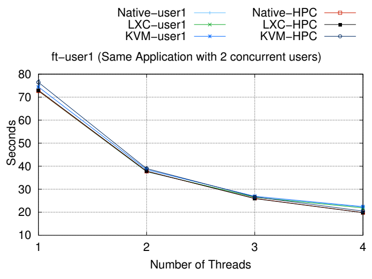
(b)



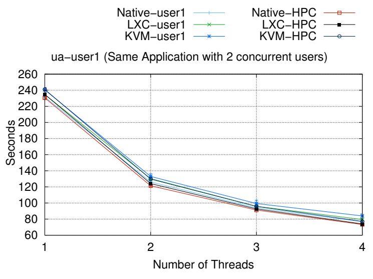
(c)



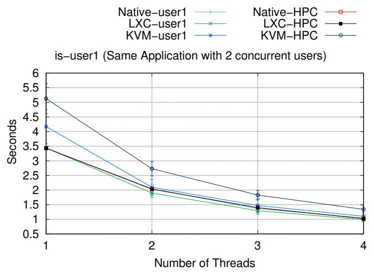
(d)



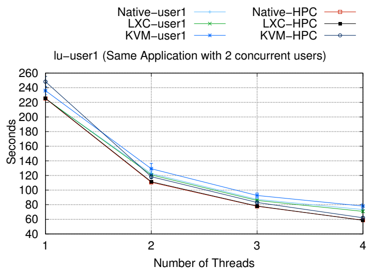
(e)



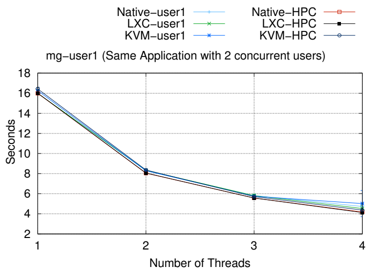
(f)



(g)



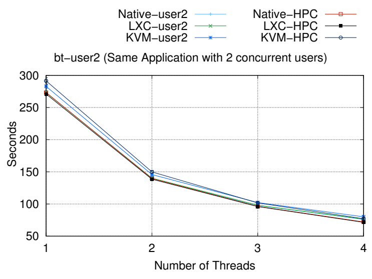
(h)



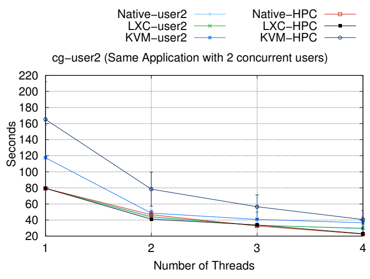
(i)

Source: Baum, Maliszewski, Griebler, 2017.

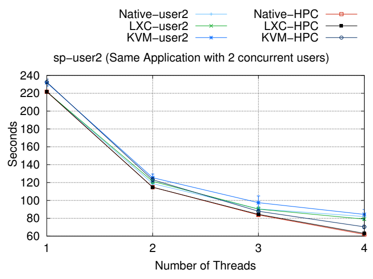
Figure 3.46: Multi-tenancy-User1 Scenario (Same Application with 2 concurrent users).



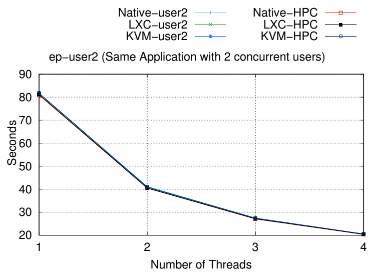
(a)



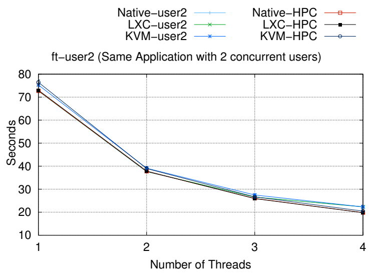
(b)



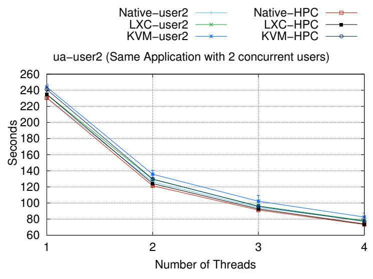
(c)



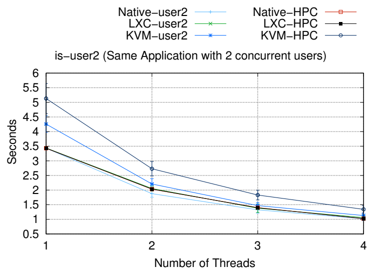
(d)



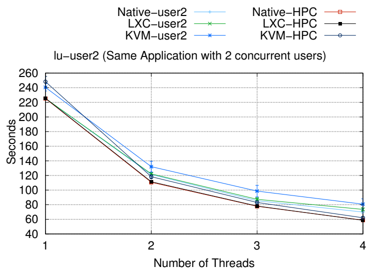
(e)



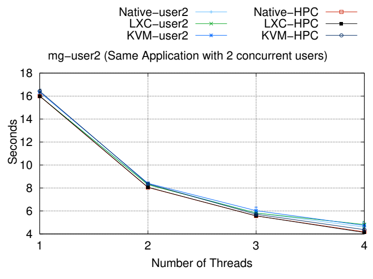
(f)



(g)



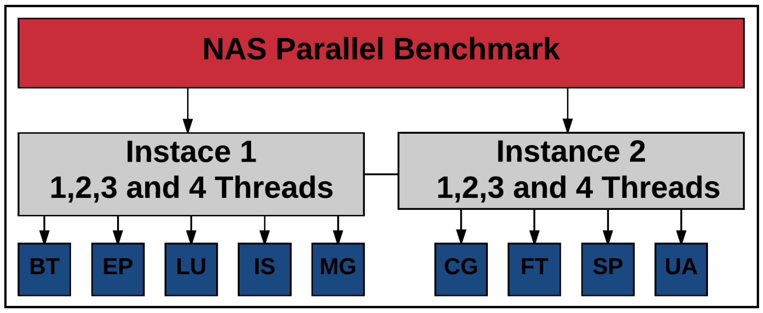
(h)



(i)

Source: Baum, Maliszewski, Griebler, 2017.

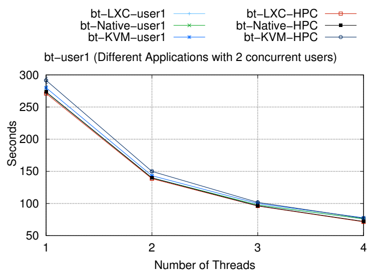
Figure 3.47: Multi-tenancy-User2 Scenario (Same Application with 2 concurrent users).



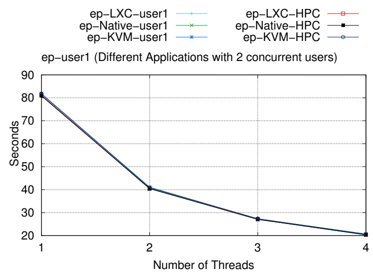
Source: Baum, Maliszewski, Griebler, 2017.

Figure 3.48: Description of NPB multi-tenancy environment with different applications on 3 concurrent users.

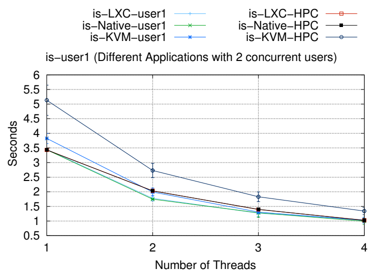
tions present very close results in the three environments (native, LXC and KVM based cloud). SP (Figure 3.50(c)), LU (Figure 3.49(d)) and BT (Figure 3.49(a)) demonstrate overhead in KVM and the LXC based cloud presents results very close to the native environment. This result is also seen in the HPC Scenario and suggests that these applications suffer overhead due to the addition of the virtualization layer. With the addition of this layer, more instructions need to be managed by the CPU, this involves more information to handle and as a result cause performance degradation Reddy and Rajamani (2014). A low performance in KVM-based cloud is presented in the CG (Figure 3.50(a)) and IS (Figure 3.49(c)), which also have already performed poorly on the KVM in the HPC scenario. CG suffers from high cache misses rates an context switching and IS is presented in the literature as an application with programming problems in virtualized environments (HUANG; LIU; ABALI; PANDA, 2006). Finally, UA (Figure 3.50(d)) presents overhead on both the LXC-based cloud and the KVM-based cloud, especially on the second thread. This result can be addressed by the performance losses presented in the processor utilization and by the cache miss rate that intensifies in the second thread.



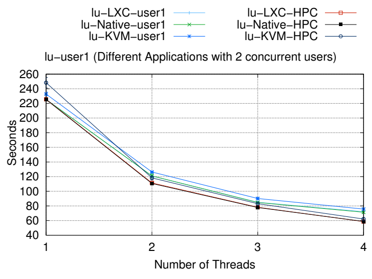
(a)



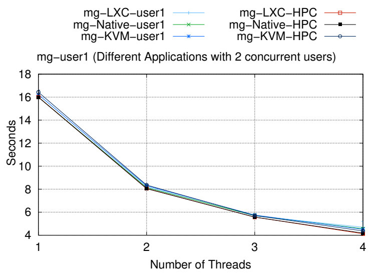
(b)



(c)



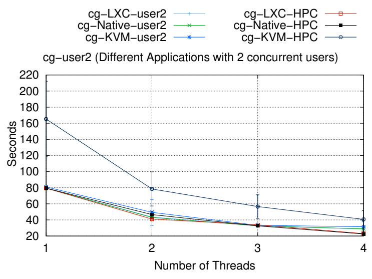
(d)



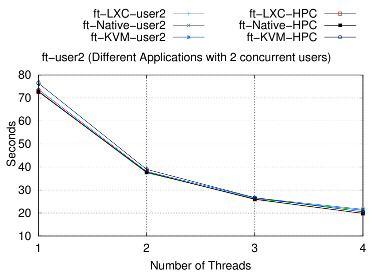
(e)

Source: Baum, Maliszewski, Griebler, 2017.

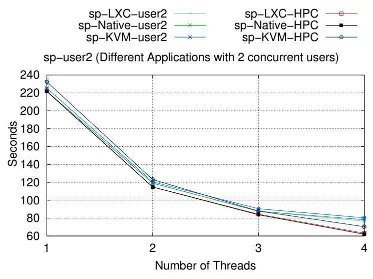
Figure 3.49: Multi-tenancy-User1 Scenario (Different Applications with 2 concurrent users).



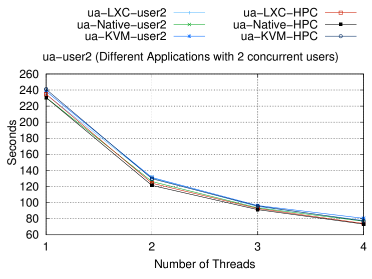
(a)



(b)



(c)



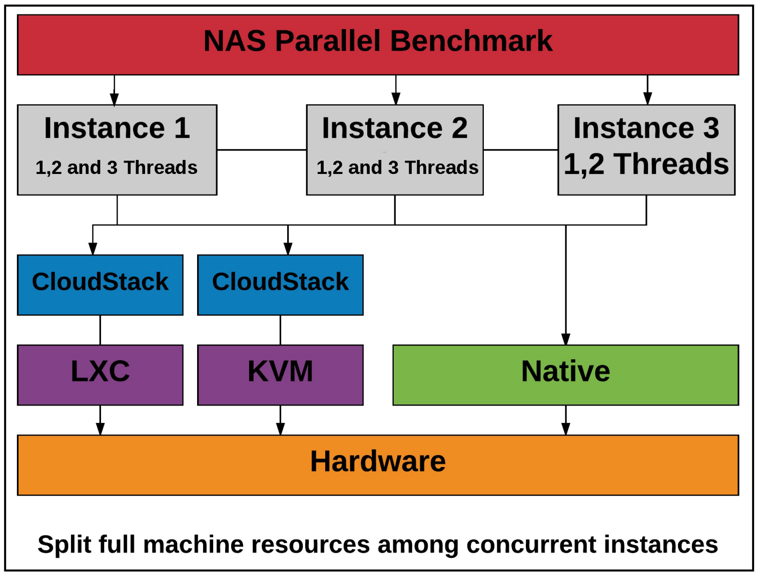
(d)

Source: Baum, Maliszewski, Griebler, 2017.

Figure 3.50: Multi-tenancy-User2 Scenario (Different Applications with 2 concurrent users).

3.4.3.2 Configuration 2

This configuration can be seen in Figure 3.51, which NAS Parallel Benchmark was executed in three concurrent instances. The first two instances had three threads allocated and the third instance had only two threads, totalizing the 8 threads available. In addition, in the cloud environment the service offer was splitted between the three instances.



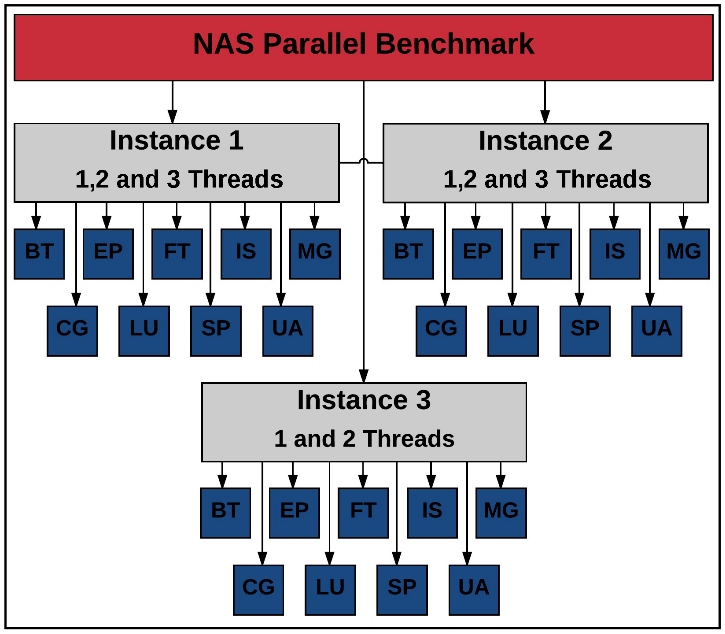
Source: Baum, Maliszewski, Griebler, 2017.

Figure 3.51: Conf2: Methodology followed in the evaluation of scientific applications in multi-tenancy scenario.

3.4.3.2.1 Same Applications

In this methodology (Figure 3.52), same applications were splitted over three concurrent instances in the LXC and KVM based clouds and in the Native environment over three users.

EP (Figure 3.53(d), 3.54(d), 3.55(d)), BT (Figure 3.53(a), 3.54(a), 3.55(a)), and UA (Figure 3.53(f), 3.54(f), 3.55(f)) workloads show few variations among virtualization technologies, with similar results already shown in previous tests.

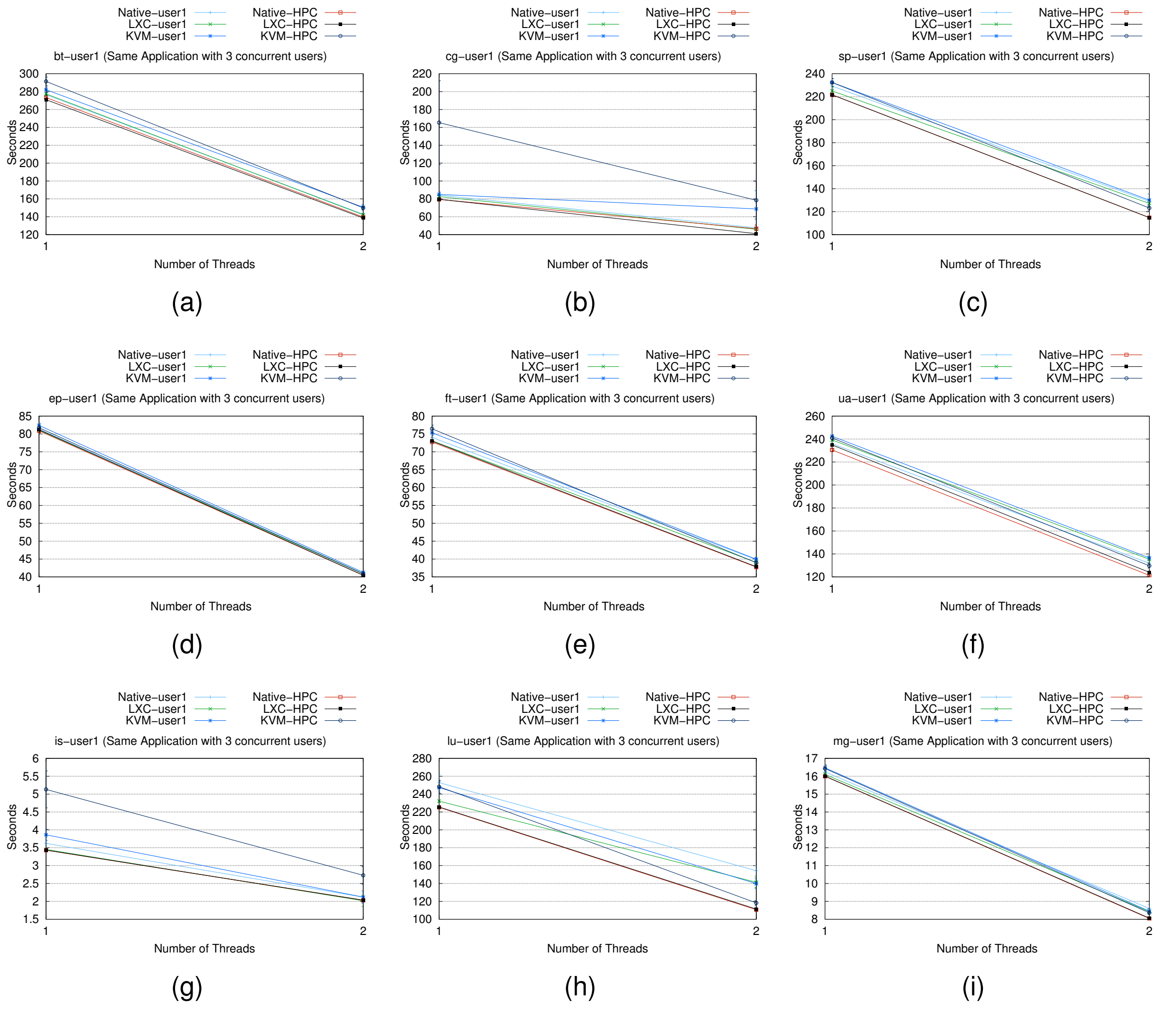


Source: Baum, Maliszewski, Griebler, 2017.

Figure 3.52: Description of NPB multi-tenancy environment with same applications on 3 concurrent users.

CG user3 (Figure 3.55(b)) shows a overhead in LXC. However, this result is minimized, taking into account the standard deviation that closely resembles the native environment. In addition, on user2 (Figure 3.54(b)) and 3 (Figure 3.55(b)) a overhead in KVM is noticed, we suspect that the cache-miss, context-switch, and virtualization layer addition factors are responsible for this result. FT (Figure 3.53(e), 3.54(e), 3.55(e)) and MG (Figure 3.53(i), 3.54(i), 3.55(i)) demonstrates a overhead in KVM and the LXC with better results than native environment. As MG requires large amounts of memory access Regola and Ducom (2010), the virtualization penalty is most significant. In addition, as explained before the native environment does not provide efficient resource isolation between users, therefore the users compete for resources, which leads to performance degradation.

