



Politecnico di Milano

SCHOOL OF INDUSTRIAL AND INFORMATION ENGINEERING

Master of Science in Computer Science and Engineering

MASTER THESIS

Performance Measurements in a High Throughput Computing Environment

Advisor

Prof. Marco Gribaudo

Co-Advisor

Dr. Domenico Giordano

Candidate

Francesca Cecilia Schiavi

Matr.836597



Acknowledgement

I want to express all of my gratitude to the people who have been helpful in achieving the successful completion of this Thesis. Everyone has been a source of professional and personal inspiration with observations and suggestions, making me aware of my capacities and limits. I would like to thank my Advisor, Marco Griboaldo, for helping me in developing this Thesis and for his precious advice. I would like to show my deep gratitude to my Co-Advisor, Domenico Giordano. I can't be grateful enough for giving me the opportunity to start the project at CERN, and for his constant help in developing this project. He has been a mentor for me. He has always encouraged my research allowing my professional growth, motivating and steering me whenever I needed it. The guidance and support I received from my Advisor and Co-Advisor were vital for the success of the project. Much acknowledgement must be addressed to the CERN staff with which I had the chance to work. I would like to express all of my gratitude to the IT-CM group leader, Tim Bell, and to Jan van Eldik, the IT-CM-RPS section leader. They gave me the opportunity get in touch with the amazing CERN environment and allowed me to be part of this stimulating project. Moreover, I want to express my gratitude to the entire IT-CM-RPS section for the advice and materials they provided for me. In particular, I want to thank Cristóvão J. D. Cordeiro, who helped me during the project and encouraged me to trust myself. A special thanks also to Arne Wiebalck for his support and fruitful discussion. I would also like to thank Łukasz Góralczyk and James Hamilton for the friendship disclosed during the CERN Openlab experience. Not last, I would express all of my gratitude to Pietro Govoni, without whom this experience would not have started.

A special thank to my mum Elisabetta and my sister Anna, no words can express how grateful I am. They have always supported me with strong determination during these years of studies, encouraging me towards my goals and advising me in the best way possible. I would like to offer my sincere thanks to Leonardo who has always supported me, morally and practically, with perseverance and determination. The deep sentiment given to me during this time became my determination.

Milano, 28 Aprile 2017

Francesca Cecilia Schiavi

Contents

Introduction	1
1 Technological Challenges in Computing	3
1.1 Big Data	4
1.1.1 Big Data Fields	5
1.2 Big Data Computing Resources	6
1.2.1 Grid Computing	7
1.2.2 Cloud Computing	8
1.2.3 Similarities and Differences between Cloud and Grid	11
1.3 The CERN Big Data Use Case	12
1.3.1 Computing Needs for the LHC Experiments	15
1.3.2 Worldwide LHC Computing Grid Infrastructure	16
1.3.3 From WLCG to Cloud Resources	18
2 CERN Computing Infrastructure	19
2.1 Short Overview of Data Centre History	19
2.2 CERN Data Centres	20
2.2.1 Evolution of the CERN Data Centres	21
2.3 Virtualisation Components	22
2.3.1 Virtual Machine	22
2.3.2 Hypervisor	23
2.4 OpenStack Cloud	24
2.4.1 Nova	24
2.4.2 Other Main Components	28
2.5 Tier-0 Batch System	29
3 Benchmarking Computing Resources	30
3.1 Performance Benchmarking	30
3.1.1 Benchmarking Process	31
3.2 Hardware Characteristics	32
3.2.1 CPU	32
3.2.2 RAM	34
3.2.3 Storage	34
3.3 Server Virtualisation	35
3.3.1 VM Provisioning Approach	36
3.4 Benchmarking Tools	38
3.4.1 Benchmark Suite	38
3.4.2 Benchmark Workload	39

3.4.3	Outline of the Benchmarks in the Configurations	42
3.4.4	Analysis Tools	43
4	Analysis of Benchmark Data	44
4.1	Data Handling	44
4.2	Configurations	45
4.2.1	VM-32	45
4.2.2	VM-16	50
4.2.3	VM-8	57
4.2.4	VM-4	72
4.2.5	VM-1	76
4.3	HEP-SPEC06 Measurements	79
4.4	Scaling Factors across Configurations	80
5	Building a Model	82
5.1	Queuing Network Model	82
5.2	Java Modelling Tool	83
5.3	Baseline of the Model	83
5.3.1	VM-16 and VM-8 ₄	84
5.3.2	Simulation of the VM-8 ₃ scenario	90
	Conclusions	94
A	Script to launch benchmarks	95
B	The Elastic Family Tools	96
B.1	Transportation Layer: Logstash	96
B.2	Storage Layer: Elasticsearch	96
B.3	Visualisation Layer: Kibana	97
C	Interaction with Elasticsearch and Kibana	99
C.1	Query to Extract Data from Elasticsearch	99
C.2	Benchmark Details in the Kibana Dashboard	101
D	Boxplot	103
	Bibliography	105

List of Figures

1.1	5V properties of big data	4
1.2	The three cloud dimensions	9
1.3	Cloud service levels	9
1.4	CERN accelerators complex.	13
1.5	Overall view of the LHC experiment	13
1.6	Detectors of the LHC experiment.	14
1.7	CMS proton-proton collision	15
1.8	Worldwide LHC Computing Grid	16
2.1	Meyrin data centre for the Tier-0 of the WLCG	20
2.2	The virtualisation concept	22
2.3	Graphical representation of the two types of hypervisors	23
2.4	Relationship between the hypervisor components	24
2.5	CERN's OpenStack architecture.	25
2.6	List of the Nova flavors hand at CERN	26
3.1	Graphical representation of the CPUs on each phisycal node	33
3.2	CERN Cloud Benchmarking Suite	38
3.3	Example of benchmarks executed in sequential and synchronized mode	39
3.4	SPECrate vs HEP-Spec06 Multiple Speed	42
3.5	Workflow from Elasticsearch to analysis	43
4.1	VM-32 - Dispersion of the results for the three benchmarks	46
4.2	VM-32 - Distribution of the DB12 values	47
4.3	VM-32 - Distribution of the WSN values	47
4.4	VM-32 - Distribution of the KV values	47
4.5	VM-32 - Dispersion of DB12 results for each VM	48
4.6	VM-32 - Dispersion of a VM's DB12 results for each half hour	49
4.7	VM-32 - Dispersion of DB12 results for each VM in a specific time	49
4.8	VM-32 - Dispersion of DB12 results	50
4.9	VM-16 - Dispersion of the results for the three benchmarks	51
4.10	VM-16 - Distribution for the DB12 values	52
4.11	VM-16 - Distribution of the WSN values	52
4.12	VM-16 - Distribution of the KV values	52
4.13	VM-16 - Dispersion of KV results for each VM	53
4.14	VM-16 - KV distribution for VM _{fast} and VM _{slow}	54
4.15	VM-16 _{fast} - Distribution of the KV values	54
4.16	VM-16 _{slow} - Distribution of the KV values	54
4.17	VM-16 - Difference between VM _{fast} and VM _{slow} classes	56

4.18	VM-16 - Percentage of VMs that perform faster than their sibling . . .	56
4.19	VM-16 - Distribution of DB12 results	57
4.20	VM-16 - Distribution of WSN results	57
4.21	VM-8 - Dispersion of the results for the three benchmarks	59
4.22	VM-8 - Distribution of the DB12 values	60
4.23	VM-8 - Distribution of the WSN values	60
4.24	VM-8 - Distribution of the KV values	60
4.25	VM-8 ₄ - Distribution of the DB12 values	62
4.26	VM-8 ₄ - Distribution of the WSN values	62
4.27	VM-8 ₄ - Distribution of the KV values	62
4.28	VM-8 ₄ - Dispersion of KV results for each VM	63
4.29	VM-8 ₄ - 25 th percentile detail	63
4.30	VM-8 ₄ - KV distribution for VM _{fast} and VM _{slow}	64
4.31	VM-8 ₄ - _{fast} - Distribution of the KV values	64
4.32	VM-8 ₄ - _{slow} - Distribution of the KV values	64
4.33	Pictorial representation of the VM-8 ₃ configuration.	65
4.34	VM-8 ₃ - Distribution of the DB12 values	66
4.35	VM-8 ₃ - Distribution of the WSN values	66
4.36	VM-8 ₃ - Distribution of the KV values	66
4.37	VM-8 ₃ - _{single} - KV distribution for VM _{fast} and VM _{slow}	67
4.38	VM-8 ₃ - _{single} - Dispersion of KV results	68
4.39	VM-8 ₃ - _{singlefast} - Distribution of the KV values	68
4.40	VM-8 ₃ - _{singleslow} - Distribution of KV results	68
4.41	VM-8 ₃ - _{couple} - Distribution of the DB12 values	69
4.42	VM-8 ₃ - _{couple} - Distribution of the WSN values	69
4.43	VM-8 ₃ - _{couple} - Distribution of the KV values	70
4.44	VM-8 ₃ - _{couple} - Dispersion of KV results in each VM	70
4.45	VM-8 ₃ - _{couple} - KV distribution for VM _{fast} and VM _{slow}	71
4.46	VM-8 ₃ - _{couplefast} - Distribution of the KV values	71
4.47	VM-8 ₃ - _{coupleslow} - Distribution of the KV values	71
4.48	VM-4 - Dispersion of the results for the three benchmarks	73
4.49	VM-4 - Distribution of DB12 values	74
4.50	VM-4 - Distribution of WSN values	74
4.51	VM-4 - KV distribution for VM _{fast} and VM _{slow}	74
4.52	VM-4 _{fast} - Distribution of the KV values	75
4.53	VM-4 _{slow} - Distribution of the KV values	75
4.54	VM-1 - Dispersion of the results for the three benchmarks	77
4.55	VM-1 - Distribution of the DB12 values	78
4.56	VM-1 - Distribution of the WSN values	78
4.57	VM-1 - Distribution of the KV values	78
4.58	HS06 - Performance comparison between bare-metal, VM-8 and VM-16	79
4.59	Performance comparison between scenarios	80
5.1	Model for VM-16 and VM-8 ₄ in WSN and DB12 simulation	84
5.2	VM-16, VM-8 ₄ - Distribution of DB12 values	86
5.3	VM-16, VM-8 ₄ - Distribution of WSN values	87
5.4	Model for VM-16 and VM-8 ₄ for KV simulation	88