IS (Figure 3.53(g), 3.54(g), 3.55(g)) presents a high overhead in KVM, a result that has already been seen in the environment without competition. The behavior of the native environment is what attracts attention, showing inferior results to the LXC

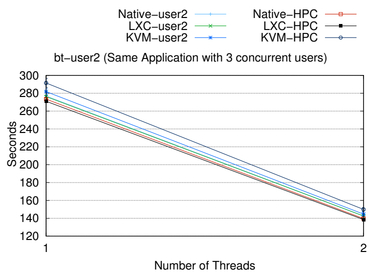


Source: Baum, Maliszewski, Griebler, 2017.

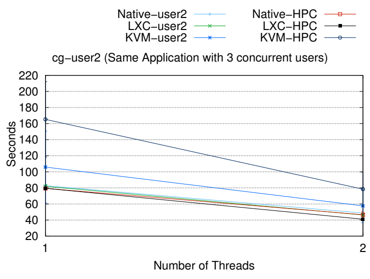
Figure 3.53: Multi-tenancy-User1 Scenario (Same Application with 3 concurrent users).

environment. This result is probably induced by the high competition of resources that occurs in the native environment.

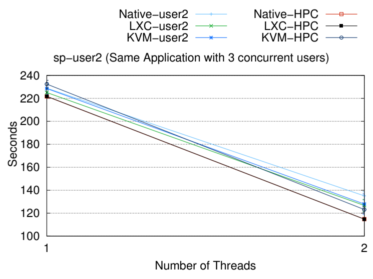
LU in turn has large variations in its results. The native environment performs poorly on the first (Figure 3.53(h)) and second users (Figure 3.54(h)). However in the third user (Figure 3.55(h)) the native environment has a performance lower than LXC and higher than KVM. We believe that this large variation of results comes from the way this application uses the processor and also the high cache miss rate presented in the first two threads.



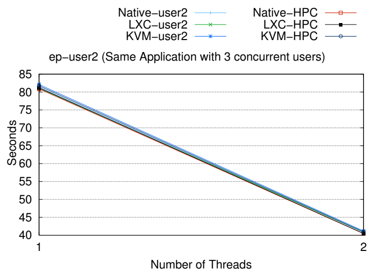
(a)



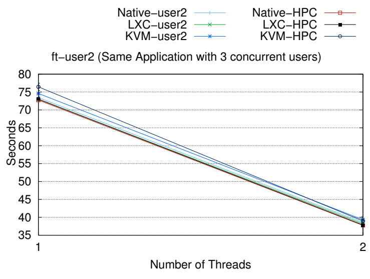
(b)



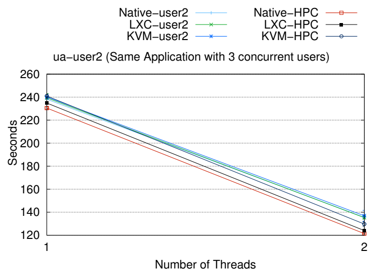
(c)



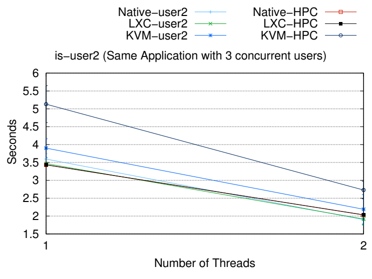
(d)



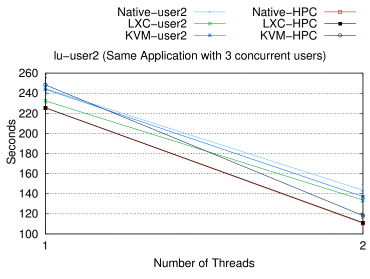
(e)



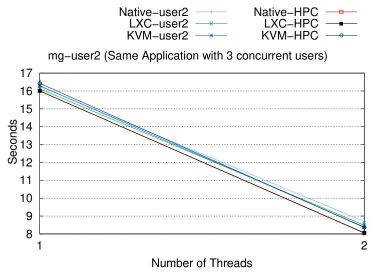
(f)



(g)



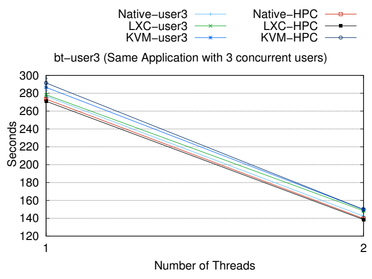
(h)



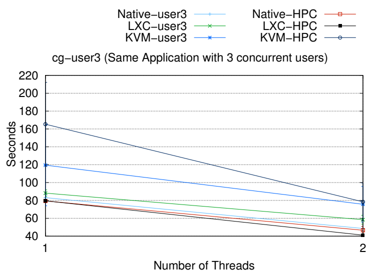
(i)

Source: Baum, Maliszewski, Griebler, 2017.

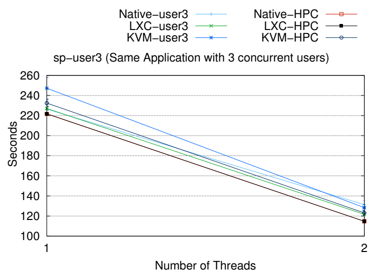
Figure 3.54: Multi-tenancy-User2 Scenario (Same Application with 3 concurrent users).



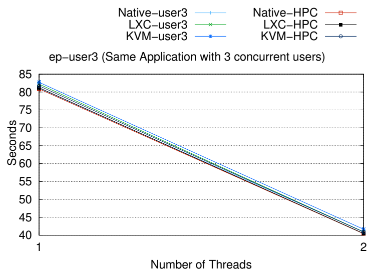
(a)



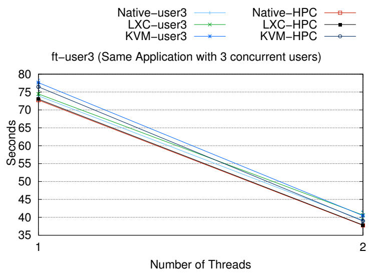
(b)



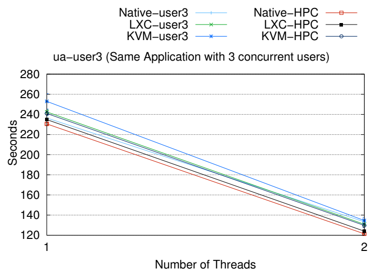
(c)



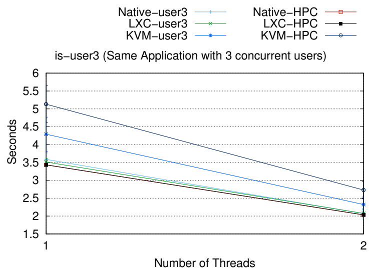
(d)



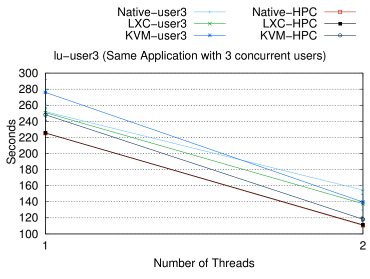
(e)



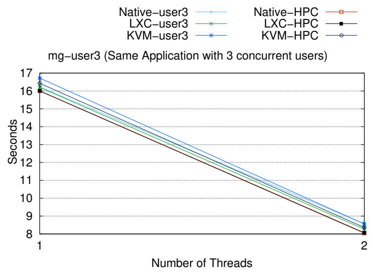
(f)



(g)



(h)



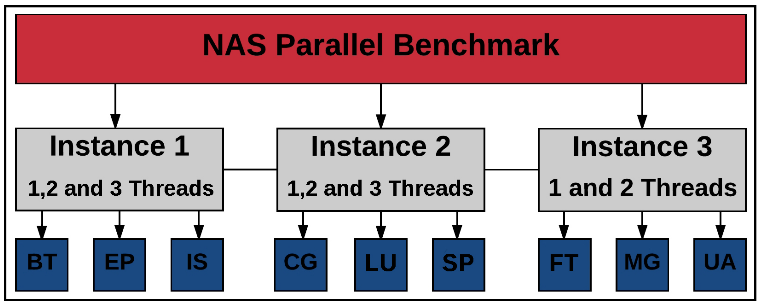
(i)

Source: Baum, Maliszewski, Griebler, 2017.

Figure 3.55: Multi-tenancy-User3 Scenario (Same Application with 3 concurrent users).

3.4.3.2.2 Different Applications

In this methodology, the multi-tenancy environment is represented by the execution of different applications of the NPB suite that are splitted into two users as shown in Figure 3.56.

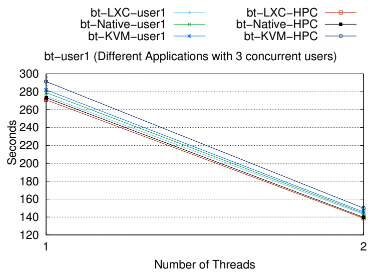


Source: Baum, Maliszewski, Griebler, 2017.

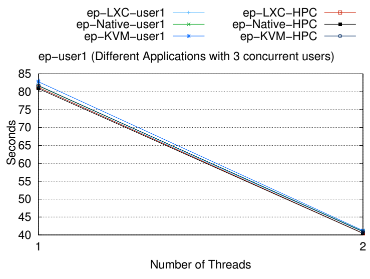
Figure 3.56: Description of NPB multi-tenancy scenario with different applications on 3 concurrent users.

BT (Figure 3.57(a)) and EP (Figure 3.57(b)) benchmarks show results very close to the native environment. The CG (Figure 3.58(a)) application shows KVM overhead, a result already shown in other tests, which we believe happens due to high rates of context switching. FT (Figure 3.59(a)), LU (Figure 3.58(b)), MG (Figure 3.59(b)) and SP (Figure 3.58(c)) demonstrated low overhead in KVM, which can be directed to the addition of the virtualization layer.

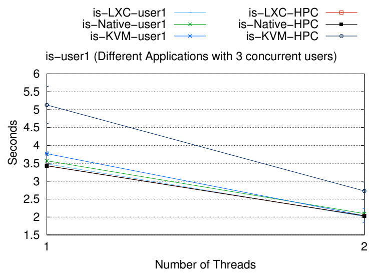
IS (Figure 3.57(c)) showed low overhead KVM and the LXC-based cloud performs better than native. This behavior is probably the result of high competition for resources in the native environment. Finally LU 3.58(b) presents overhead in LXC and KVM. This particular application has shown several behaviors in the concurrency environment, which we believe to be due to the way it uses the resources coming from the processor, which presents many performance losses.



(a)



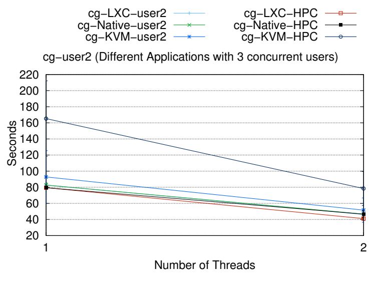
(b)



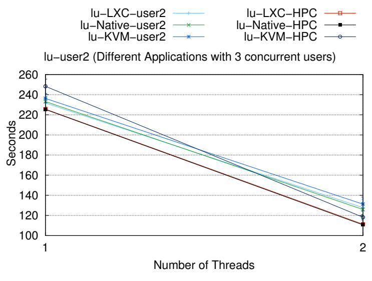
(c)

Source: Baum, Maliszewski, Griebler, 2017.

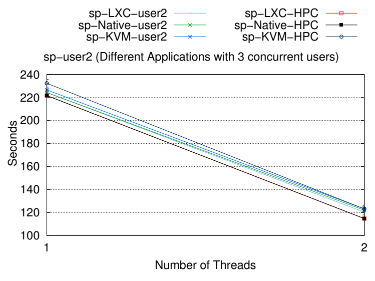
Figure 3.57: Multi-tenancy-User1 Scenario (Different Applications with 3 concurrent users).



(a)



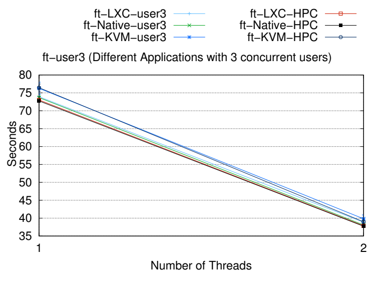
(b)



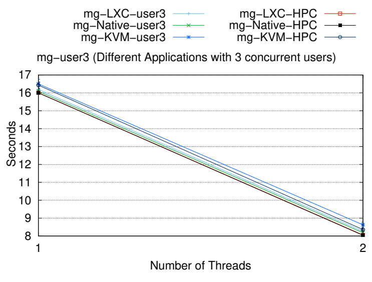
(c)

Source: Baum, Maliszewski, Griebler, 2017.

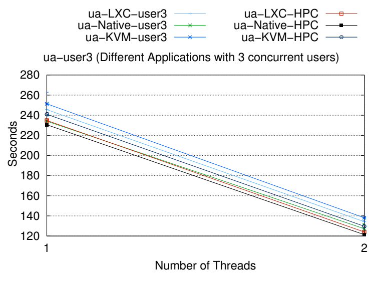
Figure 3.58: Multi-tenancy-User2 Scenario (Different Applications with 3 concurrent users).



(a)



(b)



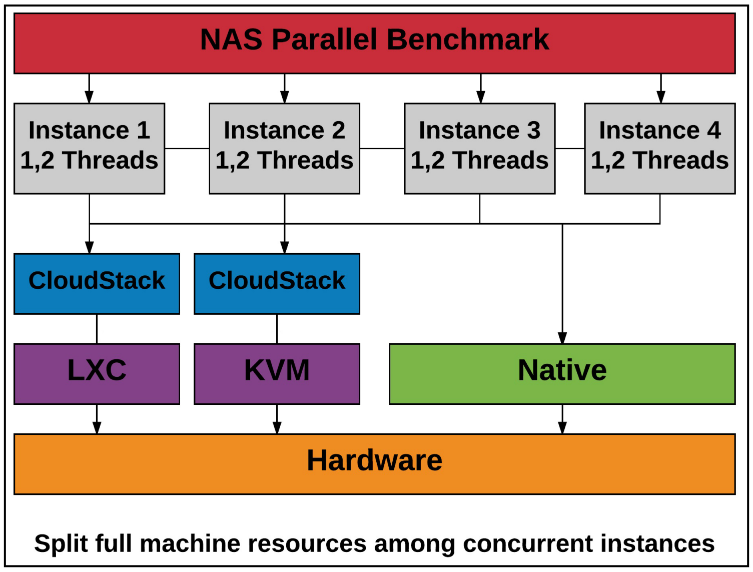
(c)

Source: Baum, Maliszewski, Griebler, 2017.

Figure 3.59: Multi-tenancy-User3 Scenario (Different Applications with 3 concurrent users).

3.4.3.3 Configuration 3

This is the last configuration and can be seen in Figure 3.60, which NAS Parallel Benchmark was executed in four concurrent instances. Each instance allocated 2 threads, totalizing the 8 threads available. In addition, in the cloud environment the service offer was splitted between the four instances.



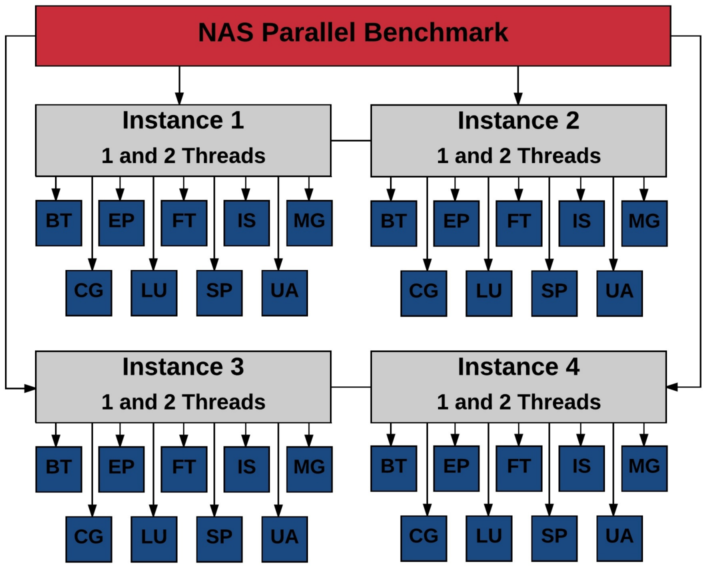
Source: Baum, Maliszewski, Griebler, 2017.

Figure 3.60: Conf3: Methodology followed in the evaluation of scientific applications in multi-tenancy scenario.

3.4.3.3.1 Same Applications

In this methodology (Figure 3.61), same applications were splitted over four concurrent instances in the LXC and KVM based clouds and in the Native environment over four users.

EP (Figures 3.62(d), 3.63(d), 3.64(d), 3.65(d)) workload shows its execution time with similar results in the three environments, suggesting that this application can be used in a competitive environment without significant losses.



Source: Baum, Maliszewski, Griebler, 2017.

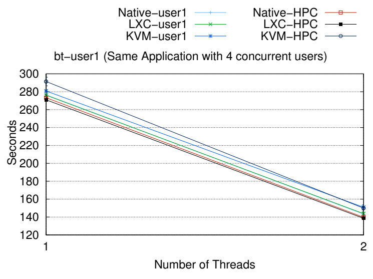
Figure 3.61: Description of NPB multi-tenancy environment with same applications on 4 concurrent users.

IS (Figure 3.62(g), 3.63(g), 3.64(g), 3.65(g)), FT (Figure 3.62(e), 3.63(e), 3.64(e), 3.65(e)), CG (Figure 3.62(b), 3.63(b), 3.64(b), 3.65(b)), LU (Figure 3.62(h), 3.63(h), 3.64(h), 3.65(h)), MG (Figure 3.62(i), 3.63(i), 3.64(i), 3.65(i)), and SP (Figure 3.62(c), 3.63(c), 3.64(c), 3.65(c)) workloads demonstrate performance losses in the native environment, suggesting that with competition for resources, applications tend to work better with virtualization than in the native environment, mainly due to the isolation that is one of the main characteristics of the virtual environment.

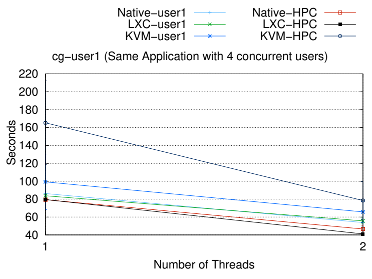
Although the UA (Figure 3.62(f), 3.63(f), 3.64(f), 3.65(f)) application presents differences between the environments, if we take into account the standard deviation of the execution time, we realize that the results are very similar.

3.4.3.3.2 Different Applications

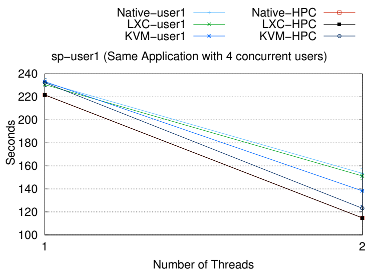
In this methodology, the multi-tenancy environment is represented by the execution of different applications of the NPB suite that are splitted into four users as



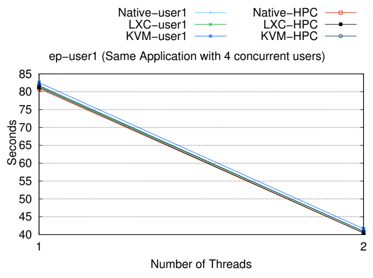
(a)



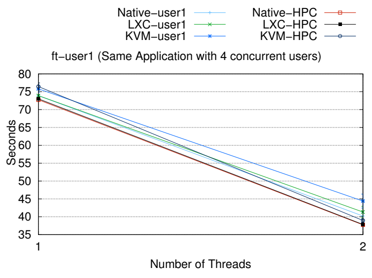
(b)



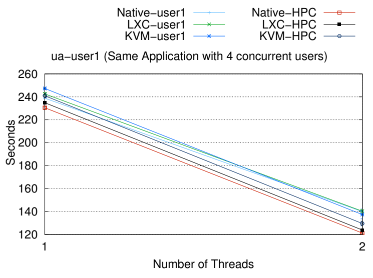
(c)



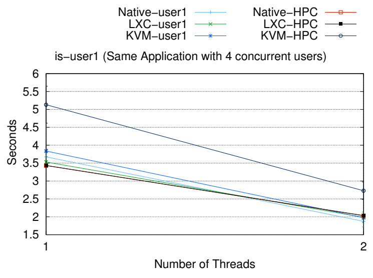
(d)



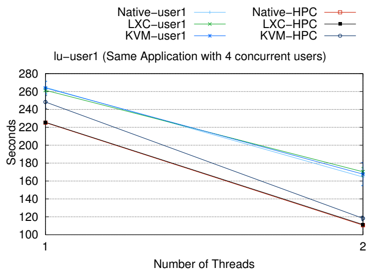
(e)



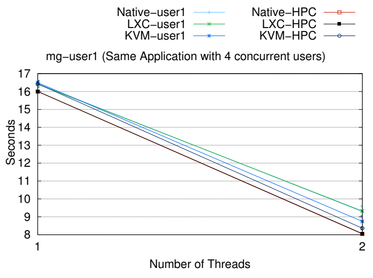
(f)



(g)



(h)



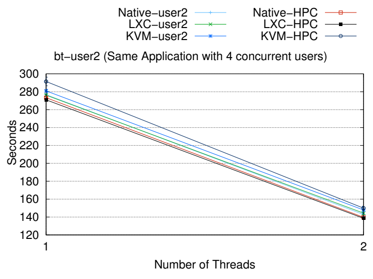
(i)

Source: Baum, Maliszewski, Griebler, 2017.

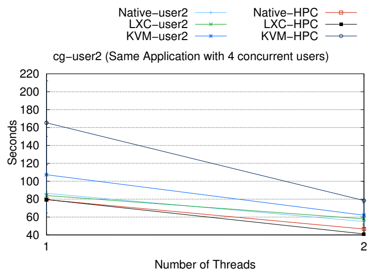
Figure 3.62: Multi-tenancy-User1 Scenario (Same Application with 4 concurrent users).

shown in Figure 3.66.

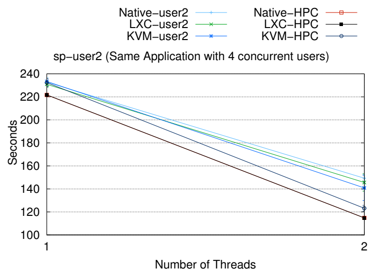
EP (Figure 3.67(a) and FT 3.70(a) show very close execution times in all environments, which demonstrates that these applications in this case can be used effectively. The applications BT (Figure 3.68(a)), CG (Figure 3.69(a)), IS (Figure 3.70(b)), MG (Figure 3.68(b)) and UA (Figure 3.67(b)) demonstrate performance degradation based on the KVM based cloud. This behavior can be directed to the virtualization layer that was added as well as concurrency presented in this methodology. However, in order to confirm this justification a thorough study needs to be done. In contrast, LU



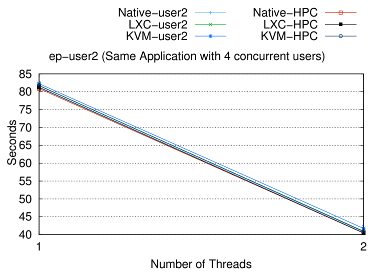
(a)



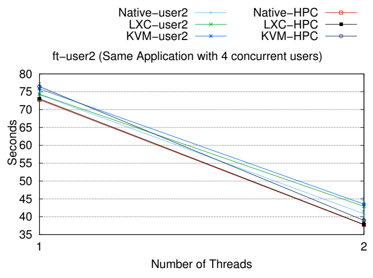
(b)



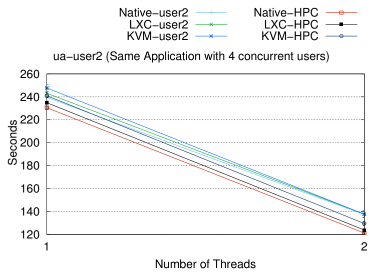
(c)



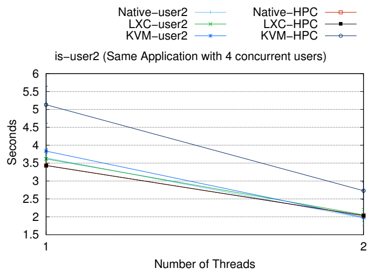
(d)



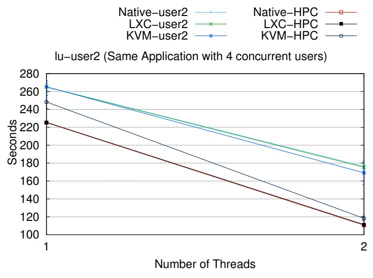
(e)



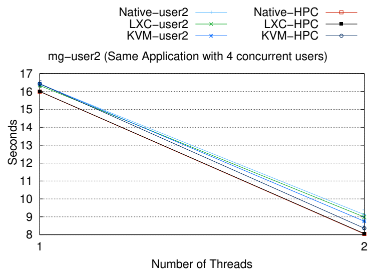
(f)



(g)



(h)

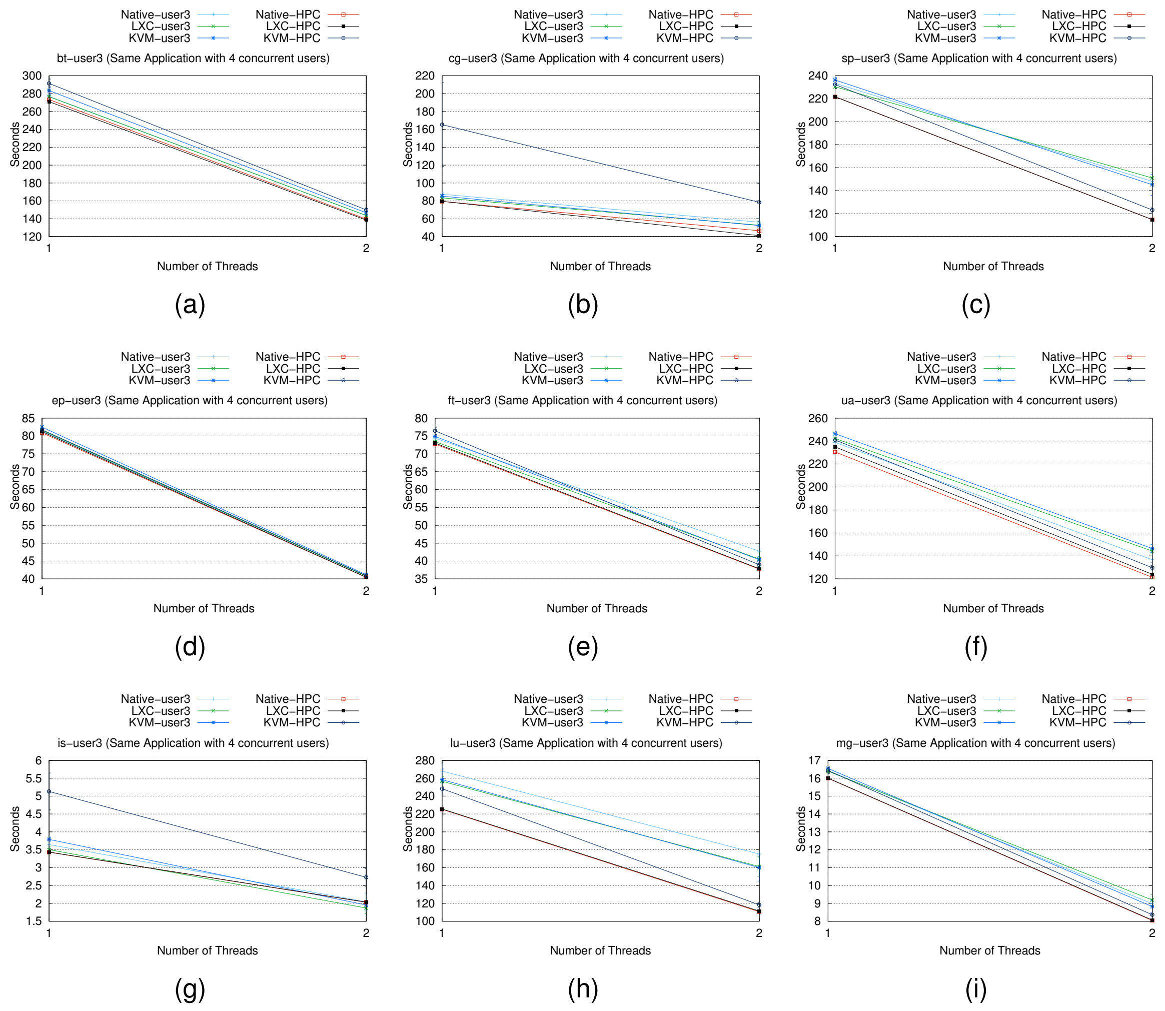


(i)

Source: Baum, Maliszewski, Griebler, 2017.

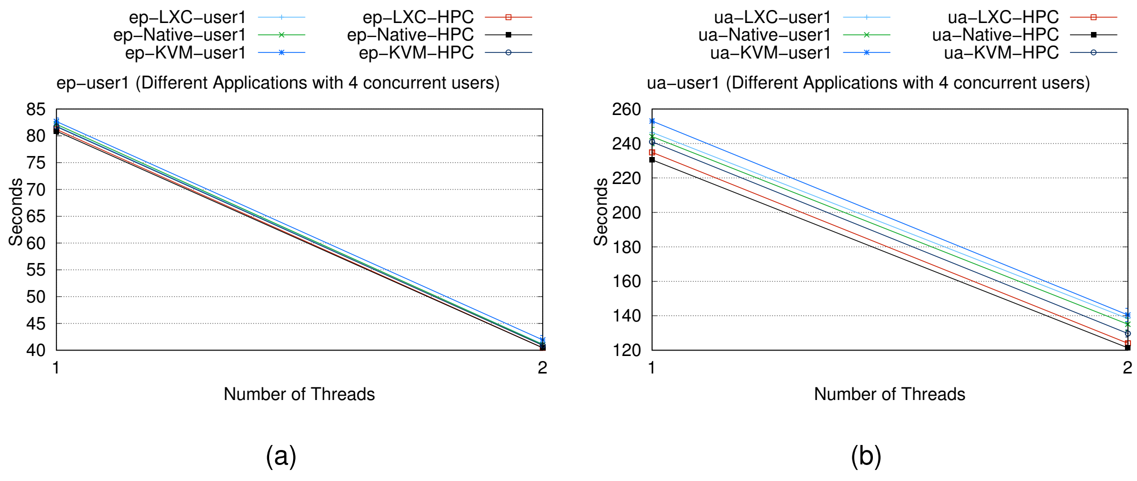
Figure 3.63: Multi-tenancy-User2 Scenario (Same Application with 4 concurrent users).

applications (Figure 3.70(c)) and SP (Figure 3.69(b)) demonstrate better performance in the KVM-based cloud, emphasizing that in a competitive environment, the isolation presents advantages for complete virtualization.



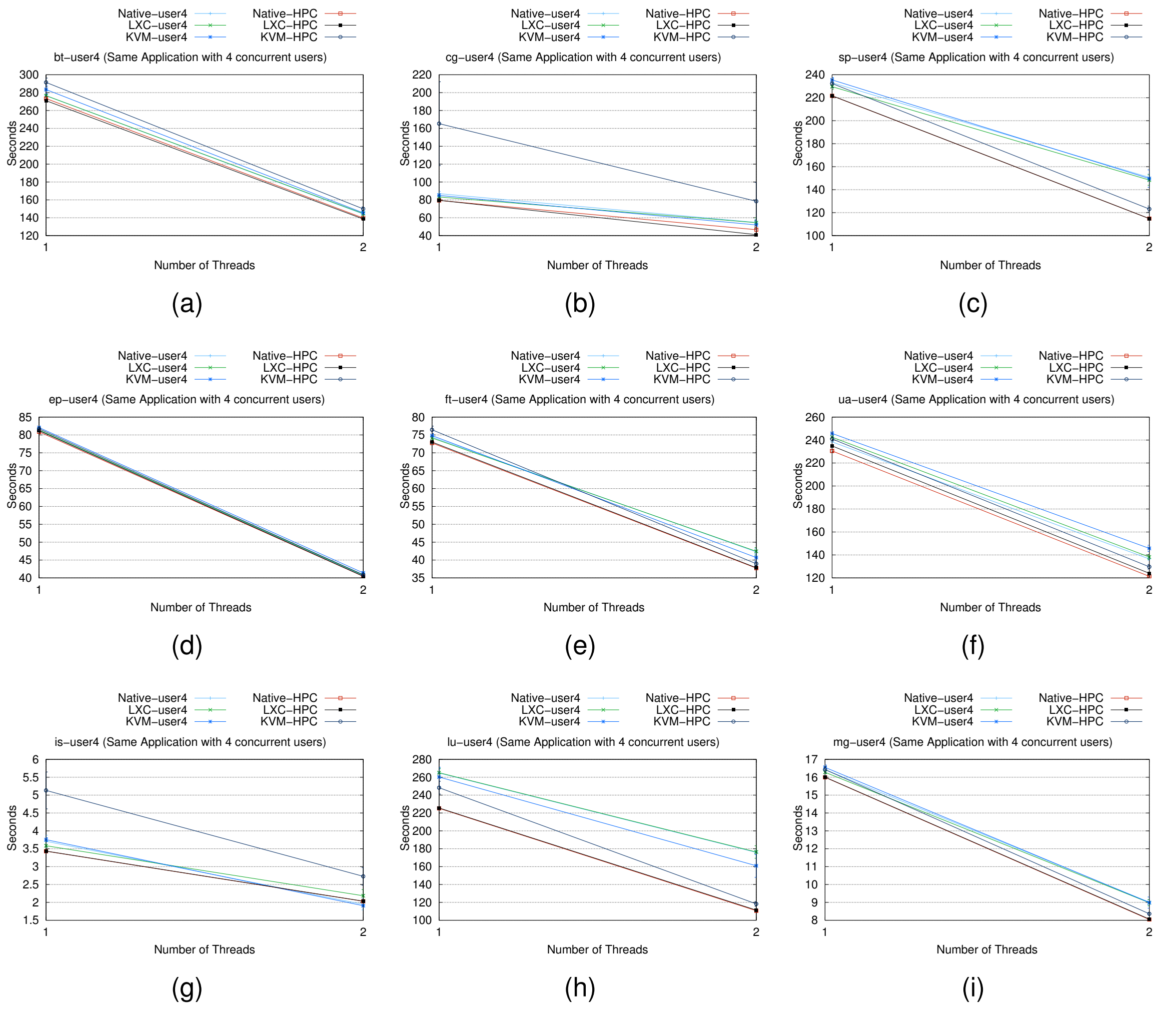
Source: Baum, Maliszewski, Griebler, 2017.

Figure 3.64: Multi-tenancy-User3 Scenario (Same Application with 4 concurrent users).



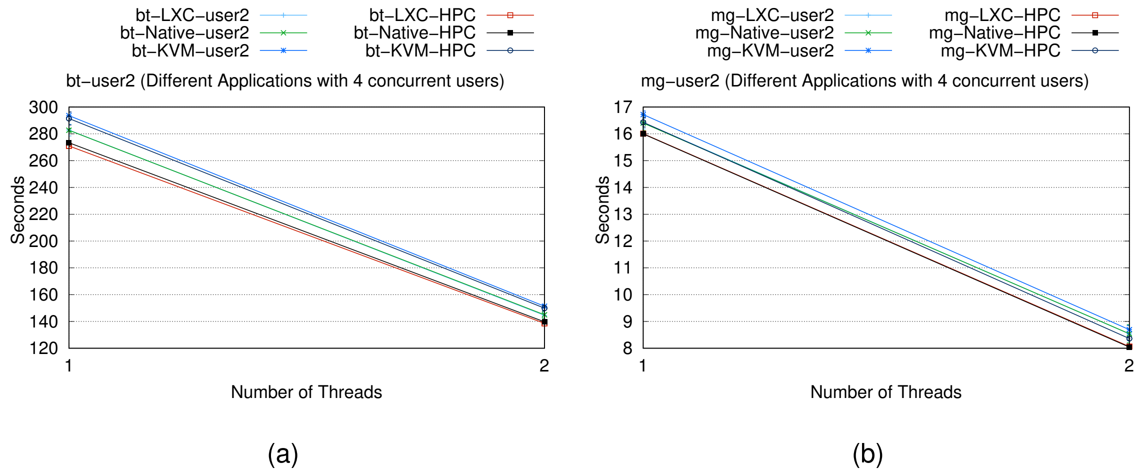
Source: Baum, Maliszewski, Griebler, 2017.

Figure 3.67: Multi-tenancy-User1 Scenario (Different Applications with 4 concurrent users).



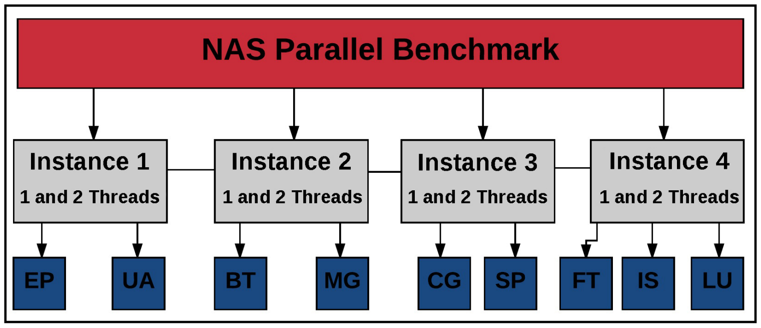
Source: Baum, Maliszewski, Griebler, 2017.