5.5	VM-16, VM-8 ₄ - Distribution of KV values	89
5.6	Model for VM-8 ₃	90
5.7	VM-8 ₃ - Distribution of DB12 values	92
5.8	VM-8 ₃ - Distribution of WSN values	93
5.9	VM-8 ₃ - Distribution of KV values	93
B.1	Monitoring tools flow	96
B.2	Elasticsearch structure example	97
B.3	Kibana visualization tool.	98
C.1	Detail of a virtual machine benchmarking	101
D.1	Main components of a boxplot.	103
D.2	Relation between the boxplot and the distribution	104

List of Tables

1.1	Data rates for LHC detectors	15
1.2	Resources of the WLCG project	17
2.1	Amount of resources in the Meyrin and Wigner data centres	21
2.2	Attributes of the flavor configuration	26
2.3	List of the operative system's images	27
2.4	Amount of resources for the CERN OpenStack Cloud	28
3.1	Intel Xeon Processor E5-2630 v3 specifications	32
3.2	Pinning of the logical cores on the NUMA nodes	33
3.3	Samsung MZ7KM960HAHP-00005 SSD specifications	35
3.4	Virtual machines configuration for each scenario	37
3.5	ALL_CPP benchmark set	41
3.6	Summary of the benchmarks performed for each scenario	42
4.1	VM-32 - Premises about the virtual environment	45
4.2	VM-16 - Premises about the virtual environment	50
4.3	VM-8 - Premises about the virtual environment	57
4.4	VM-4 - Premises about the virtual environment	72
4.5	VM-1 - Premises about the virtual environment	76
4.6	Summary of the benchmarks mean in each scenario	81
5.1	Service times and response times for DB12 and WSN in VM-16 and VM-8 ₄	85
5.2	Service times and response times for KV in VM-16 and VM-8 ₄	88
5.3	Service times and response times for DB12 and WSN in VM-8 ₃	91
5.4	Service times and response times for KV in VM-8 ₃	91
C.1	Meaning of benchmark metadata	102

Listings

A.1	Examples of cron jobs used to run the benchmarks.	95
C.1	Query to JSON from Elastic.	99
C.2	Example of JSON from Elastic.	100

Abstract

The IT infrastructures of companies and research centres are implementing new technologies to satisfy the increasing need of computing resources for big data analysis. In this context, resource profiling plays a crucial role in identifying areas where the improvement of the utilisation efficiency is needed. In order to deal with the profiling and optimisation of computing resources, two complementary approaches can be adopted: the measurement-based approach and the model-based approach. The measurement-based approach gathers and analyses performance metrics executing benchmark applications on computing resources. Instead, the model-based approach implies the design and implementation of a model as an abstraction of the real system, selecting only those aspects relevant to the study.

This Thesis originates from a project carried out by the author within the CERN IT department. CERN is an international scientific laboratory that conducts fundamental researches in the domain of elementary particle physics. The project concerns the profiling of a set of 240 servers used to sustain the High Energy Physics workloads in the CERN batch system. The virtualised resources have been tested through a benchmarking process in order to extract performance metrics that have been analysed in detail. Finally, the creation of a model that simulates the real system behaviour, starting from the aforementioned measures, has been flanked to the analytical approach.

Sommario

Le infrastrutture IT delle aziende e dei centri di ricerca stanno implementando nuove tecnologie per soddisfare la crescente necessità di risorse computazionali, tali da supportare l'analisi di big data. In questo contesto, il monitoraggio delle risorse assume un ruolo fondamentale per l'identificazione dei settori in cui migliorare l'efficienza d'utilizzo. Le procedure di profilazione e di ottimizzazione delle risorse di calcolo possono essere affrontate mediante due approcci, quello analitico e quello modellistico. Il primo raccoglie e analizza metriche di performance eseguendo attività di benchmarking sulle risorse; il secondo ha lo scopo di realizzare un modello del sistema reale, selezionando le caratteristiche rilevanti ai fini dello studio.

La presente Tesi nasce da un progetto seguito dallo scrivente autore presso il dipartimento IT del CERN. Il CERN è un laboratorio internazionale che conduce ricerche nel campo della fisica delle particelle elementari. Il progetto riguarda il profiling di un nuovo gruppo di 240 server impiegati all'interno del sistema batch del CERN, con l'obiettivo di sostenere i carichi computazionali richiesti dai progetti di ricerca nel campo dell'High Energy Physics. Le risorse virtualizzate sono state testate attraverso un processo di benchmarking al fine di verificarne le metriche di performance tramite una dettagliata attività di analisi. L'approccio analitico è stato infine affiancato dalla creazione di un modello che simulasse il comportamento del sistema reale a partire dalle metriche precedentemente ricavate.

Introduction

The scientific domain is one of the main catalysts of the big data explosion. Genomic, spatial and high energy physics research centres produce a huge amount of data that must be monitored and analysed in real time everyday. In such an environment, the IT infrastructures are required to adopt new technologies to capture, store and manipulate the massive amount of data produced. For example, cloud infrastructures have been adopted to optimise the management of on-premises resources and to exploit other data centres' resources. A wide spectrum of cloud services are being adopted by numerous IT organisations, fostering further developments by private companies as well as by open-source communities such as OpenStack. In addition to the features offered by the cloud services, a crucial component in the big data scenario is represented by the service performance, in particular in the area of computing services, where the delivered performance matches the ability to execute workloads in the shortest amount of time.

This Thesis deals with the measurement of the computing performance in a virtualised environment, considering as an use-case the computing requirements of the CERN data centre. CERN, Conseil Européen pour la Recherche Nucléaire is an international scientific laboratory that conducts fundamental researches in the domain of particle physics. CERN provides an outstanding example of the big data challenges with the computing activity connected to the CERN's Large Hadron Collider (LHC). For example, the data flow from all four experiments of the CERN's LHC has been over 40 PB in 2016. In order to tackle the profiling and optimisation of compute resources, complementary types of approaches can be adopted. In this work two main approaches have been followed: the measurement-based and the model-based performance evaluation. The measurement-based evaluation consists in gathering and analysing the performance metrics of computing resources, often exploiting appropriate benchmarks applications. Benchmarks emulate realistic workloads providing measurements compatible with the computing power that an application would access at runtime. Instead, the model-based techniques rely on the simulation of a system architecture based on specific assumptions about the service. The latter technique offers a less expensive approach, since it does not require a real implementation of the system, but it could lack the same level of accuracy achieved as with the measurements approach. Therefore, in order to have a flexible and reliable performance evaluation the two techniques have been adopted in this Thesis to mutually validate the results. This Thesis describes the work done by the writer in the context of a benchmarking activity in the CERN IT department. The project is concerned with the benchmarking of a new set of 240 servers introduced in the CERN's data centre, which are used as computing resources

for the CERN batch system. The project's aim is to evaluate the performance of the computing resources when delivered through virtual machines of different sizes. A number of benchmarking workloads – already adopted by the WLCG community – has been used to collect the performance metrics. A large amount of profile data has been collected and analysed. The analysis results have been then used as an input for the model of the system. The model's goal is to simulate the real system behaviour, providing a useful instrument to verify resource performance under different configurations. This Thesis is organised as follows. Chapter 1 gives an overview of the big data challenges, with particular focus on the computing architectures used to deal with them. This Chapter also introduces the CERN computing use-case. Chapter 2 describes the CERN computing infrastructure emphasising, in particular, the cloud infrastructure as a service. Chapter 3 explains the set-up and tools adopted for the performance measurement. Chapter 4 describes in detail the measurement-based analysis performed on the resources with different types of workloads. Chapter 5 focuses on the model-based approach for the resources evaluation, comparing the results with the ones obtained by the measurements-based approach.

Chapter 1

Technological Challenges in Computing

During the last two years the term big data become one of the most fashionable arguments to talk about and to be informed about in the IT Computing field (ITC). This term has been introduced by computer scientists to describe the constant flood of data from several sources and the evolving technology to store and process them. Even if the term is relatively new, most of the present big data sources were already fertile years ago. For instance, traditional big data sources can be defined as the financial transactions of banks and the experiments in particle physics or astrophysics. Nevertheless, the technological innovations and the introduction of the media in the last decade changed the scale on how big data are used and the sources they can be gathered from. For example, the new big data sources are retail sales and the search engines that allow customization of individual searches. As science teaches, the added value of data does not come from the raw data, but from the analysis process that allows information to be extracted from them. Therefore, big data becomes not only a matter of size, but an opportunity to expand the knowledge of organisations in order to answer questions that were previously considered beyond reach. For example, the retail business can achieve greater improvements thanks to sophisticated algorithms and cross-references between transaction which target individual customer preferences. As well, manufacturers can boost quality and production while minimizing waste thanks to the comparison between sensor data and production historical data.