Figure 3.65: Multi-tenancy-User4 Scenario (Same Application with 4 concurrent users).



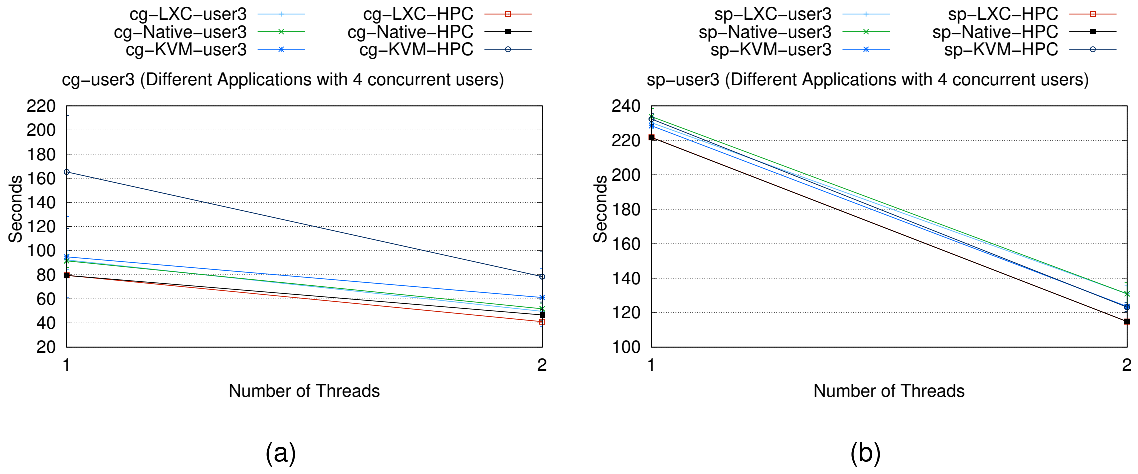
Source: Baum, Maliszewski, Griebler, 2017.

Figure 3.68: Multi-tenancy-User2 Scenario (Different Applications with 4 concurrent users).



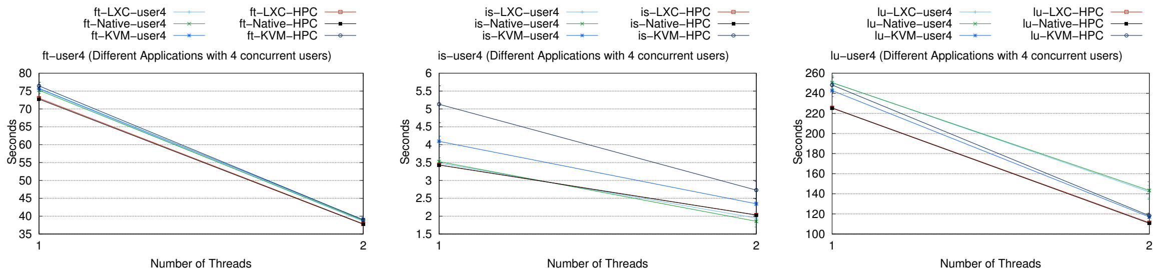
Source: Baum, Maliszewski, Griebler, 2017.

Figure 3.66: Description of NPB multi-tenancy environment with different applications on 4 concurrent users.



Source: Baum, Maliszewski, Griebler, 2017.

Figure 3.69: Multi-tenancy-User3 Scenario (Different Applications with 4 concurrent users).



(a) (b) (c)

Source: Baum, Maliszewski, Griebler, 2017.

Figure 3.70: Multi-tenancy-User4 Scenario (Different Applications with 4 concurrent users).

3.5 HYPOTHESIS TESTS

In the previous sections all the experiments were represented in graphs, and we could perceive that some differences between the native, LXC and KVM based cloud regard to its execution time. To show whether this differences are significant it must be applied a statistical process. Therefore, in this section it is presented the hypothesis and also described the statistical procedure used.

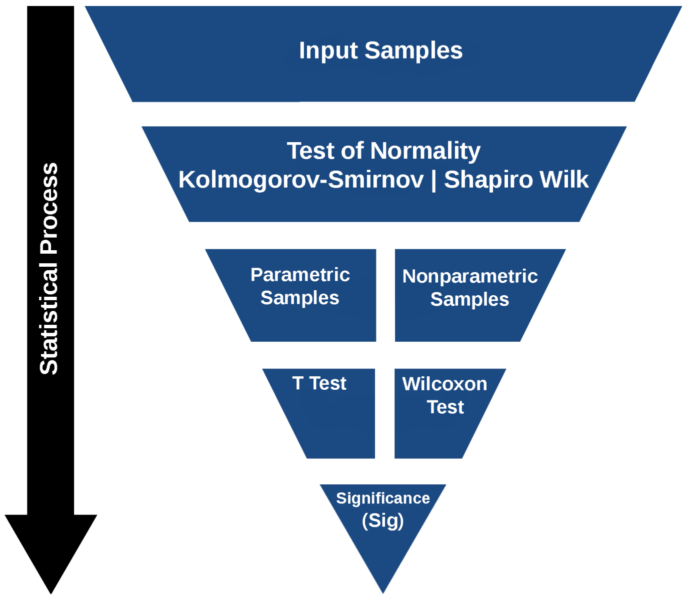
With the results obtained from the experiments, more specifically from the applications, it is possible to extract informations about its behavior using the statistical method. To do this, it was used the IBM SPSS[14] which is frequently used in experiments and statistical analysis to the hypothesis tests, avoiding calculation with complex formulas.

The margin of 95% of confiablity was used in the statistical analisys with SPSS. It means that in the comparisons done by the program, if the significance value is greater than 0,05 (5%), it is not possible to state that the samples are significantly different.

[14]https://www.ibm.com/us-en/marketplace/spss-statistics

The entire statistical process described below can be seen in Figure 3.71. This process first performed the sample collection. Subsequently, the normality test was done. This test is performed to determine if the sample distribution is on a normal curve, determining whether these are parametric or non-parametric. Normality exists when the Kolmogorov-Smirnov and Shapiro-Wilk approaches have a greater significance than 0,05 (Sig > 0,05), thus making the sample parametric. Considering that the samples in this thesis are small (10), the author Field (2013) emphasizes that the Sharpiro-Wilk approach should be used because it is specific for small samples. Instead, the Kolgomorov-Smirnov approach is specific for samples greater than 30. Then, the significance (Sig.) of the Shapiro-Wilk approach was used.

The next step was to perform the parametric or non-parametric test in each sample previously defined by the normality test. In the parametric samples the T Test was applied, which paired the samples and compared them to obtain the result of significance. On the other hand, in the non-parametric samples the Wilcoxon test was applied, which ranks the samples and also compares them to obtain the significance. The significance of these tests (T test and Wilcoxon test) will indicate whether the comparison is significantly different or not using the 95% confidence margin. That is, when the level of significance reaches the margin greater than 0.05 (Sig < 0,05), it is not possible to affirm that there are significant differences between the samples



Source: Baum, Maliszewski, Griebler, 2017.

Figure 3.71: Statistical Process.

As many statistical tests were done, the results in the item Sig. of each table show only those with significant or non-significant difference. In order to make this choice, the results were previously analyzed taking as a rule the addition of those that had the lowest number of occurrences, that is, if the test has a higher number of samples with significantly different results, only the non-significant ones are arranged in the tables and it is assumed that most of the results indicates that there is a significant differences.

3.5.0.1 Formal Configuration of the Experiments

The execution time metric (Et) provided by the benchmarks refers to differences between the final execution time (FEt) and the initial execution time (IEt), represented in the Equation 3.1.

$$Et = FEt - IEt \tag{3.1}$$

3.5.1 First Hypothesis

The performance characteristics of the scientific and enterprise workloads are statistically different among deployed cloud scenarios.

The descriptive hypothesis was formalized by dividing NAS and PARSEC suites in order to verify the significance among the samples of the deployed cloud scenarios (LXC and KVM). In addition, the variables used in the statistical hypotheses are: $EtKN$ for execution time in KVM cloud-based in NPB, $EtLN$ for execution time in LXC cloud-based in NPB, $EtKP$ for execution time in KVM cloud-based in PARSEC and $EtLP$ for execution time in LXC cloud-based in PARSEC.

In relation to the execution time of the NPB-OMP suite the hypotheses are the follows:

- *H0:* $EtKN$ == $EtLN$
- *H1:* $EtKN$!= $EtLN$

Therefore, through the analysis of the results obtained in the Sig, the great majority indicates that there are significant differences between the KVM and LXC cloud-based in relation to the scientific applications (NPB-OMP suite), so the null hypothesis (H0) is rejected, and thus assumes the alternative hypothesis (H1).

In relation to the execution time of the PARSEC suite the hypotheses are the follows:

- *H0:* $EtKP$ == $EtLP$
- *H1:* $EtKP$!= $EtLP$

Therefore, through the statistical analysis, we prove the rejection of the null hypothesis (H0), because most of the results attests significant differences between the KVM and LXC cloud-based in relation to the enterprise applications (PARSEC suite), and thus assumes the alternative hypothesis (H1).

The Table 3.3 is divided between the results of the NPB-OMP (KVM x LXC) and PARSEC (KVM x LXC) suites, with the green column being the partition. The results are arranged according to the scenario (with gray background), in this way in the first column of each suite arranged in the table are represented the applications, number of threads and in the multi-tenancy scenarios the corresponding user. In the second column of each suite is represented the Sig. variable respecting the rule of addition of the results with less occurrence. In both cases (NPB-OMP and PARSEC) the lowest number of occurrences was Sig. > 0.05, or statistically non-different.

NPB-OMP SUITE (LXC X KVM)		🟩	**PARSEC SUITE (LXC X KVM)**	
High Performance		🟩	**High Performance**	
Application-Threads	**Sig.**	🟩	**Application-Threads**	**Sig.**
EP-1	0,202	🟩	STREAMCLUSTER-6	0,057
EP-6	0,720	🟩	STREAMCLUSTER-8	0,288
FT-7	0,602	🟩	CANNEAL-8	0,849
Multi-Tenancy Same 2 Users		🟩	DEDUP-8	0,902
User-Application-Threads	**Sig.**	🟩	FACESIM-2	0,059
USER1-EP-3	0,308	🟩	FACESIM-5	0,071
USER1-EP-4	0,053	🟩	X264-5	0,139
USER1-FT-4	0,093	🟩	CANNEAL-7	0,203
USER1-IS-2	0,153	🟩	VIPS-1	0,203
Continued on next page				

Table 3.3 – Continued from previous page

USER1-IS-3	0,103	CANNEAL-6	0,285
USER1-IS-4	0,059	CANNEAL-5	0,508
USER1-MG-4	0,221	X264-1	0,878
USER1-SP-3	0,059	**Multi-Tenancy Same 2 Users**	
USER1-UA-2	0,074	**User-Application-Threads**	**Sig.**
USER2-EP-1	0,052	USER2-CANNEAL-1	0,198
USER2-FT-4	0,575	USER1-STREAMCLUSTER-2	0,230
USER2-MG-4	0,475	USER1-X264-3	0,305
USER1-MG-2	0,129	USER2-X264-3	0,366
USER1-MG-3	0,263	USER1-X264-4	0,404
USER2-EP-4	0,150	USER1-SWAPTIONS-4	0,413
USER2-IS-2	0,077	USER2-BLACKSCHOLES-1	0,504
USER2-IS3	0,235	USER1-FACESIM-4	0,520
USER2-IS-4	0,061	USER1-FACESIM-1	0,840
Multi-Tenancy Same 3 Users		USER2-FACESIM-1	0,940
User-Application-Threads	**Sig.**	USER2-STREAMCLUSTER-4	0,203
USER1-IS-2	0,169	USER1-SWAPTIONS-4	0,241
USER2-IS-2	0,093	USER1-STREAMCLUSTER-4	0,285
USER2-LU-1	0,114	USER2-X264-1	0,386
USER2-MG-1	0,074	USER1-X264-1	0,575
USER2-MG-2	0,262	USER2-STREAMCLUSTER-3	0,799
USER2-SP-1	0,308	USER1-STREAMCLUSTER-3	0,959
USER2-SP-2	0,878	USER2-STREAMCLUSTER-2	0,959
USER2-UA-2	0,799	**Multi-Tenancy Same 3 Users**	
USER3-CG-2	0,169	**User-Application-Threads**	**Sig.**
USER3-EP-1	0,059	USER1-BLACKSCHOLES-1	0,050
USER1-LU-2	0,608	USER2-FACESIM-2	0,061

Continued on next page

Table 3.3 – Continued from previous page

USER1-MG-2	0,699	USER2-RAYTRACE-1	0,078
USER1-SP-2	0,158	USER3-STREAMCLUSTER-2	0,129
USER1-UA-2	0,076	USER2-BLACKSCHOLES-2	0,143
USER2-LU-2	0,068	USER2-CANNEAL-2	0,175
USER2-UA-1	0,625	USER1-BLACKSCHOLES-2	0,239
USER3-BT-2	0,158	USER1-STREAMCLUSTER-2	0,263
USER3-FT-2	0,641	USER3-BLACKSCHOLES-2	0,532
USER3-LU-2	0,592	USER2-FACESIM-1	0,835
Multi-Tenancy Same 4 Users		USER1-X264-1	0,093
User-Application-Threads	**Sig.**	USER2-FREQMINE-2	0,114
USER1-CG-2	0,059	USER3-BODYTRACK-2	0,139
USER1-IS-2	0,838	USER1-VIPS-1	0,169
USER1-LU-1	0,203	USER2-STREAMCLUSTER-2	0,285
USER1-UA-2	0,074	USER3-FREQMINE-2	0,333
USER2-CG-1	0,093	USER2-X264-2	0,445
USER2-IS-1	0,059	USER2-BLACKSCHOLES-1	0,508
USER2-LU-1	0,959	USER2-X264-1	0,721
USER2-LU-1	0,959	USER3-VIPS-1	0,799
USER2-LU-2	0,114	**Multi-Tenancy Same 4 Users**	
USER2-MG-2	0,059	**User-Application-Threads**	**Sig.**
USER2-UA-2	0,959	USER2-BODYTRACK-2	0,059
USER3-CG-2	0,721	USER3-BLACKSCHOLES-2	0,095
USER3-FT-2	0,959	USER3-CANNEAL-1	0,126
USER3-IS-2	0,074	USER3-X264-1	0,274
USER3-LU-2	0,721	USER1-STREAMCLUSTER-2	0,322
USER4-CG-2	0,285	USER1-BODYTRACK-2	0,345
USER4-CG-2	0,285	USER2-X264-1	0,346

Continued on next page

Table 3.3 – Continued from previous page

USER4-SP-2	0,333	USER3-FACESIM-2	0,403
USER1-LU2	0,456	USER4-BODYTRACK-2	0,683
USER1-MG-1	0,217	USER3-STREAMCLUSTER-1	0,687
USER2-BT-2	0,059	USER2-STREAMCLUSTER-1	0,768
USER2-FT-2	0,464	USER1-FACESIM-2	0,768
USER2-IS-2	0,276	USER4-BLACKSCHOLES-2	0,833
USER3-LU-1	0,572	USER2-BLACKSCHOLES-1	0,884
USER3-SP-2	0,127	USER3-BLACKSCHOLES-1	0,947
USER4-BT-2	0,431	USER1-BLACKSCHOLES-2	0,575
USER4-LU-1	0,237	USER1-X264-1	0,059
Multi-TenancyDifferent 2 Users		USER4-STREAMCLUSTER-2	0,074
User-Application-Threads	**Sig.**	USER3-SWAPTIONS-2	0,093
USER1-IS-3	0,507	USER4-FACESIM-2	0,114
USER1-IS-4	0,513	USER1-VIPS-2	0,139
USER1-EP-1	0,053	USER4-BLACKSCHOLES-1	0,445
USER1-MG-4	0,553	USER4-STREAMCLUSTER-1	0,508
USER2-CG-3	0,203	USER1-VIPS-1	0,799
USER2-CG-2	0,331	USER3-STREAMCLUSTER-2	0,878
USER2-UA-2	0,513	USER3-BODYTRACK-2	0,959
Multi-TenancyDifferent 3 Users		**Multi-Tenancy Diff 2 Users**	
User-Application-Threads	**Sig.**	**User-Application-Threads**	**Sig.**
USER1-IS2	0,760	USER1-BLACKSCHOLES-1	0,353
USER2-CG-1	0,646	USER1-BLACKSCHOLES-2	0,071
USER2-CG-2	0,575	USER1-FREQMINE-4	0,276
Multi-TenancyDifferent 4 Users		USER1-FREQMINE-3	0,513
User-Application-Threads	**Sig.**	USER2-CANNEAL-3	0,826
USER1-UA-2	0,241	USER2-STREAMCLUSTER-1	0,074

Continued on next page

Table 3.3 – Continued from previous page

USER2-MG-2	0,139	USER2-SWAPTIONS-4	0,093
USER3-CG-1	0,093	USER1-BODYTRACK-4	0,139
USER3-CG-2	0,959	USER1-FREQMINE-1	0,139
USER4-FT-2	0,086	USER1-X264-4	0,386
USER3-SP-1	0,124	USER2-CANNEAL-4	0,445
USER4-FT-1	0,767	USER2-CANNEAL-2	0,646
		USER2-CANNEAL-1	0,721
		USER2-STREAMCLUSTER-3	0,799
		USER2-STREAMCLUSTER-2	0,959
		USER2-STREAMCLUSTER-4	0,959
		Multi-Tenancy Diff 3 Users	
		User-Application-Threads	**Sig.**
		USER2-X264-2	0,101
		USER3-STREAMCLUSTER-2	0,093
		USER1-CANNEAL-1	0,169
		USER3-STREAMCLUSTER-1	0,959
		Multi-Tenancy Diff 4 Users	
		User-Application-Threads	**Sig.**
		USER1-BLACKSCHOLES-1	0,321
		USER1-BLACKSCHOLES-2	0,223
		USER2-BODYTRACK2	0,959
		USER4-X2641	0,646

Table 3.3: Statistical results in the SPSS analysis of the NPB-OMP and PASERC suite [KVM X LXC].

3.5.2 Second Hypothesis

The performance characteristics of the scientific and enterprise workloads are statistically different among deployed cloud scenarios.

To test this hypothesis, it was formalized by dividing them into: Compare the LXC-based cloud computing workloads (NPB suite) against the native environment and second compare KVM-based cloud workloads (NPB suite) against the native environment. In addition, the variables used in the statistical hypotheses are: EtK for execution time in KVM cloud-based EtL for execution time in LXC cloud-based and EtN for execution time in native environment.

The results of the NPB-OMP suite (LXC X Native) were treated by SPSS as follows:

- *H0:* EtL == EtN
- *H1:* EtL != EtN

Through the statistical analysis, it is possible to reject the alternative hypothesis (H1), because most of the results indicate non-significant differences between the LXC Cloud-Based and Native environment and, therefore, assume null hypothesis (H0).

The results of the NPB-OMP suite (KVM X Native) were treated by SPSS as follows:

- *H0:* EtK == EtN
- *H1:* EtK != EtN

In turn, after the statistical analysis, it is possible to reject the null hypothesis (H0) because most of the results indicate significant differences between the KVM Cloud-Based and Native environment and, therefore assume the alternative hypothesis (H1).

The Table 3.4 is divided between the results of the NPB-OMP (LXC x Native) and NPB-OMP (KVM x Native) suite, with the green column being the partition. The results are arranged according to the scenario (with gray background), in this way in the first column of each test (LXC x Native and KVM x Native) of the NAS-OMP suite are represented the applications, number of threads and in the multi-tenancy scenarios the corresponding user. In the second column of each test (LXC x Native and KVM x Native) is represented the Sig. variable respecting the rule of addition of the results with less occurrence. In case of NPB-OMP (LXC x Native) the lowest number of occurrences was Sig. < 0,05, or statistically different. On the other hand, in case of NPB-OMP (KVM x Native) the lowest number of occurrences was Sig. > 0,05, that is, statistically non-different. It can be concluded that the NPB-OMP suite execution times present non-significant differences in the comparison of LXC cloud-based and native environment. However, in the comparison of NPB-OMP suite execution times among KVM cloud-based and native environment the statistical results indicate significant differences, emphasizing the overhead of KVM cloud-based.

NPB-OMP SUITE (LXC X NATIVE)			**NPB-OMP SUITE (KVM X NATIVO)**	
High Performance			**High Performance**	
Application-Threads	**Sig.**		**Application-Threads**	**Sig.**
BT-1	0,005		EP-5	0,383
BT-2	0,005		EP-7	0,414
CG-2	0,013		EP-8	0,074
EP-5	0,028		FT-7	0,991
MG-4	0,036		FT-8	0,209

Continued on next page

Table 3.4 – Continued from previous page

UA-1	0,005
UA-3	0,005
UA-7	0,037
BT-3	0,032
BT-4	0,015
BT-5	0,000
CG-3	0,021
CG-5	0,020
CG-7	0,036
FT-1	0,021
LU-3	0,001
MG-7	0,008
MG-8	0,000
SP-8	0,011
UA-2	0,000
UA-4	0,000
UA-6	0,001
UA-8	0,005
Multi-Tenancy Same 2 Users	
User-Application-Threads	**Sig.**
USER1-BT-4	0,007
USER1-UA-4	0,005
USER2-SP-4	0,022
USER2-UA-1	0,005
USER2-UA-2	0,005
USER2-UA-3	0,017
USER1-BT-1	0,017
Multi-Tenancy Same 2 Users	
User-Application-Threads	**Sig.**
USER1-CG-3	0,103
USER1-IS-2	0,959
USER1-IS-3	0,333
USER1-IS-4	0,333
USER1-MG-4	0,333
USER2-FT-4	0,386
USER2-IS-3	0,083
USER2-MG-2	0,126
USER2-MG-3	0,074
USER2-SP-4	0,508
USER1-FT-4	0,362
USER1-MG-2	0,868
USER1-MG-3	0,171
USER2-EP-4	0,305
Multi-Tenancy Same 3 Users	
User-Application-Threads	**Sig.**
USER1-CG-1	0,799
USER1-IS-2	0,575
USER1-SP-2	0,646
USER2-CG-1	0,203
USER2-CG-2	0,799
USER2-LU-1	0,878
USER2-MG-1	0,539
USER2-MG-2	0,059
USER2-SP-1	0,959

Continued on next page

Table 3.4 – Continued from previous page

USER1-LU-4	0,000		USER2-SP-2	0,059
USER1-SP-1	0,023		USER2-UA-2	0,445
USER1-UA-2	0,000		USER1-FT-2	0,717
USER2-BT-4	0,041		USER1-MG-2	0,148
USER2-CG-2	0,016		USER2-UA-1	0,131
USER2-IS-2	0,002		USER3-MG-2	0,832
USER2-LU-4	0,001		USER3-SP-2	0,111
USER2-SP-2	0,004		USER3-UA-2	0,406
Multi-Tenancy Same 3 Users			**Multi-Tenancy Same 4 Users**	
User-Application-Threads	**Sig.**		**User-Application-Threads**	**Sig.**
USER1-IS-1	0,005		USER1-CG-1	0,059
USER1-IS-2	0,020		USER1-CG-2	0,074
USER1-LU-1	0,005		USER1-LU-1	0,799
USER2-IS-1	0,033		USER1-SP-2	0,074
USER2-SP-2	0,009		USER1-UA-2	0,139
USER3-EP-1	0,005		USER2-CG-1	0,878
USER3-MG-2	0,007		USER2-CG-2	0,241
USER3-SP-2	0,013		USER2-IS-2	0,683
USER1-FT-1	0,007		USER2-LU-2	0,241
USER1-LU-2	0,001		USER2-MG-2	0,059
USER1-MG-1	0,004		USER2-SP-1	0,508
USER1-SP-1	0,034		USER3-IS-1	0,092
USER1-UA-1	0,012		USER3-IS-2	0,284
USER1-UA-2	0,014		USER4-IS-2	0,646
USER2-CG-2	0,026		USER4-LU-1	0,508
USER2-EP-1	0,047		USER4-MG-2	0,575
USER2-LU-1	0,013		USER4-SP-1	0,139

Continued on next page

Table 3.4 – Continued from previous page

USER2-LU-2	0,001	🟩	USER4-SP-2	0,959
USER2-MG-1	0,037	🟩	USER1-LU-2	0,056
USER2-MG-2	0,028	🟩	USER1-MG-1	0,464
USER3-BT-2	0,000	🟩	USER1-SP-1	0,532
USER3-CG-1	0,011	🟩	USER2-LU-1	0,945
USER3-CG-2	0,001	🟩	USER2-MG-1	0,512
USER3-FT-2	0,019	🟩	USER2-SP-2	0,070
USER3-LU-2	0,013	🟩	USER2-UA2	0,759
USER3-UA-1	0,000	🟩	USER3-FT-1	0,062
Multi-Tenancy Same 4 Users		🟩	USER3-LU-1	0,141
User-Application-Threads	**Sig.**	🟩	USER3-MG-1	0,120
USER2-UA-1	0,016	🟩	USER3-MG-2	0,209
USER3-FT-2	0,013	🟩	USER3-SP-1	0,119
USER3-LU-2	0,013	🟩	USER3-SP-2	0,358
USER3-UA-1	0,007	🟩	USER4-BT-2	0,204
USER3-UA-2	0,005	🟩	USER4-FT-1	0,062
USER4-IS-2	0,014	🟩	USER4-IS-1	0,526
USER1-CG-1	0,001	🟩	USER4-MG-1	0,226
USER1-FT-2	0,035	🟩	**Multi-Tenancy Diff 2 Users**	
USER1-LU-2	0,029	🟩	**User-Application-Threads**	**Sig.**
USER1-UA-1	0,001	🟩	USER1-IS-3	0,683
USER2-CG-1	0,003	🟩	USER1-IS-4	0,721
USER2-FT-2	0,030	🟩	USER2-CG-3	0,093
USER3-CG-1	0,000	🟩	USER2-CG-4	0,203
USER3-FT-1	0,001	🟩	USER2-SP-4	0,074
USER3-LU-1	0,019	🟩	USER1-MG-3	0,471
USER4-CG-1	0,001	🟩	USER1-MG-4	0,254

Continued on next page

Table 3.4 – Continued from previous page

USER4-MG-1	0,041	USER2-SP-2	0,141
USER4-UA-1	0,000	**Multi-Tenancy Diff 3 Users**	
Multi-Tenancy Diff 2 Users		**User-Application-Threads**	**Sig.**
User-Application-Threads	**Sig.**	USER1-EP-2	0,074
USER2-UA-2	0,005	USER1-IS-2	0,333
USER2-UA-3	0,005	USER2-CG-1	0,838
USER2-UA-4	0,022	USER2-CG-2	0,959
USER2-UA-1	0,000	USER2-SP-1	0,093
Multi-Tenancy Diff 3 Users		USER2-LU-1	0,203
User-Application-Threads	**Sig.**	USER2-SP-2	0,365
USER1-EP-1	0,037	**Multi-Tenancy Diff 4 Users**	
USER3-MG-1	0,005	**User-Application-Threads**	**Sig.**
USER3-MG-2	0,005	USER2-MG-2	0,203
USER3-UA-2	0,005	USER3-CG-1	0,074
USER1-BT-1	0,002	USER3-CG-2	0,959
USER1-BT-2	0,012	USER4-FT-1	0,214
USER3-FT-2	0,000	USER4-FT-2	0,415
USER3-UA-1	0,000	USER4-FT-1	0,155
Multi-Tenancy Diff 4 Users		USER4-LU-1	0,071
User-Application-Threads	**Sig.**		
USER1-UA-2	0,010		

Table 3.4: Statistical results in the SPSS analysis of the NPB-OMP suite (LXC X Native and KVM X Native).

3.5.3 Third Hypothesis

The performance characteristics of the enterprise workloads on the deployed cloud scenarios are statistically different with respect to the native computing environment.

Therefore to test this hypothesis, it was formalized by dividing them by: Compare the LXC-based cloud workloads (Parsec suite) against the native environment and compare KVM-based cloud workloads (Parsec suite) against the native environment. In addition, the variables used in the statistical hypotheses are: EtK for execution time in KVM cloud-based EtL for execution time in LXC cloud-based and EtN for execution time in native environment.

The results of the PARSEC suite (LXC X Native) were treated by SPSS as follows:

- *H0:* EtL == EtN
- *H1:* EtL != EtN

Through the statistical analysis, it is possible to reject the null hypothesis (H0) because most of the results indicate significant differences between the LXC Cloud-Based and Native environment and, therefore assume the alternative hypothesis (H1).

The results of the PARSEC suite (KVM X Native) were treated by SPSS as follows:

- *H0:* EtK == EtN
- *H1:* EtK != EtN

In turn, after the statistical analysis, it is possible to reject the null hypothesis (H0) because most of the results indicate significant differences between the KVM Cloud-Based and Native environment and, therefore assume the alternative hypothesis (H1).

The Table 3.5 is divided between the results of the PARSEC (LXC x Native) and PARSEC (KVM x Native) suite, with the green column being the partition. The results are arranged according to the scenario (with gray background), in this way in the first column of each test (LXC x Native and KVM x Native) of the PARSEC suite are represented the applications, number of threads and in the multi-tenancy scenarios the corresponding user. In the second column of each test (LXC x Native and KVM x Native) is represented the Sig. variable respecting the rule of addition of the results with less occurrence. In both cases (LXC x Native and KVM x Native) the lowest number of occurrences was Sig. > 0.05, or statistically non-different. It can be concluded that the PARSEC suite execution times present significant differences in the comparison of LXC cloud-based against native environment and significant differences in execution times among KVM cloud-based and native environment.

PARSEC SUITE (LXC X NATIVE)			**PARSEC SUITE (KVM X NATIVE)**	
High Performance			**High Performance**	
Application-Threads	**Sig.**		**Application-Threads**	**Sig.**
RAYTRACE-1	0,241		STREAMCLUSTER-2	0,799
X264-1	0,241		STREAMCLUSTER-5	0,303
STREAMCLUSTER-3	0,575		X264-1	0,322
CANNEAL-2	0,721		STREAMCLUSTER-6	0,323
FERRET-7	0,051		CANNEAL-7	0,407
RAYTRACE-2	0,085		CANNEAL-8	0,714
STREAMCLUSTER-1	0,242		**Multi-Tenancy Same 2 Users**	
RAYTRACE-6	0,261		**User-Application-Threads**	**Sig.**

Continued on next page

Table 3.5 – Continued from previous page

FACESIM-4	0,285		USER1-FREQMINE-2	0,074
RAYTRACE-5	0,349		USER1-STREAMCLUSTER-3	0,074
RAYTRACE-4	0,354		USER2-FREQMINE-2	0,074
FACESIM-3	0,428		USER2-STREAMCLUSTER-1	0,139
FREQMINE-4	0,431		USER2-CANNEAL-4	0,203
CANNEAL-8	0,459		USER2-STREAMCLUSTER-4	0,333
RAYTRACE-8	0,515		USER1-STREAMCLUSTER-1	0,445
RAYTRACE-3	0,779		USER1-STREAMCLUSTER-4	0,445
RAYTRACE-7	0,918		USER1-FREQMINE-3	0,508
Multi-Tenancy Same 2 Users			USER2-FREQMINE-3	0,508
User-Application-Threads	**Sig.**		USER2-FREQMINE-4	0,646
USER2-STREAMCLUSTER-1	0,074		USER2-BODYTRACK-4	0,721
USER2-SWAPTIONS-3	0,093		USER1-X264-1	0,799
USER1-STREAMCLUSTER-1	0,114		USER2-BLACKSCHOLES-2	0,054
USER2-STREAMCLUSTER-2	0,139		USER1-X264-4	0,119
USER2-CANNEAL-4	0,333		USER2-STREAMCLUSTER-3	0,182
USER1-STREAMCLUSTER-3	0,445		USER1-CANNEAL-1	0,197
USER2-STREAMCLUSTER-3	0,721		USER2-CANNEAL-2	0,265
USER1-STREAMCLUSTER-4	0,799		USER2-CANNEAL-1	0,421
USER1-X264-1	0,799		USER2-FACESIM-4	0,704
USER1-CANNEAL-2	0,878		USER1-STREAMCLUSTER-2	0,906
USER2-STREAMCLUSTER-4	0,878		**Multi-Tenancy Same 3 Users**	
USER2-FACESIM-1	0,055		**User-Application-Threads**	**Sig.**
USER1-SWAPTIONS-2	0,064		USER1-FREQMINE-1	0,074
USER1-X264-4	0,083		USER1-FREQMINE-2	0,074
USER2-X264-3	0,084		USER2-DEDUP-2	0,139
USER1-RAYTRACE-1	0,091		USER2-STREAMCLUSTER-2	0,241

Continued on next page

Table 3.5 – Continued from previous page

USER2-RAYTRACE-2	0,098	USER2-X264-1	0,285
USER1-X264-3	0,108	USER1-BLACKSCHOLES-2	0,172
USER2-FACESIM-4	0,126	USER2-CANNEAL-2	0,195
USER1-RAYTRACE-4	0,126	USER2-BLACKSCHOLES-1	0,426
USER1-CANNEAL-1	0,138	USER1-STREAMCLUSTER-2	0,444
USER1-RAYTRACE-3	0,150	USER2-CANNEAL-1	0,826
USER2-CANNEAL-1	0,195	**Multi-Tenancy Same 4 Users**	
USER1-CANNEAL-4	0,292	**User-Application-Threads**	**Sig.**
USER2-FERRET-4	0,298	USER4-FACESIM-2	0,059
USER1-CANNEAL-3	0,302	USER2-FREQMINE-1	0,074
USER1-STREAMCLUSTER-2	0,658	USER3-FACESIM-2	0,074
USER1-RAYTRACE-2	0,682	USER3-CANNEAL-1	0,093
USER2-CANNEAL-3	0,705	USER4-STREAMCLUSTER-1	0,241
USER2-RAYTRACE-1	0,990	USER4-STREAMCLUSTER-2	0,285
USER1-X264-2	0,995	USER2-X264-1	0,445
Multi-Tenancy Same 3 Users		USER2-RAYTRACE-1	0,061
User-Application-Threads	**Sig.**	USER3-STREAMCLUSTER-1	0,078
USER2-CANNEAL-1	0,059	USER3-BLACKSCHOLES-1	0,078
USER1-SWAPTIONS-2	0,093	USER1-STREAMCLUSTER-1	0,083
USER2-STREAMCLUSTER-2	0,114	USER2-STREAMCLUSTER-1	0,125
USER1-STREAMCLUSTER-1	0,139	USER4-CANNEAL-1	0,128
USER3-FREQMINE-1	0,169	USER1-STREAMCLUSTER-2	0,331
USER3-STREAMCLUSTER-1	0,169	USER3-STREAMCLUSTER-2	0,363
USER1-X264-1	0,203	USER2-STREAMCLUSTER-2	0,897
USER3-FERRET-2	0,203	**Multi-Tenancy Diff 2 Users**	
USER1-RAYTRACE-2	0,333	**User-Application-Threads**	**Sig.**
USER2-BLACKSCHOLES-1	0,333	USER2-CANNEAL-4	0,074

Continued on next page

Table 3.5 – Continued from previous page

USER3-X264-2	0,878
USER3-CANNEAL-1	0,959
USER3-STREAMCLUSTER-2	0,070
USER1-BLACKSCHOLES-2	0,123
USER1-CANNEAL-2	0,144
USER1-FREQMINE-2	0,158
USER3-CANNEAL-2	0,255
USER2-X264-1	0,298
USER1-X264-2	0,367
USER2-RAYTRACE-2	0,431
USER1-RAYTRACE-1	0,458
USER1-FREQMINE-1	0,552
USER1-STREAMCLUSTER-2	0,672
USER2-RAYTRACE-1	0,772
USER2-CANNEAL-2	1,000
Multi-Tenancy Same 4 Users	
User-Application-Threads	**Sig.**
USER2-FREQMINE-1	0,093
USER1-X264-1	0,169
USER2-RAYTRACE-2	0,203
USER3-FACESIM-2	0,203
USER3-FREQMINE-1	0,241
USER4-STREAMCLUSTER-2	0,241
USER2-FREQMINE-2	0,386
USER3-STREAMCLUSTER-2	0,508
USER3-RAYTRACE-2	0,575
Multi-Tenancy Diff 2 Users	
USER2-DEDUP-2	0,093
Multi-Tenancy Diff 3 Users	
User-Application-Threads	**Sig.**
USER1-CANNEAL-1	0,258
USER1-CANNEAL-2	0,447
Multi-Tenancy Diff 4 Users	
User-Application-Threads	**Sig.**
USER4-X264-1	0,386

Continued on next page

Table 3.5 – Continued from previous page

User-Application-Threads	**Sig.**
USER2-X264-1	0,646
USER4-X264-1	0,721
USER1-RAYTRACE-2	0,799
USER3-CANNEAL-1	0,799
Multi-Tenancy Diff 3 Users	
User-Application-Threads	**Sig.**
USER1-FREQMINE-1	0,064
USER2-STREAMCLUSTER-2	0,106
USER1-FREQMINE-2	0,112
USER1-CANNEAL-1	0,117
USER1-FACESIM-1	0,125
USER2-STREAMCLUSTER-1	0,197
USER1-FACESIM-2	0,255
USER1-FERRET-2	0,366
Multi-Tenancy Diff 4 Users	
User-Application-Threads	**Sig.**
USER2-X264-2	0,480
USER2-RAYTRACE-1	0,560
USER2-CANNEAL-2	0,566
USER1-CANNEAL-2	0,588
USER1-X264-2	0,611
USER1-STREAMCLUSTER-2	0,798
USER2-CANNEAL-1	0,819
USER1-RAYTRACE-1	0,841
USER1-BLACKSCHOLES-1	0,680
Multi-Tenancy Different 2 Users	

Continued on next page

Table 3.5 – Continued from previous page

User-Application-Threads	**Sig.**	
USER2-CANNEAL-4	0,074	🟩
USER2-FACESIM-4	0,169	🟩
USER1-RAYTRACE-3	0,285	🟩
USER1-FERRET-4	0,508	🟩
USER1-RAYTRACE-4	0,508	🟩
USER1-X264-1	0,508	🟩
USER1-RAYTRACE-2	0,053	🟩
USER2-STREAMCLUSTER-2	0,098	🟩
USER1-FREQMINE-4	0,195	🟩
USER1-RAYTRACE-1	0,203	🟩
Multi-Tenancy Different 3 Users		🟩
User-Application-Threads	**Sig.**	🟩
USER1-CANNEAL-2	0,169	🟩
USER1-CANNEAL-1	0,203	🟩
USER2-X264-1	0,203	🟩
Multi-Tenancy Different 4 Users		🟩
User-Application-Threads	**Sig.**	🟩
USER4-STREAMCLUSTER-1	0,139	🟩
USER4-RAYTRACE-1	0,333	🟩
USER4-STREAMCLUSTER-2	0,445	🟩
USER4-RAYTRACE-2	0,508	🟩
USER1-CANNEAL-2	0,215	🟩

Table 3.5: Statistical results in the SPSS analysis of the PARSEC suite (LXC X Native and KVM X Native).