However, the enormous data influx and the technologies to convert these vast resources into information and knowledge is straining IT infrastructures. Therefore, the IT infrastructure is required to provide the needed computing and storage resources in order to tackle the big data challenge in a performable and reliable way.

This Chapter gives an overview on what the term big data means and in particular how knowledge can be retrieved from them and which kind of computing resources big data requires. Finally the Chapter introduces a concrete use-case of big data sources: the computing challenge of particle physics and the constant need to optimise resource performance, which is the framework where this Thesis paper fits in.

1.1 Big Data

Big data is a massive collection of data, deriving from traditional and digital sources, which represent the baseline for analysis and discoveries. While the term is very fashionable nowadays, the act of gathering and storing large amounts of data is relatively old. Until the 60s, globalization led to the development of new technological innovations and media, which significantly increased the amount of data, the speed of their creation and the range of data types and sources. Indeed, already seventy years ago, the Oxford English Dictionary attempted to quantify the growth rate in data collection under the term "information explosion". The evolutionary process in technology contributed to a continuous and quick increase of the torrent of data, introducing the concept "big data era". As a matter of fact, nowadays, every source produces data streams at an unprecedented speed since each device is equipped with an internet connection that sends the data to storage centres in real time. The latter gather them and make them always available in order to meet the challenges for business growth and development. However, the velocity is not the only constraint in the big data era, since the volume of data identifies one of the biggest stumbling blocks for the new IT infrastructures. Indeed, it has been estimated [1] that the total amount of information is growing exponentially every year, reaching a value of 40 Zettabytes in 2020. In 2001, Doug Laney's report [2] standardised the big data constraints into five big data properties, summarised in Fig. 1.1. The volume parameter identifies the quantity of generated and stored data, the variety defines the nature of data due to the various types of sources data are gathered from. The velocity identifies the rate at which data should be gathered and stored, whereas the veracity defines the uncertainty of data due to inconsistencies, biases and noise caused over time. Finally, the volatility concept determines when data is no longer relevant to the current analysis and how long it should be stored.



Figure 1.1: The 5V model defines the big data with five different measures: volume, velocity, variety, veracity and volatility [2].

1.1.1 Big Data Fields

In most business use-cases, any single source of data on its own is not useful. The real value comes from combining these streams of big data with each other and analysing them to generate new insights, which enables decisions and actions. Generally, the big data sources can be boiled down to a few varieties of data generated by machines, people, and organizations.

Data generated by machines are gathered from real time sensors in industrial machinery or vehicles, as well as the sensors that track user behaviour and environment with smart devices. Generally speaking, there are three main properties of smart devices based on what they do with sensors: they can connect to other devices or networks, they can execute services and collect data autonomously and they have some knowledge of the environment. Moreover, machines collect data 24/7 via their built-in sensors, causing them to be the largest of all the big data sources. For example, a Boeing 787 produces half a TB of data every flight, since almost every part of the plane constantly updates both the flight and the ground team about its status [3]. The dedicated real-time analysis of all the data collected sustains the monitoring and the problem detection in-site enabling real-time actions.

The other main type of big data that has been evolving in the last years, leading to a huge growth volume, are the human generated data. People are generating massive amounts of data every day through their activities on various social media like Facebook, or on-line photo sharing sites like Instagram, and video sharing websites like YouTube. As well, data are generated via blogging, internet searches, via text messages and emails. Most of this data is unstructured, that means non conforming to a well defined data model and, therefore, no relation model and no SQL are suitable. For example, Facebook deals every year with about 180 PB composed mostly of unstructured data such as the 50 billion of users' photos and videos [4]. Indeed, due to their unstructured nature, the costs and time of the process of acquiring, storing, cleaning, retrieving and processing data can add up the investments before start reaping valuable insights. For example, Amazon and Netflix use customer behaviour modelling and sentiment analysis to analyse preferences of their customers. Indeed, based on consumer behaviour, organizations suggest products to customers, and in turn have happier customers and higher profits. Whereas, sentiment analysis analyses social media to find whether people associate positively or negatively with preferences. Another application area where value comes in the form of societal impact and social welfare is disaster management. Data in the form of pictures and tweets, helps facilitate a collective response to disaster situations, such as evacuations through the safest route based on community feedback through social media. For example, the International Network of Crisis Mappers [5], is the largest of such networks and use big data in the form of aerial and satellite imagery, participatory maps and live Twitter updates to analyse the data using geospatial platforms, advanced visualization, live simulation and computational and statistical models. However, the daily generation of the huge amount of unstructured data makes the traditional hardware and software unable to deal efficiently. For this reason, new set of techniques and technologies are required in order to be able to capture, create, store, manipulate and manage

the massive scale dataset. For example, a mixture of long-standing techniques such as graph and statistical analysis, new high-performance analytic tools such as MapReduce/Hadoop [6], knowledge discovery algorithms and machine learning techniques are required to extract real-time value from data in a faster and more intelligent way.

The last type of big data are data generated by organizations, which refers to more traditional types of data unique with respect to organization and context. Each organization has distinct operation practices and business models for which almost every event can be potentially stored. For example, commercial transactions, government institutions, e-commerce, banking, stock records and medical records. Big data becomes an opportunity for improving performance and efficiency of the organisation, as well as to create strategic values in traditional segments and to create opportunities to expand products and service offerings. In fact, big data allows better models to be built, produces innovative approaches in market strategies, defines how humans behave and how physical events evolve. Moreover, they can be used for pattern recognition to detect correlated products and to estimate the demand for products likely to go up in sales. For example, Amazon and Netflix, through recommendation engines, keep track of browsed articles to personalize the categories that could interest the customer and to predict best match products. Therefore, this process allows for personalized marketing. Research in the healthcare field is also enabling the development of methods used to analyse large scale data. This in turn develops solutions which are tailored to each individual. These methods should prove to be more effective. In particular, genomic data analysis is one of the largest growing big data, being that between 100 million and 2 billion human genomes could be sequenced by the year 2025. This sequenced data has a demand of dozens of exabytes in data storage, which could take up to 10 000 trillion CPU hours in analysis.

1.2 Big Data Computing Resources

The data produced from several scientific activities and business transactions require tools to facilitate efficient data management and analysis while preserving the intrinsic value of the data. Therefore, the infrastructure needed for analysing big data must be able to hold deeper analysis such as statistical analysis and data mining, to scale the excessive data volumes, to deliver faster response times and to automate decisions based on analytical models. Knowledge discovery and decision making from such rapidly growing voluminous data is a challenging task in terms of data organization and processing. Consequently, data centres are facing challenges caused by an exponential growth in data traffic, while having to contain ever-increasing costs and comply with environmental regulations. The increasing requirements is causing data center managers to look for new ways to reduce power consumption while maintaining the same output. They must also demonstrate greater performance and store larger amounts of data without increasing the physical space used. In order to cope with the required technology improvements, IT organizations have reorganized their infrastructures designing sophisticated computing architectures to tackle these extraordinary computing challenges. The goal is to optimize the data center

resources, as well as to maximize the usage of other datacenter resources thanks to technologies such as grid computing (see Section 1.2.1) and cloud computing (see Section 1.2.2). These mentioned technologies have access to huge amounts of computing power by summing up resources and offering a single system view. These technologies enable large-scale and complex computing, and have revolutionized the way computing infrastructure is abstracted and used.

1.2.1 Grid Computing

Data grids provide an infrastructure to support data storage, data discovery, data handling, data publication, and data manipulation of large volumes of data presently stored in various heterogeneous databases and file systems. The computing grid [7] is defined as a *"a large-scale geographically distributed hardware and software infra-structure composed of heterogeneous networked resources owned and shared by multiple administrative organizations"*. It consists of a distributed computing infrastructure based on individually managed and network-connected computer resources perceived as a single entity by the end user. For this reason the concept of Virtual Organization (VO) [8] is introduced as the shared space between different partners whose resources are managed in a transparent way with respect to the end user, independently from his geographical location. From the end users' point of view, the grid is a virtual powerful device based on coordinated resources supplied by different partners. In order to be able to take advantage of these services in a certain context the user is only required to have a set of credentials available to the VO that authorise him. The Grid concept has been introduced in the 90s, when researchers, inspired by the availability of high-speed wide area networks and challenged by the new computational requirements, came up with a computing infrastructure that provided access on demand and permitted coordinated resource sharing among dynamic institutions and individuals. It is employed in several areas such as, distributed supercomputing, high-throughput computing, on-demand computing, data-intensive computing and collaborative computing. There are several essential key elements that have contributed to the development of the grid. Its distributed nature and its site independence allow computing centres to join or to leave, with minimal impact over other domains. Moreover, the services can cross the administrative boundaries of the institutions, appearing as a single service to the users. The multi-domain nature of the grid implies that administrative authorities may add, remove or change rights of the VO's users. Finally, the early adoption of standard interfaces and open-source technologies allows for the contribution from a variety of development groups. In such a large distributed system, small problems can be magnified and affect many other things. For this reason, high standards of reliability and availability, performance, and manageability are required. Indeed, the infrastructure never stops, and therefore upgrades and maintenance must be performed during production running. Of course, individual services must undergo some periods of interruption, but usually redundancy is required to avoid the creation of problems. As a disadvantage, since sites are autonomous, the resources and services running may be very different from one another. The heterogeneity of the resources originates from the geographical location, history of the site, available know-how and need for compatibility with other applications at the site.