CONCLUSION

Private IaaS clouds bring the potential of this technology to the organization-managed infrastructure, stablishing a pool of virtualized resources, that combines rapid elasticity, on-demand service and provision of resources. However, porting latency-sensitive applications to the virtualized cloud environment can result in unexpected performance issues.

In this thesis, a performance characterization was conducted using the CloudStack IaaS tool to run applications from scientific and enterprise fields using different virtualization technologies. To better represent the cloud environment and also simulate the concurrency between the multi-tenancy environment, a set of combination of experiments and methodologies were applied to each proposed scenario. Thus, the experiments were performed combining a total of four users, using the same and different applications. Then, a discussion was made comparing the results with the native environment, which served as the comparative baseline. In addition, a statistical process was applied to the data collected to verify if they are significantly different in the presented scenario. Finally, two papers are created and accepted in the ERAD event. One of them evaluates the performance of the management operations (create and delete instances) with the OpenStack and CloudStack cloud management tools Maliszewski et al. (2017) and the other makes a performance characterization of pipeline applications in KVM and LXC instances with CloudStack Baum et al. (2017).

The complexity of the proposed experiments added with the combination of features presented by the cloud management tool, makes it possible to effectively evaluate the proposed workloads using different virtualizations technologies. The result was a large number of experiments, which gives the opportunity to make a deep research designed to characterize the two different applications domains (scientific and enterprise) in the cloud environment. Furthermore, container-based technology and the full virtualization analyses can be used for further studies to characterized the applications running in this type of environment, because in general, these virtualization technologies are extensively used, whether in the cloud or not. However, the characterization also extends to the native environment, especially as it is frequently used for many researches as a baseline to comparative methods.

This knowledge achieved from the presented work goes further and the characterizations of the multi-tenancy environment demonstrate to be more challenging than expected, demanding a lot of research and even inducing the utilization of a trace application used for monitoring purposes. However, it gives an important overview of how applications behave effectively in this environment, which is known to compete intensely for machine resources.

The proposed HPC scenarios demonstrated, in most cases, an expected result of the native and KVM environment. Expectations were high with regard to LXC container, because it is well-debated in the academia, demonstrating low overhead and have a near-native performance. However, the experiments reveal the opposite in some workloads, with LXC in some cases exhibiting an impressive poor performance when compared to KVM virtualization. It was quite intriguing, and a different approach was required to respond to this unexpected behavior. Therefore, a new test set was prepared to investigate as discussed in subsection 3.3.3, and the result was assigned to NFS used by CloudStack, which induces some performance degradations, especially in the case of intense disk I/O applications. This reveals that the cloud deployment used is not favorable for this workloads.

In the case of same application scenario, it was possible to identify the fierce dispute for computing resources among the instances, exhibiting an unpredictable result in many applications. This became evident as more users were added, presenting major distortions in the native and LXC environment, especially for applications that make an intense use of the LLC cache, such as Freqmine. In this scenario, KVM-based cloud has proved to be more predictable and performs better with significant resource isolation, revealing the benefits of the full virtualization technologies.

On the other hand, the experiments conducted simulating multi-tenancy using different applications, presented more predictable and stable results in comparison to the same applications, essentially for native and LXC environments, where KVM showed some overhead. However, it is important to mention that, especially in the case of native and LXC, much of the performance interference suffered by the instance is due to the execution of the workload in the neighbor instance, that relies only on the mechanisms of the operating system, which implements a basic resource isolation.

The container-based virtualization is not a new technology it is indeed promising and has become a hot topic because it is known to demonstrate, by referenced studies, to have a near-native performance with good levels of resource isolation, which is a demanding requirement for cloud computing. In this study, LXC presented low overhead in case of CPU and memory workload. However, an issue that is not usually discussed by other studies, is the implementation of this technology, which in the case of CloudStack, the NFS is responsible for providing VM access to the disk, inevitably using the network for such operations. Therefore, this study indirectly discovered that it may be a wrong choice for applications that requires intense disk usage, as presented by Dedup and Vips in subsection 3.3.3. In addition, an important fact discussed was the inefficient resource isolation presented in the same scenario with the same applications, where the rampant competition for resources causes unpredictable results in certain workloads.

In general, KVM exhibits a more stable and predictable performance. This is possible due to the driver's maturity of this technology, which among other benefits, implements buffer mechanisms that provide relatively good disk performance. In addition, the high levels of isolation provided, such as presented in the case of same applications scenario, are clearly noticed, where uncontrolled competition for computing resources in the native and LXC induced significant variations among the workloads. However, the overhead introduced by the virtualization layer is more noticeable in applications that are more CPU demanding.

In the case of Parsec benchmark suit, although the diverse number of workloads, in general, a similar pattern allowed to estimate the behavior of some applications in the virtualized cloud environment after a few experiments performed. A correlation was made to the processor cache miss with the size of the application working set. Thus, the larger the working set, the greater was the cache miss and more sensitive to the memory latency. Therefore, applications that have more performance variation, especially in the case of native and LXC multi-tenancy, were most dependent on LLC cache, memory latency and disk I/O operations. In addition, the serial portions of the stages presented in some programs negatively affect overall performance and scalability, especially in the case of pipeline parallelism, which has serial input and output stages.

In relation to the scientific workloads represented by the NPB-OMP suite, it can be seen and verified that most of them have applications that are CPU-bound and memory-bound extensively exploiting the parallelism and available resources. Another proof already expected was the use of disk (I/O) in which low and sporadic use is verified. The characterization of scientific applications showed a relation between memory, CPU, and Cache Miss, in which it is possible to observe that applications with large work sets have problems with miss cache and scheduling related to processor utilization. In addition it is noticeable the degradation of performance that happens during context switching, in which even in milliseconds shown both in memory and in the processor, representing a loss of performance.

Cloudstack IaaS tool has proven to be challenging, especially due to the complexity of this environment that demonstrates to have some peculiarities. On the other hand, it proves to be an inspiring research test-bed, and once explored, it offers an extensible number of features. However, attention is necessary to use some technologies, especially as demonstrated in the case of LXC deployment. Essentially because the VM instantiated in Cloudstack uses NFS to access the disk in primary storage, while resources such as CPU and memory are provided by the compute node chosen to run the instances. The results are poor LXC performance, which does not implement the mechanisms necessary to manage disk access under these circumstances. Consequently, further investigation is required to confirm whether this issue is in fact a Cloudstack limitation, because even setting the specific compute node to host the LXC instance, the container data is manipulated on the front end node. This results in increasing latency, especially in the case of smaller data.

The cloud IaaS is capable of supporting a wide range of technologies and much wider applications domains. The key question is to predict whether the state of the art cloud environment has the resources available to provide the reasonable performance needed to become a compelling place to meet demanding applications. Moreover, it hosts a diverse number of applications in a concurrent multi-tenancy environment. In this research, we conclude that the cloud deployment presented is not recommended for intensive disk I/O applications running on LXC containers, because disk access is done through NFS, which deteriorates the disk performance. While it is important to mention that it is not an LXC issue, but a possible Cloudstack deployment limitation, however further investigation is necessary regarding this issue. In addition, LXC has some limitations in respect to the resource isolation, and its utilization in the production environment, especially in the case of the same applications that are cache demanding, reveals to be counter-productive. However, despite this fact, the LXC is a promising virtualization technology that is still evolving and expects its maturity to bring enhancements. KVM proves to be most effective in this case, and it is more recommended if the need is to achieve efficient division of computational resources but at the cost of some overhead.

While different virtualization technologies introduce new features, all applications demand the same thing, hardware resources. Therefore, it is important to correlate workloads types, virtualization technologies and hardware, after all, all layers of features depend on hardware. In our research, as well as the related works, it becomes clear the effect that the applications presented in conditions with low cache availability and memory latency, and as more instances are added, there is more concurrency for existent resources. Thus, it is critical to provide the hardware infrastructure required to host such applications.

Finally, to have an efficient cloud environment, it is recommended to deploy different applications with different hardware necessities. Then choose between different virtualization technologies, where the container promisses to have near-native performance, except in the case of the Cloudstack deployment, or on the other hand, choose the KVM, which has some overhead, but a better resource isolation.

3.5.4 Hypothesis Validation

The first hypothesis says that the performance characteristics of the scientific and enterprise workloads are statistically different among deployed cloud scenarios. This hypothesis is not entirely true because, although the vast majority of the results present significant differences in the comparison of the LXC and KVM-based clouds in relation to the scientific and enterprise workloads, it can be seen in the Table 3.3 that there are results with non-significant differences.

The second hypothesis proposes that the performance characteristics of the scientific workloads in the deployed cloud scenarios are statistically different with respect to the native computing environment. This hypothesis is not entirely true. The comparison of the LXC-based cloud against the native environment shows that the results are not significant, that is, the majority of the results indicate that the LXC-based cloud and the native environment have very close results in the performance of scientific workloads, however as can be seen in the left side of the Table 3.4, there are results with significant difference. On the other hand, the comparison of the KVM-based cloud against the native environment shows that the results are significantly different, however in the same way it can be seen in the right side of the Table 3.4, that there are also non-significant results, then this hypothesis becomes partially true.

Finally, the third hypothesis affirms that the performance characteristics of enterprise workloads in the deployed cloud scenarios are statistically different with respect to the native computing environment. This hypothesis is not entirely true. In both comparisons, the results of LXC or KVM compared to the native environment show significant differences in the execution of enterprise workloads. However, as can be seen in the Table 3.5, there are also results that show non-significant differences, so that there are not 100% of the results with significant differences, this hypothesis becomes partially true.

3.6 FUTURE WORKS

As future works, we intend to proceed in this area by investigating the applications behavior using other forms of workloads implementations, such as the Message Passing Interface (MPI), which is a standard for distributed memory used to achieve another form of parallelism. In this way, we can analyze the impact of applications using different computational power and expanding the analysis on the applications running in the cloud.

CloudStack has had quite disappointing results when deploying LXC with some workloads, it also requires a further investigation to verify if there is a different deployment method to avoid the NFS limitations in relation to disk I/O operations. In addition, the workloads that shows the behavior studied should be submitted into other cloud management tools to analyze how the underlying technologies affect it. Through this, we can compare different cloud management tools for a specific workload.

Furthermore, in this thesis, the experiments are performed over the CloudStack IaaS cloud platform and therefore, a future work is to make the same evaluation and use the same methodologies with other IaaS Cloud Tool platforms, such as OpenStack or OpenNebula.

Containers-based virtualization is a promising technology that proves to be competitive to frequently used full virtualization. It gives the opportunity to search for other operating system-level virtualization such as OpenVZ and software container like Docker. Thus, it should also be characterized in the cloud environment applying the same methodology used in present work. This will help the costumers and cloud providers to choose what best fits to a certain cloud-scenario and it also brings more knowledge to the research community.

Multi-tenancy is a complex environment that applies competition among users for computing resources. This presents many variations in the execution times of the workloads. In this thesis, it was not possible to deepen this subject to the point of obtaining all the answers about the functioning of this environment. Therefore, a new work specifically focused on multi-tenancy becomes necessary.

In this thesis it was not possible to deepen this theme so that all the answers to the functioning of the environment can be obtained. Therefore, a new work focused specifically on multi-tenancy becomes necessary.

Finally, the practice of overcommitment, which refers to the allocation of more than one vCPU per physical core Ghosh and Naik (2012), is not yet explored in the literature. According to McDougall and Anderson (2010), the number of vCPUs per physical core is referred to the consolidation ration and the exact number is a trade secret for cloud providers. In addition, as emphasized by Nikounia et al. (2015), taking into account these two facts, the results presents up to 16x slowdown due to the noise of the neighbors, which is a high bottleneck compared to the performance of a single vCPU VMs without overcommitment. Therefore, overcommitment must be compared and characterized to determine the performance of the applications and discover the bottlenecks.

REFERENCES

ADRIANO VOGEL CARLOS A. F. MARON, V. L. L. B. F. S. C. S. D. G. HiPerfCloud: um projeto de alto desempenho em nuvem. In: JORNADA DE PESQUISA SETREM, 14., Três de Maio, Brazil. **Anais. . .** SETREM, 2015. p.4.

AL-MUKHTAR, M. M.; MARDAN, A. A. A. Performance Evaluation of Private Clouds Eucalyptus versus CloudStack. **International Journal of Advanced Computer Science and Applications**, [S.l.], p.1–10, 2014.

AMDAHL, G. M. Validity of the single processor approach to achieving large scale computing capabilities. In: APRIL 18-20, 1967, SPRING JOINT COMPUTER CONFERENCE. **Proceedings. . .** [S.l.: s.n.], 1967. p.483–485.

ANTONIOU, A. Performance Evaluation of Cloud Infrastructure using Complex Workloads. In: OF THE . **Anais. . .** [S.l.: s.n.], 2012. p.1–79.

BADGER, L. et al. **Cloud Computing Synopsis using and Recommendations**. [S.l.]: Computer Security Division, Information Technology Laboratory, National Institute of Standards and Technology, 2012. 81p. v.1.

BAER, J.-L. **Multiprocessor Architecture**. [S.l.]: Cambridge University Press, 2010.

BAILEY, D. H. The NAS Parallel Benchmarks. , [S.l.], November 2009.

BAILEY, D. H. et al. The NAS parallel benchmarks. **The International Journal of Supercomputing Applications**, [S.l.], v.5, n.3, p.63–73, 1991.

BARKER et al. Empirical evaluation of latency-sensitive application performance in the cloud. In: ACM SIGMM CONFERENCE ON MULTIMEDIA SYSTEMS. **Proceedings...** [S.l.: s.n.], 2010. p.35–46.

BARKER, S. K.; SHENOY, P. Empirical Evaluation of Latency-sensitive Application Performance in the Cloud. **Departament of Comuter Science, University of Massachusetts Amhrest**, [S.l.], p.12, 2010.

BARNEY, B. What is parallel computing. **Introduction to Parallel Computing**, [S.l.], 2012.

BARROW-WILLIAMS et al. A communication characterisation of Splash-2 and Parsec. In: WORKLOAD CHARACTERIZATION, 2009. IISWC 2009. IEEE INTERNATIONAL SYMPOSIUM ON. **Anais...** [S.l.: s.n.], 2009. p.86–97.

BAUM, W. et al. Caracterização do Desempenho de Aplicações Pipeline em Instâncias KVM e LXC de uma Nuvem CloudStack. In: ESCOLA REGIONAL DE ALTO DESEMPENHO DO ESTADO DO RIO GRANDE DO SUL (ERAD/RS), 17., Ijuí, RS, Brazil. **Anais...** Sociedade Brasileira de Computação, 2017.

BESERRA, D.; ENDO, P. T.; BARRETO, J. Performance Evaluation of a Lightweight Virtualization Solution for HPC I/O Scenarios. **International Conference on Systems, Man, and Cybernetics**, [S.l.], p.1–8, 2016.

BESERRA, D. et al. Performance Analysis of LXC for HPC Environments. **9th International Conference on Complex, Intelligent, and Software Intensive Systems**, [S.l.], p.6, 2015.

BESERRA, D. et al. Performance analysis of LXC for HPC environments. In: COMPLEX, INTELLIGENT, AND SOFTWARE INTENSIVE SYSTEMS (CISIS), 2015 NINTH INTERNATIONAL CONFERENCE ON. **Anais...** [S.l.: s.n.], 2015. p.358–363.

BHADAURIA, M.; WEAVER, V. M.; MCKEE, S. A. Understanding PARSEC performance on contemporary CMPs. In: WORKLOAD CHARACTERIZATION, 2009.

IISWC 2009. IEEE INTERNATIONAL SYMPOSIUM ON. **Anais...** [S.l.: s.n.], 2009. p.98–107.

BIENIA, C. **Benchmarking Modern Multiprocessors**. [S.l.]: Princeton University, 2011.

BIENIA, C. et al. The PARSEC benchmark suite: characterization and architectural implications. In: PARALLEL ARCHITECTURES AND COMPILATION TECHNIQUES, 17. **Proceedings...** [S.l.: s.n.], 2008. p.72–81.

BIENIA, C.; LI, K. Parsec 2.0: a new benchmark suite for chip-multiprocessors. In: ANNUAL WORKSHOP ON MODELING, BENCHMARKING AND SIMULATION, 5. **Proceedings...** [S.l.: s.n.], 2009. v.2011.

BLOKLAND, K.; MENGERINK, J.; POL, M. **Testing Cloud Services**: how to test saas, paas and iaas. [S.l.]: Rocky Nook, 2013.

BUTENHOF, D. R. **Programming with POSIX threads**. [S.l.]: Addison-Wesley Professional, 1997.

BUYYA, R.; BROBERG, J.; GOSCINSKI, A. **Cloud Computing**: principles and paradigms. [S.l.]: Wiley, 2010. (Wiley Series on Parallel and Distributed Computing).

BUYYA, R. et al. Cloud computing and emerging IT platforms: vision, hype, and reality for delivering computing as the 5th utility. **Future Generation computer systems**, [S.l.], v.25, n.6, p.599–616, 2009.

BUYYA, R.; VECCHIOLA, C.; SELVI, S. T. **Mastering Cloud Computing**: foundations and applications programming. 1st.ed. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2013.

CAFARO, M.; ALOISIO, G. **Grids, Clouds and Virtualization**. [S.l.]: Springer London, 2010. (Computer Communications and Networks).

CAMPOS, E. et al. Performance Evaluation of Virtual Machines Instantiation in a Private Cloud. **2015 IEEE World Congress on Services**, [S.l.], p.1–8, 2015.

CARROLL, M.; KOTZé, P.; MERWE, A. van der. Securign Virtual and Cloud Environments. **Service Science: Reaserch and Innovations in the Service Economy**, [S.l.], 2012.

CARVER, R. H.; TAI, K.-C. **Modern Multithreading** : implementing, testing, and debugging multithreaded java and c++/pthreads/win32 programs. [S.l.]: Wiley-Interscience, 2005.