1.2.2 Cloud Computing

The other key player in technology serving as a data intensive platform for information mining and knowledge discovery over a elastically scalable infrastructure is cloud computing. This emerging trend is known as Big Data Cloud. It is a new data science paradigm which combines large scale computing, storage and software applications for new services over private or public networks based on pay-as-you-go delivery models. For this reason, cloud based approaches are seen as agile and low capital intensive solutions. The features of Big Data Clouds that allow its development are several. The possibility to benefit for a wide range of computing facilities with transparent access to scalable storage repositories and data services. The platform supports both traditional and emerging data intensive computing models as well as scalable deployment and execution of applications. Moreover, the infrastructure ensures a high performance data computation and high availability of computing resources and data. Among the several definitions of cloud computing found in literature [9], is a pool of virtualised resources that host a variety of different workloads and allows them to be deployed and scaled-out through the rapid provisioning of virtual or physical machines; it supports redundant, self-recovering, highly scalable programming models and the real time monitoring of resources usage to enable rebalancing of allocations when needed. Following the flow of the previous sentence it is possible to deepen some key points on which cloud computing is based on. The cloud relies on the resource pooling concept, which means that different kinds of computing and storage services can be offered to the end user, who is uninterested in resource details because they are managed by the provider itself. Moreover, the customer pays the provider on a consumption basis for resources that have been used by him, which have been autonomously asked for when required. Another strength of the cloud is that the resources could be dynamically configured and delivered on demand according to customer needs whereas the cloud providers try to limit the under-utilisation of resources, thanks to virtualisation, in order to exploit economies of scales. Moreover, the pool of computing and storage resources must be accessed by the end user through standard protocols via an abstract interface, which implies the need of a mechanism for users' authentication. Virtualisation is the concept that represents the basis of cloud computing. It provides the necessary abstraction to unify the underlying computing, storage and networking hardware resources as a pool of shared virtual facilities that can run simultaneously for the different users' applications. However, cloud computing is not the same thing as virtualisation, rather, it's something that can be achieved through virtualisation.

For a proper categorization of the cloud computing architecture, three models must be analysed (Fig. 1.2): the service model, the deployment model and the isolation model.

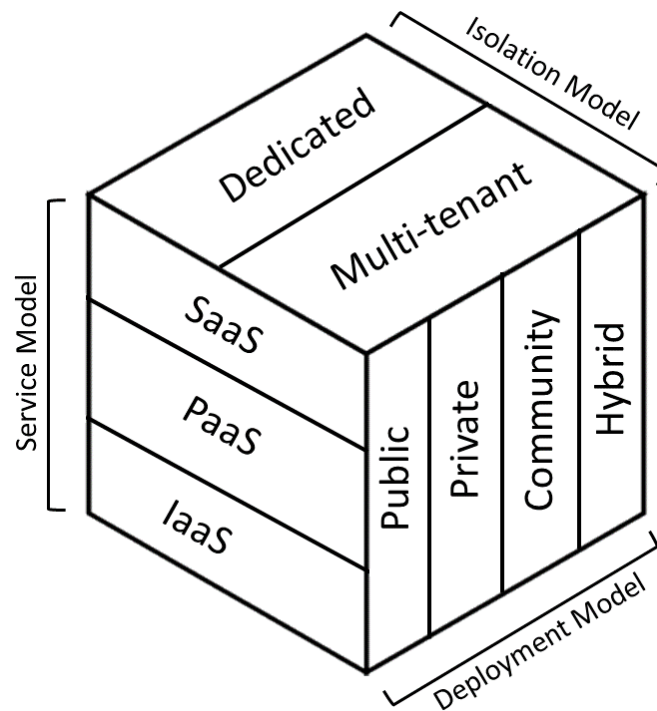


Figure 1.2: The three cloud dimensions [10].

Service Level of Clouds

The stack components shown in Fig. 1.3 categorise cloud computing in three service levels, and give insights about what type of resources the cloud can provide.

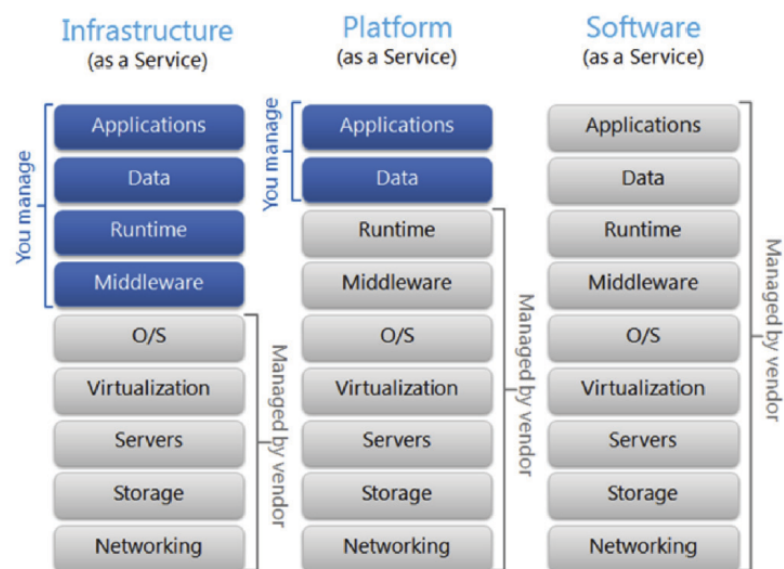


Figure 1.3: The three cloud service levels are, from left to right, IaaS, PaaS and SaaS. Users' level of delegation increases from left to right [10].

The stack on the left identifies the Infrastructure as a Service (IaaS) cloud. It provides the user resources provisioned in terms of infrastructure. The latter is

composed of a set of physical or virtual machines with their related operating system, physical or virtual storage space and network connectivity between them. In this kind of infrastructure, the user retains full responsibility in installing, maintaining and operating the services. In the center the Platform as a Service (PaaS) cloud is depicted, which provides a high level integrated environment to build, test and deploy custom applications. The customer has the responsibility of connecting his own data and applications to the cloud provider services through APIs. An example of this is the Google's App Engine [11]. The last stack defines the Software as a Service (SaaS) cloud; it allows users to benefit, without management responsibilities, from ready-to-use applications which make use of the underlying layers in a transparent way with respect to the user. An example of this are Microsoft's Office Web Apps [12] or Google Drive that deliver browser-based versions of, for instance, word processors, spreadsheets.

Deployment Models

The previously defined service models can be divided into multiple deployment models. The private cloud is an advantageous solution when services are procured and provisioned for the exclusive use of the organisation. Private cloud services draw their resources from a distinct pool of physical computers accessed across secure connections via public networks. For this reason, it can recall the traditional model of individual local access networks used in the past by enterprises but with the added advantages of virtualisation. Its benefits are high security and privacy in storing and processing sensitive data and a high level of control, since it is only accessible by a single organisation. However, the latter property partially removes the economies of scale generated in public clouds by having centralised management of large scale devices. On the contrary, when the service is offered to a public with different types of goals, the term public cloud is used. The cloud resources are available on demand from a vast pools of resources, which therefore can respond seamlessly to fluctuations in activity. However, the service is cost effective since it can benefit from the largest economies of scale and the pay-as-you-go charging model. The spread of public clouds is evident in the myriad of IaaS, PaaS and SaaS services available on the market accessible as a service from any internet enabled device. An in the middle solution between the private and the public models is the community cloud, in which services are procured and provisioned for a community that shares common goals and regulations. For example, a scientific community might be spread across several countries, and have the single objective to analyse some data in a coordinated way. Finally, the term hybrid cloud is used to identify a combination of two or more of the models described above. For example, private clouds can even be integrated with public cloud services where non-sensitive functions are always allocated to the public cloud in order to maximise the efficiency on offer.

Isolation Model

The last described category is the isolation model, which defines how cloud resources are isolated; it is a critical issue in contexts such as disaster recovery and the security

of data and applications. If resources are served and isolated only for the users' need, the infrastructure can be defined as dedicated infrastructures; while a multi-tenant infrastructure is modelled when several customers can share the same physical resources in a transparent way.

1.2.3 Similarities and Differences between Cloud and Grid

Cloud computing and grid computing are prime examples of distributed computing architectures, a concept introduced when computing elements of a network are spread over a large geographical area. They have in common the property of abstraction of the computing environment, which masks complex processes in a system and presents the user a simplified interface to easily interact with. Both are used to economise computing by maximising the existing resources thanks to the multitasking behaviour of the physical resources, meaning that many customers can perform different tasks on the same resource at the same time. Both provide Service Level Agreements (SLA) to guarantee full uptime availability (above 99.9%). However, between the grid and cloud, focusing on the IaaS, there are some differences [13] among the properties introduced in the previous sections. The grid business model is project oriented, which means that the project community has a certain number of service units they can spend, whereas for the cloud architecture the customer will pay the provider on a consumption basis (see Section 1.2.2). In the grid model resources are heterogeneous and dynamic since they depend on the VO's infrastructure. This implies different usage policies, different hardware and software configurations, and also different availability and capacity properties. Cloud has both homogeneous and heterogeneous resources. The first case applies where the resources are provided by one vendor. On the contrary, an heterogeneous cloud integrates components of many different vendors, even at different levels. Virtualisation is characteristic of both the grid and cloud model. For the first, virtualisation applies to the level of data and computing resources (application layer), whereas for the cloud model, virtualisation applies to hardware and software platforms (see Section 1.2.2). In the grid model, the processing power is typically used to compute one single problem on a huge data set. The job itself is controlled by one main computer and is broken down into multiple tasks which can be executed simultaneously on different machines. In this way, unused processing power is effectively used by maximizing available resources and the time taken to complete the large job is significantly reduced. The cloud controls applications in a similar way with respect to the grid, being able to distribute and organize them on different virtual machines. In addition, in order to balance the workload, it can dynamically increase or reduce the cluster dimension thanks to the re-provisioning or deleting of the virtual machines, as well as resizing them in terms of associated computational resources. For the grid, the architecture is service oriented since it depends on the vendor from which the service is hosted, whereas for the cloud the architecture typically consists of a front end platform, a back end platforms, a cloud based delivery and a network.