CHAKTHRANONT, N. et al. Exploring the performance impact of virtualization on an HPC cloud. In: CLOUD COMPUTING TECHNOLOGY AND SCIENCE (CLOUDCOM), 2014 IEEE 6TH INTERNATIONAL CONFERENCE ON. **Anais...** [S.l.: s.n.], 2014. p.426–432.

CHANDRASEKARAN, K. **Essentials of Cloud Computing**. [S.l.]: Taylor & Francis, 2014.

CHANG, V. **Delivery and Adoption of Cloud Computing Services in Contemporary Organizations**. [S.l.]: IGI Global, 2015. (Advances in Systems Analysis, Software Engineering, and High Performance Computing).

CHANG, V.; WALTERS, R. J.; WILLS, G. The development that leads to the Cloud Computing Business Framework. **International Journal of Information Management**, http://www.sciencedirect.com/science/article/pii/S026840121300008X, v.33, n.3, p.524 – 538, 2013.

CHAPMAN, B.; JOST, G.; PAS, R. van der. **Using OpenMP**: portable shared memory parallel programming. [S.l.]: MIT Press, 2008. n.v. 10. (Scientific Computation Series).

CHASAPIS, D. et al. PARSECSs: evaluating the impact of task parallelism in the parsec benchmark suite. **ACM Transactions on Architecture and Code Optimization (TACO)**, [S.l.], v.12, n.4, p.41, 2016.

CHENG, Y.; CHEN, W. Evaluation of Virtual Machine Performance on NUMA Multicore Systems. In: **Anais...** [S.l.: s.n.], 2013. p.1–8.

CHENG, Y.; CHEN, W. Evaluation of virtual machine performance on NUMA multicore systems. In: P2P, PARALLEL, GRID, CLOUD AND INTERNET COMPUTING (3PGCIC), 2013 EIGHTH INTERNATIONAL CONFERENCE ON. **Anais. . .** [S.l.: s.n.], 2013. p.136–143.

CHEVERESAN, R. et al. Characteristics of Workloads Used in High Performance and Technical Computing. In: ANNUAL INTERNATIONAL CONFERENCE ON SUPERCOMPUTING, 21., New York, NY, USA. **Proceedings. . .** ACM, 2007. p.73–82. (ICS 07).

CISCO. **Data Center Architecture Overview** <**http**://www.cisco.com/c/en/us/td/docs/solutions/enterprise/data_center/dc_infra2_5/dcinfra_1.html#wp1066094>. Last access mar, 2017.

CLOUDSTACK. **Official Page**<**https**://cloudstack.apache.org/>. Last access jan, 2017.

CLOUDSTACK. **Concepts and Terminology**< **http**://docs.cloudstack.apache.org/en latest/concepts.html#about-clu>. Last access jan, 2017.

CLOUDSTACK. **Host LXC Installationy**< **http**://docs.cloudstack.apache.org/projects/cloudstack-installation/en/4.9/hypervisor/lxc.html>. Last access jan, 2017.

COUTINHO, E. F.; PAILLARD, G.; SOUZA, J. N. de. Performance Analysis on Scientific Computing and Cloud Computing Environments. **Proceedings of the 7th Euro American Conference on Telematics and Information Systems**, [S.l.], p.1–6, 2014.

CUPITT, J.; MARTINEZ, K. VIPS: an image processing system for large images. In: ELECTRONIC IMAGING: SCIENCE & TECHNOLOGY. **Anais. . .** [S.l.: s.n.], 1996. p.19–28.

DUKARIC, R.; JURIC, M. B. Towards a unified taxonomy and architecture of cloud frameworks. **Future Generation Computer Systems**, [S.l.], v.29, n.5, p.1196–1210, 2013.

EL-REWINI, H.; ABD-EL-BARR, M. **Advanced Computer Architecture and Parallel Processing**. [S.l.]: Wiley, 2005. (Wiley Series on Parallel and Distributed Computing).

EVOY, G. M.; MURY, A. R.; SCHULZE, B. An analysis of definition and placement of virtual machines for high performance applications on Clouds. **Concurrency and Computation Pratice and Experience**, [S.l.], p.1789–1814, 2014.

EXPÓSITO, R. R. et al. Evaluation of messaging middleware for high-performance cloud computing. **Personal and Ubiquitous Computing**, [S.l.], v.17, n.8, p.1709–1719, 2013.

FELTER, W. et al. An updated performance comparison of virtual machines and linux containers. In: PERFORMANCE ANALYSIS OF SYSTEMS AND SOFTWARE (ISPASS). **Anais...** [S.l.: s.n.], 2015. p.171–172.

FENG, H. et al. **Unstructured Adaptive (UA) NAS parallel benchmark**. [S.l.]: version 1.0. Technical report, NASA Technical Report NAS-04-006, 2004.

FERDMAN, M. et al. Clearing the clouds: a study of emerging scale-out workloads on modern hardware. In: ACM SIGPLAN NOTICES. **Anais...** [S.l.: s.n.], 2012. v.47, n.4, p.37–48.

FERNÁNDEZ, J. et al. SCE Toolboxes for the development of high-level parallel applications. In: INTERNATIONAL CONFERENCE ON COMPUTATIONAL SCIENCE. **Anais...** [S.l.: s.n.], 2006. p.518–525.

FIELD, A. **Discovering Statistics Using SPSS**. [S.l.]: SAGE Publications, 2009. (ISM (London, England)).

FIELD, A. **Discovering Statistics Using IBM SPSS Statistics**. [S.l.]: SAGE Publications, 2013. (Discovering Statistics Using IBM SPSS Statistics: And Sex and Drugs and Rock 'n' Roll).

FISHER, R. **Statistical Methods For Research Workers**. [S.l.]: Cosmo Publications, 1925. (Cosmo study guides).

FOSTER, I.; KESSELMAN, C.; TUECKE, S. The anatomy of the grid: enabling scalable virtual organizations. **International journal of high performance computing applications**, [S.l.], v.15, n.3, p.200–222, 2001.

FOUNDATION, O. **The Crossroads of Cloud and HPC**: openstack for scientific research: exploring openstack cloud computing for scientific workloads. [S.l.]: CreateSpace Independent Publishing Platform, 2016.

FREUND, J. **Estatística Aplicada Economicamente**. [S.l.]: Bookman, 2009.

FRUMKIN, M. A.; SHABANO, L. **Arithmetic data cube as a data intensive benchmark**. [S.l.: s.n.], 2003.

FRUMKIN, M. Data flow pattern analysis of scientific applications. In: WORKSHOP ON PATTERNS IN HIGH PERFORMANCE COMPUTING. **Anais...** [S.l.: s.n.], 2005.

GARCIA, G. A.; FREITAS, H. C. Avaliação de Desempenho de um Cluster Raspberry Pi com NAS Parallel Benchmarks. , Florianópolis, Santa Catarina, p.57–62, October 2015.

GHOSH, R.; NAIK, V. K. Biting off safely more than you can chew: predictive analytics for resource over-commit in iaas cloud. In: CLOUD COMPUTING (CLOUD), 2012 IEEE 5TH INTERNATIONAL CONFERENCE ON. **Anais...** [S.l.: s.n.], 2012. p.25–32.

GHOSHAL, D.; CANON, R. S.; RAMAKRISHNAN, L. I/O Performance of Virtualized Cloud Environments. **DataCloud-SC '11 Proceedings of the second international workshop on Data intensive computing in the clouds**, [S.l.], p.71–80, 2011.

GHOSHAL, D.; CANON, R. S.; RAMAKRISHNAN, L. I/o performance of virtualized cloud environments. In: DATA INTENSIVE COMPUTING IN THE CLOUDS. **Proceedings...** [S.l.: s.n.], 2011. p.71–80.

GOLDWORM, B.; SKAMAROCK, A. **Blade Servers and Virtualization**: transforming enterprise computing while cutting costs. [S.l.]: Wiley, 2007.

GOTO, Y. Kernel-based virtual machine technology. **Fujitsu Scientific and Technical Journal**, [S.l.], v.47, p.362–368, 2011.

GROPP, W. et al. **Using Advanced MPI**: modern features of the message-passing interface. [S.l.]: MIT Press, 2014. (Computer science & intelligent systems).

GRÜN, T.; HILLEBRAND, M. A. NAS Integer Sort on multi-threaded shared memory machines. In: EUROPEAN CONFERENCE ON PARALLEL PROCESSING. **Anais. . .** [S.l.: s.n.], 1998. p.999–1009.

GUPTA, A. et al. HPC-Aware VM Placement in Infrastructure Clouds. **Cloud Engineering (IC2E), 2013 IEEE International Conference**, [S.l.], 2013.

GUPTA, A. et al. Improving HPC Application Performance in Cloud through Dynamic Load Balancing. **Cluster, Cloud and Grid Computing (CCGrid), 2013 13th IEEE/ACM International Symposium**, [S.l.], 2013.

GUPTA, A. et al. The Who, What, Why, and How of High Performance Computing in the Cloud. **Cloud Computing Technology and Science (CloudCom), 2013 IEEE 5th International Conference**, [S.l.], p.1–9, 2013.

GUPTA, A. et al. The who, what, why, and how of high performance computing in the cloud. In: CLOUD COMPUTING TECHNOLOGY AND SCIENCE (CLOUDCOM), 2013 IEEE 5TH INTERNATIONAL CONFERENCE ON. **Anais. . .** [S.l.: s.n.], 2013. v.1, p.306–314.

GUPTA, A.; KALE, L. V. Towards Efficient Mapping, Scheduling, and Execution of HPC Applications on Platforms in Cloud. **Parallel and Distributed Processing Symposium Workshops PhD Forum (IPDPSW), 2013 IEEE 27th International**, [S.l.], p.1–6, 2013.

GUPTA, A.; MILOJICIC, D. Evaluation of hpc applications on cloud. In: OPEN CIRRUS SUMMIT (OCS), 2011 SIXTH. **Anais. . .** [S.l.: s.n.], 2011. p.22–26.

GUSTAFSON, J. L.; SNELL, Q. O. HINT: a new way to measure computer performance. In: SYSTEM SCIENCES, 1995. PROCEEDINGS OF THE TWENTY-EIGHTH HAWAII INTERNATIONAL CONFERENCE ON. **Anais. . .** [S.l.: s.n.], 1995. v.2, p.392–401.

HAN, J.; PEI, J.; YIN, Y. Mining frequent patterns without candidate generation. In: ACM SIGMOD RECORD. **Anais. . .** [S.l.: s.n.], 2000. v.29, n.2, p.1–12.

HASHIMOTO, Y.; AIDA, K. Evaluation of Performance Degradation in HPC Applications with VM Consolidation. **Third International Conference on Networking and Computing**, [S.l.], p.1–5, 2012.

HENNESSY, J. L.; PATTERSON, D. A. **Computer Architecture A Quantitative Approach**. [S.l.]: Morgan Kaufmann, 2012.

HONG, C.-H. et al. Performance Prediction and Evaluation of Parallel Applications in KVM, Xen, and VMware. **European Conference on Parallel Processing**, [S.l.], p.1–12, 2014.

HONG, C.-H. et al. Performance prediction and evaluation of parallel applications in KVM, Xen, and VMware. In: EUROPEAN CONFERENCE ON PARALLEL PROCESSING. **Anais. . .** [S.l.: s.n.], 2014. p.99–110.

HP. **Stripped-down grid** : a lightweight grid for computing's have-nots < http://www.hpl.hp.com/news/2005/jan-mar/grid.html>. Last access jan, 2017.

HUANG, Q. et al. Evaluating open-source cloud computing solutions for geosciences. **Computers Geosciences**, [S.l.], 2013.

HUANG, W. et al. A case for high performance computing with virtual machines. In: SUPERCOMPUTING, 20. **Proceedings. . .** [S.l.: s.n.], 2006. p.125–134.

HUBER, N. et al. Evaluating and Modeling Virtualization Performance Overhead for Cloud Environments. In: CLOSER. **Anais...** [S.l.: s.n.], 2011. p.563–573.

HURWITZ, J. et al. **Cloud Computing For Dummies**. [S.l.]: Wiley Publishing, Inc., 2010.

IBM. **LXC**: linux container tools <http://www.ibm.com/developerworks/linux/library/l-lxc-containers/>. Last access jan, 2017.

INTEL. **Intel® Virtualization Technology for Directed I/O (VT-d)**: enhancing intel platforms for efficient virtualization of i/o devices <https://software.intel.com/en-us/articles/intel-virtualization-technology-for-directed-io-vt-d-enhancing-intel-platforms-for-efficient-virtualization-of-io-devices>. Last access jan, 2017.

INTEL. **Multithreaded Code Optimization in PARSEC 3.0**: blackscholes< https://software.intel.com/en-us/articles/multithreaded-code-optimization-in-parsec-30-blackscholes>. Last access jan, 2017.

IOSUP, A. et al. Performance Analysis of Cloud Computing Services for Many-Tasks Scientific Computing. **IEEE Transactions on Parallel and Distributed Systems**, [S.l.], v.22, n.6, p.931–945, June 2011.

ISO/IEC-17788. Information technology — Cloud computing — Overview and vocabulary. , [S.l.], 10 2014.

JACKSON, K.; BUNCH, C.; SIGLER, E. **OpenStack Cloud Computing Cookbook**. [S.l.]: Packt Publishing, 2015.

JANSSEN, C. L.; NIELSEN, I. M. B. **Parallel Computing in Quantum Chemistry**. [S.l.]: CRC Press, 2008.

JAYASINGHE, D. et al. Variations in performance and scalability: an experimental study in iaas clouds using multi-tier workloads. **IEEE Transactions on Services Computing**, [S.l.], v.7, n.2, p.293–306, 2014.

JIANG, T. et al. Micro-architectural characterization of desktop cloud workloads. In: WORKLOAD CHARACTERIZATION (IISWC), 2012 IEEE INTERNATIONAL SYMPOSIUM ON. **Anais. . .** [S.l.: s.n.], 2012. p.131–140.

JIN, H.; WIJNGAART, R. F. V. der. NAS Parallel Benchmarks, Multi-Zone Versions. , [S.l.], p.9, July 2003.

JUVE, G. et al. Scientific Workflow Applications on Amazon EC2. , [S.l.], p.8, 2009.

KAVIS, M. **Architecting the Cloud**: design decisions for cloud computing service models (saas, paas, and iaas). [S.l.]: Wiley, 2014. (Wiley CIO).

KERBYSON, D. J. et al. A performance comparison of current HPC systems: blue gene/q, cray xe6 and infiniband systems. **Future Generation Computer Systems**, [S.l.], p.1–14, 2014.

KIRKPATRICK, S. et al. Optimization by simmulated annealing. **science**, [S.l.], v.220, n.4598, p.671–680, 1983.

KUDRYAVTSEV, A. et al. Virtualizing HPC applications using modern hypervisors. In: CLOUD SERVICES, FEDERATION, AND THE 8TH OPEN CIRRUS SUMMIT, 2012. **Proceedings. . .** [S.l.: s.n.], 2012. p.7–12.

LEITE, D. et al. Performance Evaluation of Virtual Machine Monitors for Cloud Computing. **2012 13th Symposium on Computing Systems**, [S.l.], p.1–7, 2012.

LI, C.; XIE, J.; ZHANG, X. Performance evaluation based on open source cloud platforms for high performance computing. In: INTELLIGENT NETWORKS AND INTELLIGENT SYSTEMS (ICINIS), 2013 6TH INTERNATIONAL CONFERENCE ON. **Anais. . .** [S.l.: s.n.], 2013. p.90–94.

LI, D.; HUANG, S.; CAMERON, K. CG-Cell: an npb benchmark implementation on cell broadband engine. In: INTERNATIONAL CONFERENCE ON DISTRIBUTED COMPUTING AND NETWORKING. **Anais. . .** [S.l.: s.n.], 2008. p.263–273.

LIBVIRT. **What is libvirt?** <**http**://wiki.libvirt.org/page/faq#what_is_libvirt.3f>. Last access jan, 2017.

LIBVIRT. **Virtio** < **https**://wiki.libvirt.org/page/virtio>. Last access jan, 2017.

LINUX. **KSM** <**http**://www.linux-kvm.org/page/ksm>. Last access jan, 2017.

LINUXPLANET. **Virtualizing the Embedded World**: vista over linux in a cell phone? <http://www.linuxplanet.com/linuxplanet/reports/6490/1/screenshot3735/>. Last access jan, 2017.

LXC. **Official Page**< **https**://linuxcontainers.org/>. Last access jan, 2017.

MAHMOOD, Z. **Cloud Computing Methods and Practical Approaches**. [S.l.]: Springer, 2013.

MALISZEWSKI, A. M. et al. Desempenho das Operações de Criar e Deletar Instâncias KVM Simultâneas em Nuvens CloudStack e OpenStack. In: ESCOLA REGIONAL DE ALTO DESEMPENHO DO ESTADO DO RIO GRANDE DO SUL (ERAD/RS), 17., Ijuí, RS, Brazil. **Anais...** Sociedade Brasileira de Computação, 2017.

MARINESCU, D. C. **Cloud Computing**: theory and practice. [S.l.]: Elsevier, 2013.

MARON, C. A. F. **Avaliação e Comparação da Computação de Alto Desempenho em Ferramentas Opensource de Administração de Nuvem Usando Estações De Trabalho**. 2014. Dissertação (Mestrado em Ciência da Computação) — Undergraduate Thesis, Sociedade Educacional Três de Maio (SETREM), Três de Maio, RS, Brazil.

MARON, C. A. F. et al. Avaliação e Comparação do Desempenho das Ferramentas OpenStack e OpenNebula. In: ESCOLA REGIONAL DE REDES DE COMPUTADORES (ERRC), 12., Canoas. **Anais...** Sociedade Brasileira de Computação, 2014. p.1–5.

MARON, C. A. F. et al. Desempenho de OpenStack e OpenNebula em Estações de Trabalho: uma avaliação com microbenchmarks e npb. **Revista Eletrônica**

Argentina-Brasil de Tecnologias da Informação e da Comunicação (REABTIC), Três de Maio, Brazil, v.6, n.1, p.15, December 2016.

MARON, C. A. F.; GRIEBLER, D.; SCHEPKE, C. Comparação das Ferramentas OpenNebula e OpenStack em Nuvem Composta de Estações de Trabalho. In: ESCOLA REGIONAL DE ALTO DESEMPENHO DO ESTADO DO RIO GRANDE DO SUL (ERAD/RS), 14., Alegrete, RS, Brazil. **Anais...** Sociedade Brasileira de Computação, 2014. p.173–176.

MATTHEWS, J. et al. **Running Xen**: a hands-on guide to the art of virtualization. [S.l.]: Pearson Education, 2008.

MATTSON, T.; SANDERS, B.; MASSINGILL, B. **Patterns for Parallel Programming**. [S.l.]: Pearson Education, 2004. (Software Patterns Series).

MAUCH, V.; KUNZE, M.; HILLENBRAND, M. High performance cloud computing. **International journal of high performance computing applications**, [S.l.], v.9, n.2, 2012.

MAUCH, V.; KUNZE, M.; HILLENBRAND, M. High performance cloud computing. **Future Generation Computer Systems**, [S.l.], v.29, n.6, p.1408–1416, 2013.