1.3 The CERN Big Data Use Case

Beyond the business oriented applications, one of the main catalysts of the big data explosion is the scientific domain. For example, high energy physics and astronomical experiments, earth observation systems, weather forecasting and monitoring of natural disasters produce a huge amount of data. The volume of data and the complex algorithms needed to analyse them require expanded computing resources. An outstanding example of big data in particle physics is the computing activity connected to the CERN's Large Hadron Collider (LHC). The LHC is the world's biggest particle accelerator and its related experiments, with over 150 million sensors in their detectors, enable the gathering of about 1 Petabyte of data per second.

CERN [14], Conseil Européen pour la Recherche Nucléaire is the biggest international scientific laboratory, involving 22 member states¹. Founded in 1954, CERN's mission is to conduct fundamental research in the domain of particle physics. Alongside the studies on fundamental science, CERN pillars involve practical applications, such as technology innovation and research in engineering and computer science, the best known examples are the World Wide Web and the hadron therapy. Moreover, CERN emphasises the role of science in education thanks to its multicultural and stimulating environment. CERN has made major contributions in the understanding of the behaviour of fundamental particles in nature and their interactions. To study the basic constituents of matter, physicists and engineers at CERN have adopted the world's largest and most complex scientific instruments in the field: accelerators and detectors.

Accelerators can be considered gigantic microscopes used to study the structure of the matter, and as time machines used to reproduce the conditions of the universe one millionth of a second after the Big Bang. The purpose of an accelerator is to increase the momentum of a beam of particles. The CERN's Large Hadron Collider consists of a 27-kilometre ring of superconducting magnets crossing two countries, France and Switzerland, 100 m underground. Inside the accelerator, two high-energy proton (or lead) beams travel in opposite direction close to the speed of light at an energy of about 7 TeV. The accelerator complex at CERN (Fig. 1.4) is a succession of machines that boosts the energy of beams of particles injecting them into the next machine in a sequence of increasing higher energies, until the value of 7 TeV is reached in the LHC, the last element in this chain.

Protons in the LHC circulate around the ring in well-defined bunches; under nominal operating conditions there are 2 808 proton bunches, each of which contains about 10^{11} protons. The LHC collides the two beams at four interaction points, where detectors are placed. The purpose of these detectors is to study the phenomena generated during the collision by identifying secondary particles produced in collisions, measuring their charge, momentum and energy. The detectors are installed in four huge underground caverns (Fig. 1.5) built around the four collision points of the LHC.

¹ Austria, Belgium, Bulgaria, Czech Republic, Denmark, Finland, France, Germany, Greece, Hungary, Israel, Italy, Netherlands, Norway, Poland, Portugal, Romania, Slovak Republic, Spain, Sweden, Switzerland, United Kingdom.

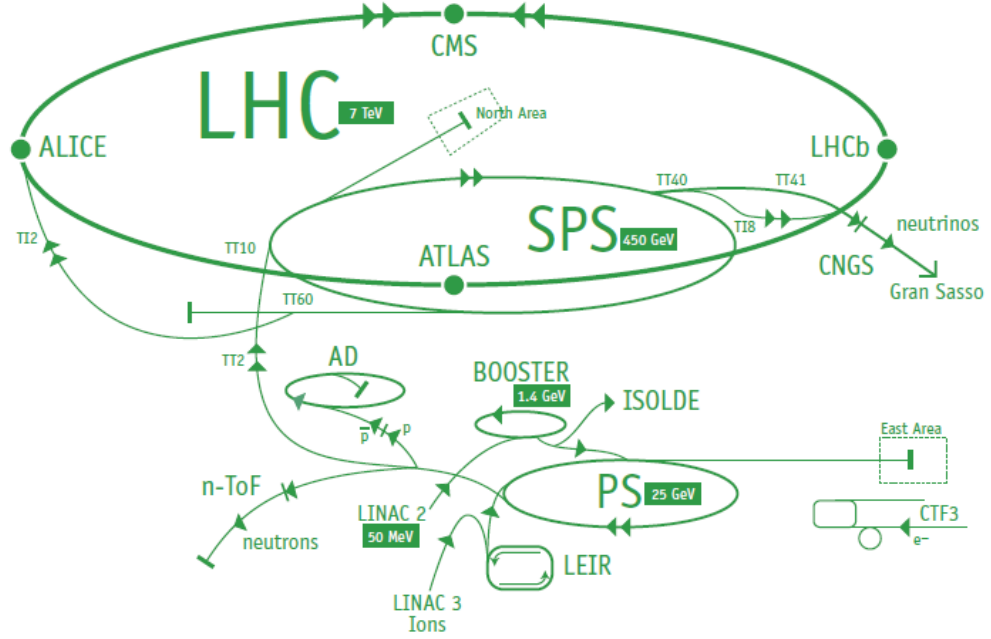


Figure 1.4: Protons are injected into the Booster at an energy of 50 MeV from Linac2; the booster accelerates them to 1.4 GeV. The beam is then fed to the Proton Synchrotron (PS) where it is accelerated to 25 GeV. Protons are then sent to the Super Proton Synchrotron (SPS) to reach an energy of 450 GeV. They are finally transferred to the LHC both in a clockwise and an anticlockwise direction. It takes 4 minutes and 20 seconds to fill each LHC ring, and 20 minutes for the protons to reach their maximum energy of 7 TeV [15].

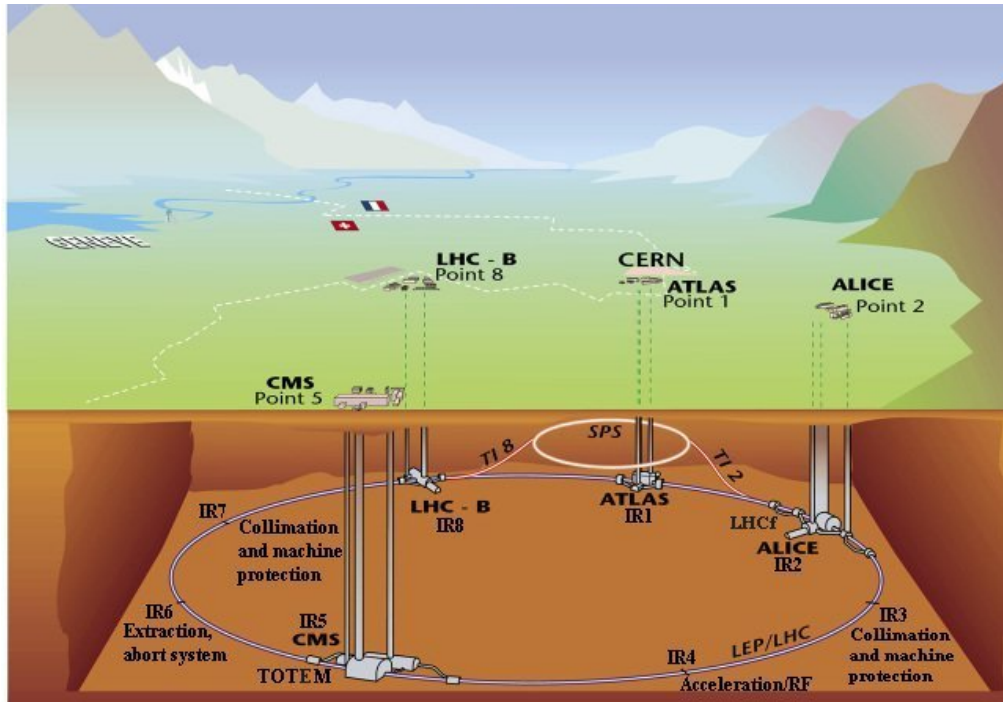
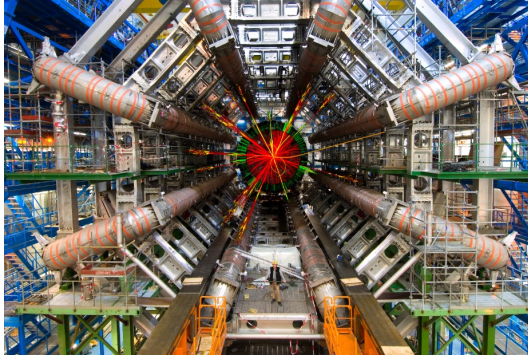
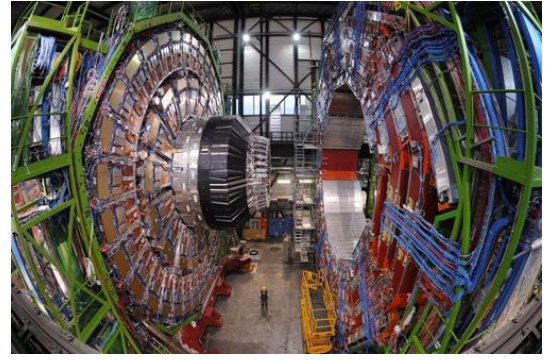


Figure 1.5: Overall view of the LHC experiment [16].

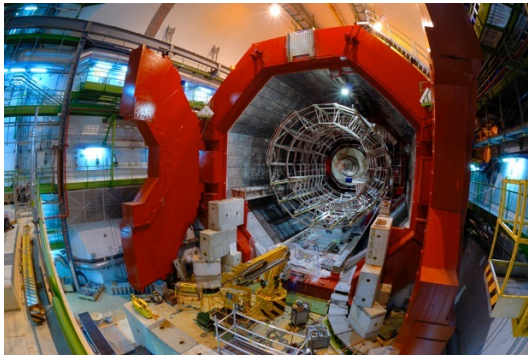
Each detector is the device of a LHC experiment, namely: A Large Ion Collider Experiment (ALICE) [17], A Toroidal LHC Apparatus (ATLAS) [18], Compact Muon Solenoid (CMS) [19], Large Hadron Collider beauty (LHCb) [20]. ATLAS (Fig. 1.6a) and CMS (Fig. 1.6b) detectors are large general-purpose detectors used to cover the widest possible range of physics research, from the Higgs boson to supersymmetry (SUSY). The main characteristic of ATLAS is its enormous doughnut-shaped magnet system, which consists of eight 25 m long superconducting magnet coils, arranged to form a cylinder around the beam pipe through the centre of the detector. CMS has the same physics goals as ATLAS, but a different technical design. It is built around a huge superconducting solenoid. This takes the form of a cylindrical coil of superconducting cables that will generate a magnetic field of 4 T, about 100 000 times that of the Earth. The other two detectors have more specialized research goals. The ALICE detector (Fig. 1.6c) is designed specifically to study heavy-ion collisions, when the LHC accelerates lead ions in place of protons. This experiment allows the reproduction of states of matter close to those of the universe very early on. The fourth detector, LHCb (Fig. 1.6d), is designed to study the asymmetries of matter and anti-matter.



(a) ATLAS detector [21].



(b) CMS detector [22].



(c) ALICE detector [23].



(d) LHCb detector [24].

Figure 1.6: Detectors of the LHC experiment.

1.3.1 Computing Needs for the LHC Experiments

In each detector the proton-proton collisions are recorded in the form of discrete events at a rate of 40 MHz². In addition, due to the high density of proton beams, many interactions overlap at the same time. The size of data for each collision event is about 1-2 MB in each of them, but given the high event rate, the full detector data rate approaches 1 PB/s [25]. Figure 1.7 gives an idea of the complexity of each recorded event.

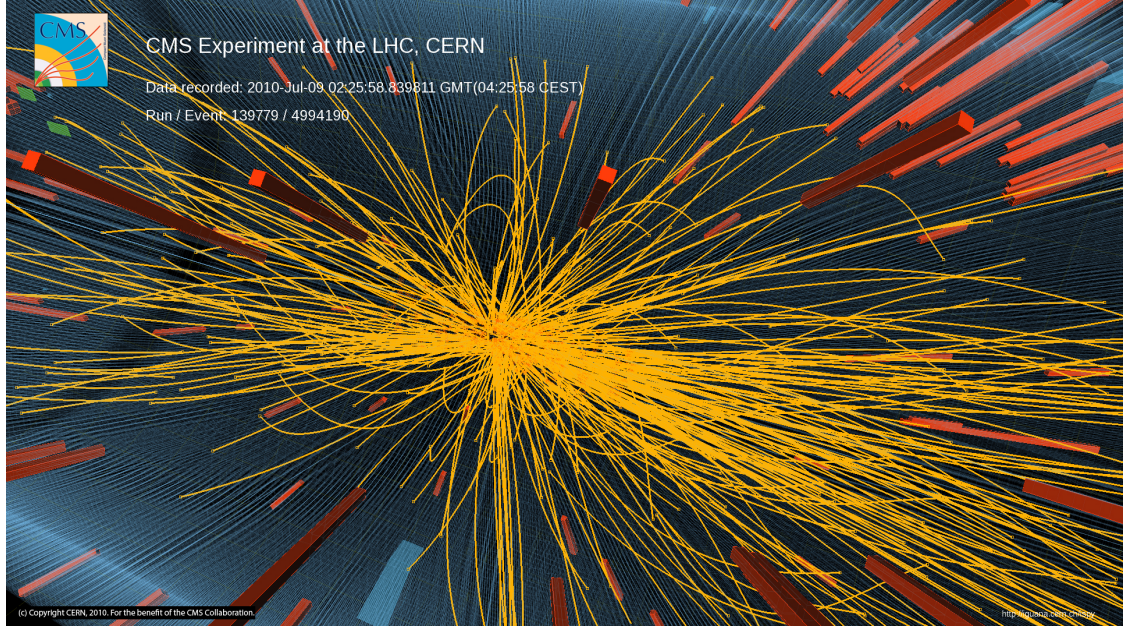


Figure 1.7: A 7 TeV proton-proton collision in CMS, event reconstructed consisting of more than 100 secondary charged particles [26].

Due to the data rate and size, not all of this data can be fully processed and recorded. Indeed, not all of them are of interest for further analysis. For this reason, interesting events are selected thanks firstly to dedicated electronics and hardware triggers, secondly, to large computer clusters located close to the detector (High Level Trigger) that make real-time decisions regarding the data that should be retained. This approach results in diminishing the data rates at a manageable level that allows data to be written on persistent storage (disks and tapes) for later analysis. Table 1.1 summarizes the data rates for the LHC detectors updated to 2016.

ATLAS	CMS	ALICE	LHCb
1 GB/s	1 GB/s	5 GB/s (heavy-ion data taking)	200 MB/s

Table 1.1: Data rates for LHC detectors during the 2016 [25].

The main computing challenge for the LHC experiments is the processing and archiving of the huge volume of detectors' data, as well as of the simulated data.

² Except for the LHCb experiment, which rate is of about 3 kHz

Nowadays, the LHC computing needs add up to approximately 200 000 of the fastest CPUs and a storage space of about 45 PB per year of LHC operations. CERN provides 20% of the overall requirements. Already in 2001, in order to avoid significant investments in hardware, power and cooling infrastructure, it seemed reasonable to build a globally distributed computing environment with the help of the member states. In that way the needed scale has been achieved integrating computing resources across multiple administrative domains and enabling an authentication and authorisation infrastructure to allow worldwide scientists to access resources spread around the world. Therefore, in 2008, a long-term infrastructure for managing LHC computing has been created as a global collaboration of more than 170 computing centres in 42 countries. This infrastructure is known as the Worldwide LHC Computing Grid (WLCG).

1.3.2 Worldwide LHC Computing Grid Infrastructure

The WLCG [27] has a hierarchical infrastructure based on tiers and computer centres (Fig. 1.8).

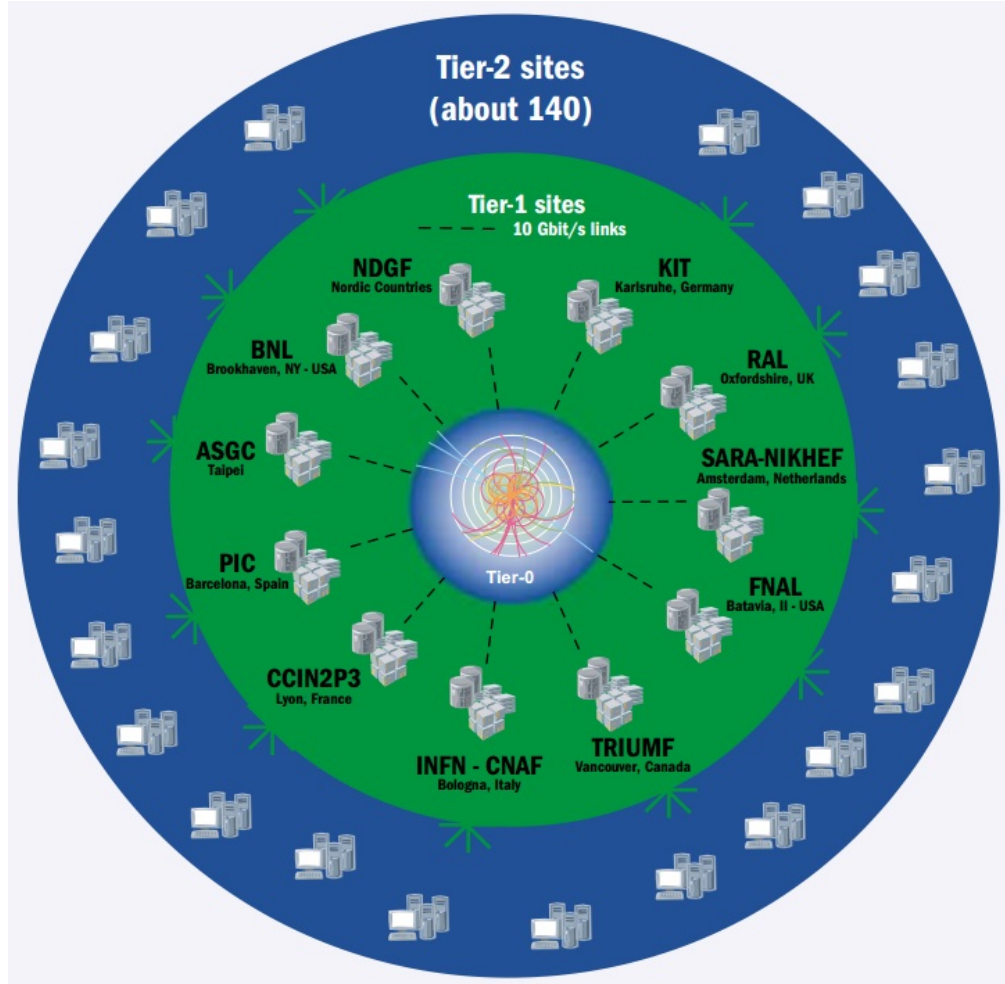


Figure 1.8: Diagram showing the tier system of the Worldwide LHC Computing Grid project [28].