MCDOUGALL, R.; ANDERSON, J. Virtualization performance: perspectives and challenges ahead. **ACM SIGOPS Operating Systems Review**, [S.l.], v.44, n.4, p.40–56, 2010.

MEHROTRA, P. et al. Performance Evaluation of Amazon EC2 for NASA HPC Applications. **ScienceCloud '12 Proceedings of the 3rd workshop on Scientific Cloud Computing**, [S.l.], p.41–50, 2012.

MELL, P.; GRACE, T. **The NIST definition of Clous Computing**. [S.l.]: Computer Security Division, Information Technology Laboratory, National Institute of Standards and Technology, 2011. 7p. v.1.

MORABITO, R.; KJÄLLMAN, J.; KOMU, M. Hypervisors vs. lightghweit virtualization: a performance comparison. In: CLOUD ENGINEERING (IC2E), 2015 IEEE INTERNATIONAL CONFERENCE ON. **Anais. . .** [S.l.: s.n.], 2015. p.386–393.

MUKHEDKAR, P.; VETTATHU, A.; CHIRAMMAL, H. **Mastering KVM Virtualization**. [S.l.]: Packt Publishing, Limited, 2016.

MULERIKKAL, J. P.; SASTRI, Y. A Comparative Study of OpenStack and CloudStack. **Advances in Computing and Communications (ICACC), 2015 Fifth International Conference**, [S.l.], p.1–4, 2015.

MÜLLER, M.; CHARYPAR, D.; GROSS, M. Particle-based fluid simulation for interactive applications. In: ACM SIGGRAPH/EUROGRAPHICS SYMPOSIUM ON COMPUTER ANIMATION, 2003. **Proceedings. . .** [S.l.: s.n.], 2003. p.154–159.

NAVARRO, A. et al. Analytical modeling of pipeline parallelism. In: PARALLEL ARCHITECTURES AND COMPILATION TECHNIQUES, 2009. PACT'09. 18TH INTERNATIONAL CONFERENCE ON. **Anais. . .** [S.l.: s.n.], 2009. p.281–290.

NEUHAUS, C. et al. A Practical Evaluation of Searchable Encryption for Data Archives in the Cloud. In: INTERNATIONAL CONFERENCE ON CLOUD COMPUTING AND SERVICES SCIENCE. **Anais. . .** [S.l.: s.n.], 2015. p.171–192.

NICHOLS, B.; BUTTLAR, D.; FARRELL, J. **Pthreads programming**: a posix standard for better multiprocessing. [S.l.]: O'Reilly Media, Inc., 1996.

NICHOLS, B.; BUTTLAR, D.; FARRELL, J. P. **Pthreads Programming**: a posix standard for better multiprocessing. 1.ed. [S.l.]: O'Reilly Media, 1996. (O'Reilly Nutshell).

NIKOUNIA, S. H. et al. Hypervisor and Neighbors' Noise: performance degradation in virtualized environments. **IEEE Transactions on Services Computing**, [S.l.], 2015.

NIKOUNIA, S. H.; MOHAMMADI, S. Hypervisor and Neighbors' Noise: performance degradation in virtualized environments. **IEEE Transactions on Services Computing**, [S.l.], p.1–11, 2015.

NPB. **NAS Parallel Benchmarks** <**https**://www.nas.nasa.gov/publications/npb.html>. Last access jan, 2017.

OGRIZOVIĆ, D.; CAR, Z.; KOVAČIĆ, B. Scientific Applications in Cloud Computing. **The IPSI BgD Transactions on Advanced Research**, [S.l.], v.10, n.1, p.27–33, 2014.

OKADA, D. K.; GOLDMAN, A.; CAVALHEIRO, G. G. H. Using NAS Parallel Benchmarks to Evaluate HPC Performance in Clouds. **International Symposium on Network Computing and Applications**, [S.l.], p.1–4, 2016.

OPENSTACK. **Official Page** <**https**://www.openstack.org>. Last Access in jan, 2017.

OPENSTACK. **Manila**<**https**://wiki.openstack.org/wiki/manila>. Last access jan, 2017.

PARADOWSKI, A.; LIU, L.; YUAN, B. Benchmarking the Performance of Openstack and Cloudstack. In: IEEE 17TH INTERNATIONAL SYMPOSIUM ON OBJECT/COMPONENT/SERVICE-ORIENTED REAL-TIME DISTRIBUTED COMPUTING, 2014. **Anais...** [S.l.: s.n.], 2014.

PARSEC. **Princeton Application Repository for Shared-Memory Computers** <**http**://parsec.cs.princeton.edu/overview.htm>. Last access jan, 2017.

PARSEC. **PARSEC** <**http**://wiki.cs.princeton.edu/index.php/parsec#blackscholes>. Last access jan, 2017.

PARSEC. **PARSEC** <**http**://wiki.cs.princeton.edu/index.php/parsec#bodytrack>. Last access jan, 2017.

PARSEC. **PARSEC** <**http**://wiki.cs.princeton.edu/index.php/parsec#facesim>. Last access jan, 2017.

PENNYCOOK, S. J. et al. Performance Analysis of a Hybrid MPICUDA Implementation of the NASLU Benchmark. **SIGMETRICS Perform. Eval. Rev.**, New York, NY, USA, v.38, n.4, p.23–29, Mar. 2011.

PFLANZNER, T. et al. Performance Analysis of an OpenStack Private Cloud. , [S.l.], v.2, p.282–289, 2016.

PILLA, L. L. Análise de desempenho da arquitetura CUDA utilizando os NAS parallel benchmarks. , Porto Alegre, RS, Brazil, 2009.

PORTNOY, M. **Virtualization Essentials**. [S.l.]: Wiley, 2016.

PRODAN, R.; OSTERMANN, S. A survey and Taxonomy of Infrastructure as a Service and Web Hosting Cloud Providers. **Grid Computing 10th IEEE/ACM International Conference**, [S.l.], p.17–25, 2009.

PUJOLLE, G. **Software Networks**: virtualization, sdn, 5g, security. [S.l.]: Wiley, 2015. n.v. 1. (Networks & Telecommunication: Advanced Networks).

QEMU. **Main Page** <**http**://wiki.qemu.org/main_page>. Last access jan, 2017.

RAMACHANDRAN, A. et al. Performance Evaluation of NAS Parallel Benchmarks on Intel Xeon Phi. **Parallel Processing (ICPP), 2013 42nd International Conference**, [S.l.], p.1–8, 2013.

REDDY, P. V. V.; RAJAMANI, L. Evaluation of different hypervisors performance in the private cloud with SIGAR framework. **International Journal of Advanced Computer Science and Applications**, [S.l.], v.5, n.2, 2014.

REGOLA, N.; DUCOM, J. C. Recommendations for Virtualization Technologies in High Performance Computing. In: IEEE 2º CLOUDCOM, 2010. **Anais...** [S.l.: s.n.], 2010. p.409–416.

ROLOFF, E. et al. High Performance Computing in the cloud: deployment, performance and cost efficiency. In: CLOUD COMPUTING TECHNOLOGY AND SCIENCE (CLOUDCOM), 2012 IEEE 4TH INTERNATIONAL CONFERENCE ON. **Anais...** [S.l.: s.n.], 2012. p.371–378.

RON LARSON, B. F. **Estatística Aplicada**. 4ª.ed. [S.l.]: Pearson Prentice Hall, 2010.

ROSSO, J. P. **Análise de Desempenho de Aplicações Científicas em Ambiente de Nuvem Privada**. 2015. Dissertação (Mestrado em Ciência da Computação) — Universidade Federal do Pampa (UNIPAMPA), Alegrete, RS, Brazil.

RUIZ, C.; JEANVOINE, E.; NUSSBAUM, L. Performance evaluation of containers for HPC. **10th Workshop on Virtualization in High-Performance Cloud Computing**, [S.l.], p.1–13, 2015.

RUPARELIA, N. **Cloud Computing**. [S.l.]: MIT Press, 2016. (The MIT Press Essential Knowledge series).

RUPARELIA, N. B. **Cloud Computing**. [S.l.]: MIT Press, 2016.

SABHARWAL, N. **Apache CloudStack Cloud Computing**. [S.l.]: Packt Publishing, 2013.

SADOOGHI, I. et al. Understanding the performance and potential of cloud computing for scientific applications. **IEEE Transactions on Cloud Computing**, [S.l.], 2015.

SAINI, S. et al. An Application-Based Performance Evaluation of NASA's Nebula Cloud Computing Platform. **High Performance Computing and Communication 2012 IEEE 9th International Conference on Embedded Software and Systems (HPCC-ICESS), 2012 IEEE 14th International Conference**, [S.l.], p.1–8, 2012.

SAKR, S.; GABER, M. **Large Scale and Big Data**: processing and management. [S.l.]: Auerbach Publications,CRC Press, 2014.

SCHEEPERS, M. J. Virtualization and containerization of application infrastructure: a comparison. In: TWENTE STUDENT CONFERENCE ON IT, 21. **Anais. . .** [S.l.: s.n.], 2014. p.1–7.

SHRIVASTWA, A.; SARAT, S. **Learning OpenStack**. [S.l.]: Packt Publishing, 2015.

SHROFF, G. **Enterprise Cloud Computing**: technology, architecture, applications. [S.l.]: Cambridge University Press, 2010.

SILBERSCHATZ, A.; GALVIN, P.; GAGNE, G. **Operating System Concepts**. [S.l.]: Wiley, 2012.

SILVA, M.; RYU, K. D.; DA SILVA, D. VM performance isolation to support qos in cloud. In: PARALLEL AND DISTRIBUTED PROCESSING SYMPOSIUM WORKSHOPS & PHD FORUM (IPDPSW), 2012 IEEE 26TH INTERNATIONAL. **Anais. . .** [S.l.: s.n.], 2012. p.1144–1151.

SMOOT, S.; TAN, N. **Private Cloud Computing**: consolidation, virtualization, and service-oriented infrastructure. [S.l.]: Elsevier Science, 2011.

SNIR, M. **MPI–the Complete Reference**: the mpi core. [S.l.]: MIT press, 1998. v.1.

SOSINSKY, B. **Cloud Computing Bible**. [S.l.]: Wiley, 2010. (Bible).

SOTOMAYOR, B. et al. Capacity leasing in cloud systems using the opennebula engine. In: WORKSHOP ON CLOUD COMPUTING AND ITS APPLICATIONS. **Anais. . .** [S.l.: s.n.], 2008. v.3.

SOUTHERN, G.; RENAU, J. Analysis of PARSEC Workload Scalability. **Performance Analysis of Systems and Software (ISPASS)**, [S.l.], p.1–0, 2016.

STEINMETZ, D. et al. Cloud computing performance benchmarking and virtual machine launch time. In: INFORMATION TECHNOLOGY EDUCATION, 13. **Proceedings. . .** [S.l.: s.n.], 2012. p.89–90.

STONEBRAKER, M. et al. Enterprise Database Applications and the Cloud: a difficult road ahead. In: IEEE INTERNATIONAL CONFERENCE ON CLOUD ENGINEERING, 2014. **Anais. . .** [S.l.: s.n.], 2014. p.1–6.

STRAZDINS, P. E. et al. Scientific application performance on hpc, private and public cloud resources: a case study using climate, cardiac model codes and the npb benchmark suite. In: PARALLEL AND DISTRIBUTED PROCESSING SYMPOSIUM WORKSHOPS & PHD FORUM (IPDPSW), 2012 IEEE 26TH INTERNATIONAL. **Anais. . .** [S.l.: s.n.], 2012. p.1416–1424.

TAN, T.; ARZBERGER, P.; KONAGAYA, A. **Grid Computing in Life Sciences**: lsgrid2005, the second international life science grid workshop, biopolis, singapore, 5-6 may 2005. [S.l.]: World Scientific, 2006.

TANENBAUM, A. S.; AUSTIN, T. **Organização Estruturada de Computadores**. [S.l.]: Pearson, 2013.

TANENBAUM, A. S.; BOS, H. **Modern Operating Systems**. 4th.ed. [S.l.]: Pearson, 2014.

TANENBAUM, A.; STEEN, M. van. **Distributed Systems**: principles and paradigms. [S.l.]: Pearson Prentice Hall, 2007.

TELFER, S. **The Crossroads of Cloud and HPC**: openstack for scientific research. [S.l.]: CreateSpace, 2016.

TORALDO, G. **OpenNebula 3 Cloud Computing**. [S.l.]: Packt Publishing, 2012.

UEDA, Y.; NAKATANI, T. Performance Variations of Two Open-Source Cloud Platforms. In: WORKLOAD CHARACTERIZATION (IISWC). **Anais. . .** [S.l.: s.n.], 2010. p.1–10.

VARGHESE, B. et al. Container-Based Cloud Virtual Machine Benchmarking. In: CLOUD ENGINEERING (IC2E), 2016 IEEE INTERNATIONAL CONFERENCE ON. **Anais. . .** [S.l.: s.n.], 2016. p.192–201.

VECCHIOLA, C.; PANDEY, S.; BUYYA, R. High-Performance Cloud Computing A View of Scientific Applications. , [S.l.], p.13, 2009.

VIRT-MANAGER. **Manage virtual machines with virt-manager** <**https**://virt-manager.org/>. Last access jan, 2017.

VMWARE. **Understanding Full Virtualization, Paravirtualization, and Hardware Assist** <**http**://www.vmware.com/content/dam/digitalmarketing/vmware/en/pdf/techpaper/vmware_paravirtualization.pdf>. Last Acess in mar, 2017.

VOGEL, A. **Surveying the Robustness and Analyzing the Performance Impact of Open Source Infrastructure as a Service Management Tools**. 2015. Dissertação (Mestrado em Ciência da Computação) — Undergraduate Thesis, Sociedade Educacional Três de Maio (SETREM), Três de Maio, RS, Brazil.

VOGEL, A. et al. Private IaaS Clouds: a comparative analysis of opennebula, cloudstack and openstack. In: EUROMICRO INTERNATIONAL CONFERENCE ON PARALLEL, DISTRIBUTED AND NETWORK-BASED PROCESSING (PDP), 24., Heraklion Crete, Greece. **Anais. . .** IEEE, 2016. p.672–679.

VOGEL, A. et al. Medindo o Desempenho de Implantações de OpenStack, CloudStack e OpenNebula em Aplicações Científicas. In: ESCOLA REGIONAL DE ALTO DESEMPENHO DO ESTADO DO RIO GRANDE DO SUL (ERAD/RS), 16., São Leopoldo, RS, Brazil. **Anais. . .** Sociedade Brasileira de Computação, 2016. p.279–282.

VOGEL, A. et al. An Intra-Cloud Networking Performance Evaluation on CloudStack Environment. In: EUROMICRO INTERNATIONAL CONFERENCE ON PARALLEL, DISTRIBUTED AND NETWORK-BASED PROCESSING (PDP), 25., St. Petersburg, Russia. **Anais. . .** IEEE, 2017. p.5.

WAKU, G. M. Massage Passing Interface(MPI). , [S.l.], p.13, 2012.

WALTERS, J. P. et al. A comparison of virtualization technologies for HPC. In: ADVANCED INFORMATION NETWORKING AND APPLICATIONS, 2008. AINA 2008. 22ND INTERNATIONAL CONFERENCE ON. **Anais. . .** [S.l.: s.n.], 2008. p.861–868.

WHITE, S.; ALUND, A.; SUNDERAM, V. S. Performance of the NAS parallel benchmarks on PVM-Based networks. **Journal of Parallel and Distributed Computing**, [S.l.], v.26, n.1, p.61–71, 1995.

WHITTED, T. An improved illumination model for shaded display. In: ACM SIGGRAPH 2005 COURSES. **Anais. . .** [S.l.: s.n.], 2005. p.4.

WIJNGAART, R. F.; HAOPIANG, J. **Nas parallel benchmarks, multi-zone versions**. [S.l.: s.n.], 2003.

WIJNGAART, R. F. Van der; FRUMKIN, M. NAS grid benchmarks version 1.0. , [S.l.], 2002.

WIJNGAART, R. F. Van der; SRIDHARAN, S.; LEE, V. W. Extending the BT NAS parallel benchmark to exascale computing. In: INTERNATIONAL CONFERENCE ON HIGH PERFORMANCE COMPUTING, NETWORKING, STORAGE AND ANALYSIS. **Proceedings...** [S.l.: s.n.], 2012. p.94.

WILTER, A. et al. The biopaua project: a portal for molecular dynamics using grid environment. In: BRAZILIAN SYMPOSIUM ON BIOINFORMATICS. **Anais...** [S.l.: s.n.], 2005. p.214–217.

WONG, P.; DER WIJNGAART, R. NAS parallel benchmarks I/O version 2.4. **NASA Ames Research Center, Moffet Field, CA, Tech. Rep. NAS-03-002**, [S.l.], 2003.

XAVIER, M. G. 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.

XAVIER, M. G. et al. A Performance Isolation Analysis of Disk-intensive Workloads on Container-based Clouds. **Parallel, Distributed and Network-Based Processing (PDP), 2015 23rd Euromicro International Conference**, [S.l.], p.1–8, 2015.

XU, F. et al. Managing performance overhead of virtual machines in cloud computing: a survey, state of the art, and future directions. **Proceedings of the IEEE**, [S.l.], v.102, n.1, p.11–31, 2014.

XU, F.; LIU, F.; VASILAKOS, A. V. Managing Performance Overhead in Cloud Computing: a survey, state of the art, and future directions. **IEEE**, [S.l.], v.102, p.21, 2014.

YANG, L. T.; GUO, M. **High-Performance Computing** : paradigm and infrastructure. [S.l.]: Wiley-Interscience, 2005.

ZHANG, J.; LU, X.; K., D. Performance Characterization of Hypervisor and Container-based Virtualization for HPC on SR-IOV Enabled InfiniBand Clusters. **International Parallel and Distributed Processing Symposium Workshop**, [S.l.], p.1–8, 2016.

ZHANG, J.; LU, X.; PANDA, D. K. Performance Characterization of Hypervisor-and Container-Based Virtualization for HPC on SR-IOV Enabled InfiniBand Clusters. In: PARALLEL AND DISTRIBUTED PROCESSING SYMPOSIUM WORKSHOPS, 2016 IEEE INTERNATIONAL. **Anais. . .** [S.l.: s.n.], 2016. p.1777–1784.

ZHANG, Y. et al. Parallelization of the nas conjugate gradient benchmark using the global arrays shared memory programming model. In: PARALLEL AND DISTRIBUTED PROCESSING SYMPOSIUM, 2005. PROCEEDINGS. 19TH IEEE INTERNATIONAL. **Anais. . .** [S.l.: s.n.], 2005. p.8–pp.

ZHOU, L. et al. Virtual Machine Scheduling for Parallel Soft Real-Time Applications. **Modeling, Analysis Simulation of Computer and Telecommunication Systems (MASCOTS), 2013 IEEE 21st International Symposium**, [S.l.], p.1–10, 2013.

ZHU, J. **Quantitative Models for Performance Evaluation and Benchmarking**: data envelopment analysis with spreadsheets. [S.l.]: Springer International Publishing, 2014. (International Series in Operations Research & Management Science).