Tier-0, which is the coordinator of the overall grid, is located at CERN and provides about 20% of computing power. Its main role is to accept the data streaming from the experiments at the full data rates providing large computing clusters; the 50 000 CPU cores close to the experiments are used for first-pass reconstructions, such as detector calibration and alignment, and utilised as data-intensive analysis facilities. Due to its fundamental tasks it must be always available during the accelerators run. Moreover, Tier-0 must ensure data redundancy through the dispatching and storage of raw data among Tier-1 centres thanks to a dedicated network of multiple 10 Gb/s per connected site. Tier-1 is composed of approximately 13 computer centres distributed between Europe, USA and Canada, providing about 40% of the computing power. It plays a fundamental role in providing data intensive analysis facilities as well as in the long term storage of raw and simulated data from Tier-0. Moreover, it controls each Tier-2's centres assigned to it, in terms of providing them with the data and the support in case of operational problems. Finally, Tier-2 is composed of approximately 160 computer centres, including universities and institutions, and provides about 40% of the computing power. Its goal is to provide computational resources for an entire experiment and to send back the simulation results to Tier-1 that takes care of the archiving processes. Table 1.2 summarises the amount of resources exposed by the Worldwide LHC Computing Grid between the Tiers' level.

Resources	Estimated quantity
Computing cores	$\sim 600\,000$
Disk and tape storage	$\sim 700\text{ PB}$
Processed job/day	$> 2\text{ million}$

Table 1.2: Amount of resources of the WLCG project distributed over the Tiers [29].

The connection between the resources is a critical factor for the success of the WLCG infrastructure. Since the data volumes and the ability to ensure global access to data are primary concepts, the *Optical Private Network* (LHCOPN) connection between Tier-0 and Tier-1 is based on dedicated and redundant links whose bandwidth is around 10 Gb/s. On the other hand, the connection between Tier-1 and Tier-2 sites relies on *National Research and Educational Network* (NRENs) coordinated by GEANT [30] in Europe and ESNET [31] in USA. Another crucial component for the CERN grid infrastructure is the authentication and authorisation layer. The latter is controlled by a two-part process thanks to a network of trust. To provide authentication, the users present a certificate that identifies and recognises them as members of a valid virtual organization. Having a comprehensive policy has been essential to deploy an infrastructure across different countries, allowing the grid to work within the requirements of privacy and security.

The collaboration is managed between the countries by a *Memorandum of Understandings* (MoU) that defines the level of robustness, fault tolerance and supportability to be provided by the sites on each level of the grid as well as the mechanism to define the computing and storage resources provided by each site. In order to measure grid-site reliability and availability, the Site Availability Monitoring system

(SAM) has been introduced executing grid jobs in each site with the aim to test the overall service. The results of these tests are used to send alarms to the site in order to report a possible problem. Nowadays the majority of sites are extremely reliable, and most of the problems arise from hardware problems such as power cooling failures and equipment failures. Another key component is how the storage resources are managed. In the CERN grid, a Storage Element provides uniform access through different protocols and interfaces to data storage resources consisting of disk servers, large disk arrays or tape-based mass storage systems. Then, for what concerns the data management, in the grid environment files can have replicas at many different sites and, when created, files are no longer modified. The Data Management services are used to locate and access them. Finally, job submission is the role of the Workload Management System, which accepts user jobs assigning them to the most appropriate computing element and recording their status as well as retrieving their output.

1.3.3 From WLCG to Cloud Resources

During the second decade of this century, the rise of Cloud Computing (see Section 1.2.2) has challenged many assumptions at the basis of the grid computing model, such as the complex installation and maintenance of the sites as well as the adoption of complex and not user-friendly interfaces. In fact, clouds can be considered the new generation of the grid computing model: both need to be able to manage large facilities and to define methods useful to the consumers to discover, request and use the resources provided by the central provider. Cloud management in grid sites gives the way, using virtualisation, to improve the cluster management and efficiency without the need to maintain special grid software with its associated costs. Moreover, the cloud gives the possibility to access resources on-demand from commercial providers. In recent years, CERN introduced a private and multi-tenant cloud infrastructure as a IaaS model. This topic is the subject of the next chapter.

Chapter 2

CERN Computing Infrastructure

In this Chapter a brief introduction about the evolution of the CERN data centre is given, with particular focus on its development toward a cloud-based model. The virtualisation concept as well as the OpenStack infrastructure are introduced and described.

2.1 Short Overview of Data Centre History

Computing infrastructures, also known as data centres, play a critical role in the big data business providing the computational power to process and maintain data. The technology surrounding data centres is evolving rapidly, in terms of conception, configuration, and utilization. In 1950s, historical sources [32] associate the birth of the computing infrastructure concept in the U.S. Army that developed incomparable storage for that time, computation abled machine called ENIAC used to store artillery firing codes. Even if this kind of computational resource was mainly available for government and military purposes, during the 1960s IBM [33] introduced a substantial jump in computing ability thanks to the usage of mainframes as commercial systems. The performance increase gave rise to the need to partition the hardware into smaller units to enable the best possible resource utilization. In 1982, when mainframes were extremely expensive to use and required enormous resources in terms of space and cooling, IBM [34] introduced the first PC with significant benefits in space, air-cooling and costs. In the 90s microcomputers started to move into the old mainframe rooms, consisting of thousands of servers, with networking equipment giving rise to the client-server computing model. Up to that point, the main limits to the data centres' capacity were driven by the infrastructure costs, such as buildings, cooling and electricity.

For this reason, around 2010, the "IT consolidation", which is an organization's strategy to reduce IT assets by using more efficient technologies, moved the focus of data centres towards virtual technology. It enabled the creation of a central and more flexible computing, network and storage pool of resources that could be dynamically reallocated based on the needs of several formerly siloed data centres. In particular, since IT continue to evolve toward on-demand services, many data centres moved toward cloud-based infrastructures (see Section [1.2.2](#)).

2.2 CERN Data Centres

The development of the data centres described in the previous section obviously also concerns the CERN organisation, whose requirements of computing power go growth hand in hand with the computing technology evolution. Historically [35], the mainframe era at CERN started in the late 50s, when the first mainframe, Ferranti Mercury, had been installed. It was one million times slower than today's large computers, with a clock cycle of 60 μ s. Then, in the 60s, the arrival of several IBM mainframes introduced the need of grouping them into a unique data centre, which had been relocated at the CERN Meyrin site in Geneva. In the early 90s, CERN migrated from mainframes to UNIX [36] based workstations, which had later been replaced by commodity PCs providing better performance with less costs. In the following years, the increasing volume of data produced by the LHC experiments and the necessity of processing them led to the improvement of the existing storage and computing infrastructure.



Figure 2.1: The CERN data centre houses servers and data storage systems for Tier-0 of the WLCG and for systems needed in the daily functioning of the laboratory [37].

Indeed, nowadays, as described in [38], CERN has two data centres. The main one is still at the Meyrin site (Fig. 2.1) and represents the core of the scientific, administrative and computing infrastructure. The second one identifies a support data centre at the Wigner Institute in Budapest that consists of a total of 56 000 compute cores, boosting CERN's processing capacity of about the 70%. The two data centres are 1 800 km apart, linked by two independent 100 Gb/s fibre optic lines with a network latency of about 25 milliseconds. Table 2.1 summarises

the resources in the two data centres. The data centres have many different roles such as storing and processing gigabytes of data received every second from the LHC's experiments, supporting different types of user services such as emails and desktop services as well as supplying the computing infrastructure to physicists and engineers inside and outside CERN. Nowadays, these duties are supported by different technologies in regard to the goal. For example, the LHC data storage is focused on three areas, speed, capacity and reliability. Data archives rely on magnetic tapes as a long-term storage medium, since tapes are a robust storage material that can store huge volumes of data with few costs in terms of space and electricity consumption, moreover, it is durable for long-term storage. At the state of art, CERN has about 140 Petabytes of stored data in tapes among previous experiment and new data. However, due to the fact that accessing tape is relatively slow with respect to the needs of physics processing, data is also made available on EOS disk pool system [39] in order to have the fastest access time. It is a system optimized for fast analysis access by many concurrent users with an availability of 99.5%. Moreover, EOS provides a highly-scalable hierarchical implementation, since, in addition to the usual replication approach, users can have the option to distribute the data across multiple disks ensuring the content will not be lost even if multiple disks fail.

Meyrin		Wigner	
Resource	Quantity	Resource	Quantity
Processors	16 600	Processors	7 000
Servers	9 000	Servers	3 500
Disk space (TB)	144 596	Disk space (TB)	97 282
Memory capacity (TB)	615	Memory capacity (TB)	221

Table 2.1: Amount of resources in the Meyrin and Wigner data centres at the state of art [40].

2.2.1 Evolution of the CERN Data Centres

Since 2011, the Meyrin data center was no longer able to cope with the over abundance of resources so, in 2013, CERN developed a new computing infrastructure for the Tier-0 of the WLCG, involving the procurement of a hosted data centre used to extend capacity. The winning bid went to the Wigner data centre in Budapest in order to increase the computing capacity without raising the man-power involved in operation. A reconsideration of management approach was done. Since that time CERN computing infrastructure was no longer unique, so, IT engineers investigated modern and widely used procedures developed by other large scales computing centres such as Google and Amazon in order to deal with the new concept of virtualisation and cloud. The cloud infrastructure has been introduced in order to improve how servers are managed and used, as well as to improve response time to users.

The OpenStack toolkit (see Section 2.4) has been adopted for the cloud infrastructure management [25] since summer 2013. The CERN private cloud is achieving annual

increase in the number of virtual machines, users and projects. It is a single system that scales across the two data centres in terms of compute nodes, with the master controller being at the Meyrin site.

2.3 Virtualisation Components

Virtualisation is the process of running more applications or operating systems on one physical server (host), living in different software containers (Fig. 2.2) called *Virtual Machines* (VM) which are completely isolated from each other. Storage and networking are managed together and delivered dynamically to each VM through a software layer called *hypervisor* that guarantees the isolation property.

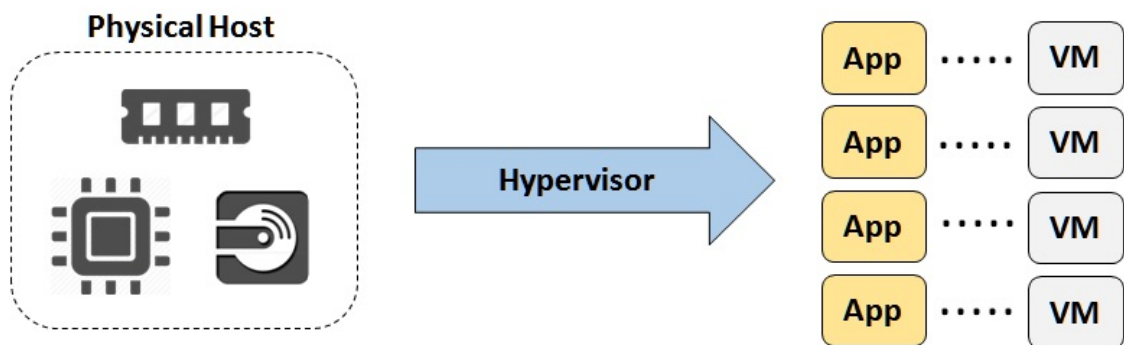


Figure 2.2: The virtualisation concept.

Virtualisation can be performed on hardware platforms, operating systems, storage devices, and computer network resources. Operating system-level virtualization deals with hosting multiple virtualized environments within a single operating system instance. Storage virtualization is the process of completely abstracting logical storage from physical storage. Finally, network virtualization defines the creation of a virtualized network addressing space within or across network subnets. For the purpose of this thesis, only hardware virtualisation is described. Hardware virtualisation consists of a host machine, the hardware on which the virtual machines are running, and a guest machine, representing the software that runs inside the virtual machine.

2.3.1 Virtual Machine

The key concept of the hardware virtualisation process [41] is the VM, which identifies a virtual environment that typically emulates the behaviour of a physical machine and in which applications can be executed. The focal elements of the virtual machine's concept are partitioning, isolation and encapsulation. The first one enables multiple operating systems to run on a physical machine while sharing the system resources between VMs. The second property defines that VMs can run independently and simultaneously on the same host and among multiple users with several separate operating environments. Finally, the encapsulation allows every VM to save its state. Each virtual machine is activated dynamically on demand

thanks to a thin layer of software called hypervisor (see Section 2.3.2) or Virtual Machine Manager (VMM), which decouples the virtual machines from the physical host.

2.3.2 Hypervisor

Hypervisor is a software component that dynamically maps virtual resources to physical resources. In literature there are two types of hypervisors: the native hypervisor and the hosted hypervisor.

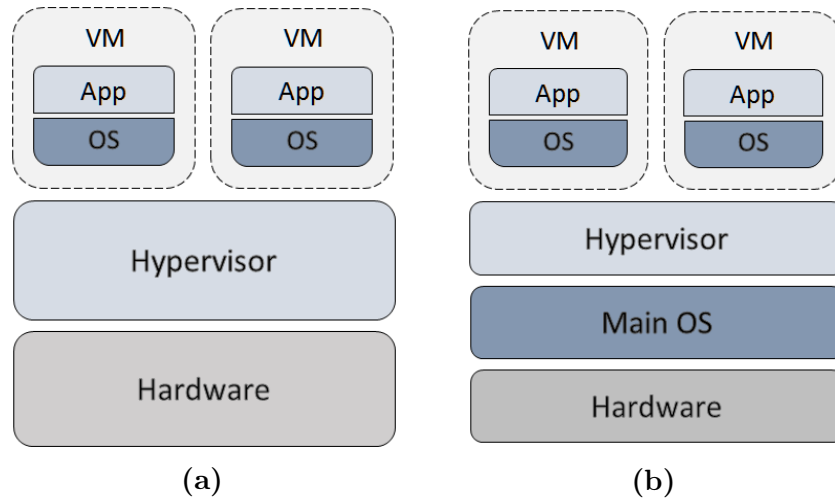


Figure 2.3: Graphical representation of the two types of hypervisors: native hypervisor (a) and hosted hypervisor (b).

The native hypervisor, also called bare-metal hypervisor or embedded hypervisor, is a thin layer between hardware and software components that exposes hardware resources to virtual machines and directly controls the guest operating system reducing the running overhead (Fig. 2.3a). Examples of modern native hypervisors are: Microsoft HyperV [42], the Citrix XenServer [43], Oracle VM Server for SPARC [44], Oracle VM Server for x86 [45] and VMware ESX/ESXi [46]. The second type of hypervisor (Fig. 2.3b) is called hosted hypervisor. It performs software virtualisation, running the guest operating systems as a process on the host operating system. Examples of hosted hypervisors are: VMware Workstation [47], VMware Player [48], VirtualBox [49] and QEMU [50]. The QEMU hypervisor, which is a machine hardware emulator, allows for the emulation of the node's hardware resources in multiple VM's instances to run on the same physical host. Its role is to intercept the instructions meant for the virtual CPUs of the virtual machine and to use the host operating system to get those instructions executed on to the physical CPU. However, this translation has a performance overhead. In order to minimise it, modern processors support a virtualisation extension. This technology allows a slice of physical CPUs to be directly mapped on the virtual CPU, boosting the performance of the virtual machines. QEMU can use KVM [51] for hardware acceleration. KVM is a Linux kernel module that enables the mapping of physical CPUs to virtual CPUs.

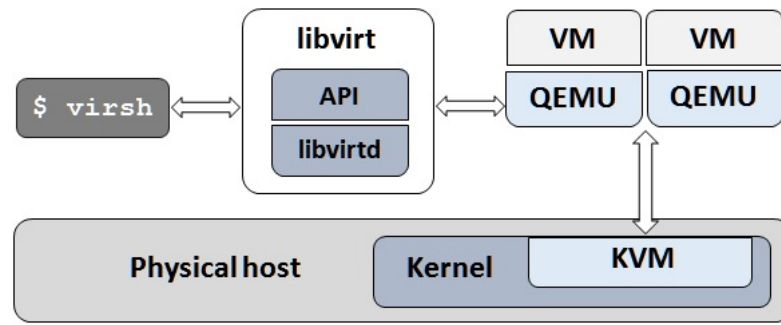


Figure 2.4: Relationship between the hypervisor components.

The combination of the two components becomes a native hypervisor. Figure 2.4 shows how the main hypervisor components are related. The controller for both QEMU/KVM and the virtual machines is libvirt, which consists of an API library, a daemon (libvirtd) and a command line interface. The latter allows the user to configure the initial settings of the virtual machine in terms of operating system and hardware resources. The libvirtd daemon listens for the requested parameters and lets the virtual machines be instantiated and managed through the API.

2.4 OpenStack Cloud

OpenStack [52] is an open source software toolkit composed of a set of interrelated tools used to build and manage cloud computing infrastructures for public and private clouds. OpenStack started in 2010 as a joint project by RackSpace and NASA; nowadays it has been developed into a project managed by the OpenStack Foundation. The latter includes over 500 IT companies, such as RedHat, Dell and Cisco, and its role is to promote the global development, distribution and adoption of the OpenStack cloud operating system. As explained in Section 1.2.2, a cloud infrastructure provides computing resources for end users in a virtual environment, where the actual software runs as a service on reliable and scalable servers. Indeed, OpenStack allows VMs to be deployed on the computing, storage and networking resources of data centre. It is considered an Infrastructure as a Service cloud model, made up of different key parts to which one can add several other components thanks to its open source nature. With a focus on the CERN use case [53] Nova, Glance, Cinder, Horizon and Keystone are described in the following sections. In addition, Swift [54] can be listed, which provides a scalable storage system, Neutron [55] provides network connectivity-as-a-service between interface devices, Ceilometer [56] as an accounting system and Heat [57] as an orchestration service for multiple composite cloud applications. Finally, Trove [58] provides database-as-a-service provisioning for relational and non-relational database engines, Sahara [59] provides data processing services and Ironi [60], whose provisions are bare metal machines.

2.4.1 Nova

The OpenStack component responsible for provisioning instances on hardware resources is Nova [61]. To deploy the CERN's cloud infrastructure, the resources of

the two data centres of Geneva and Budapest have been partitioned into smaller groups, known as cells.

Nova defines the cloud infrastructure as a tree structure (Fig. 2.5) divided into two hierarchical levels. On the top level there is the parent node, which runs the user facing APIs and the cell scheduler. At the bottom, there are the child nodes, that run all the other Nova services in order to carry out computations. Communication between the final user and the parent cell, as well as between the parent cell and the children cells is established through a message queue broker, which allows the infrastructure to scale up in a distributed way. The creation of a new child cell involves the set up of a new Nova infrastructure and its connection to the parent cell.

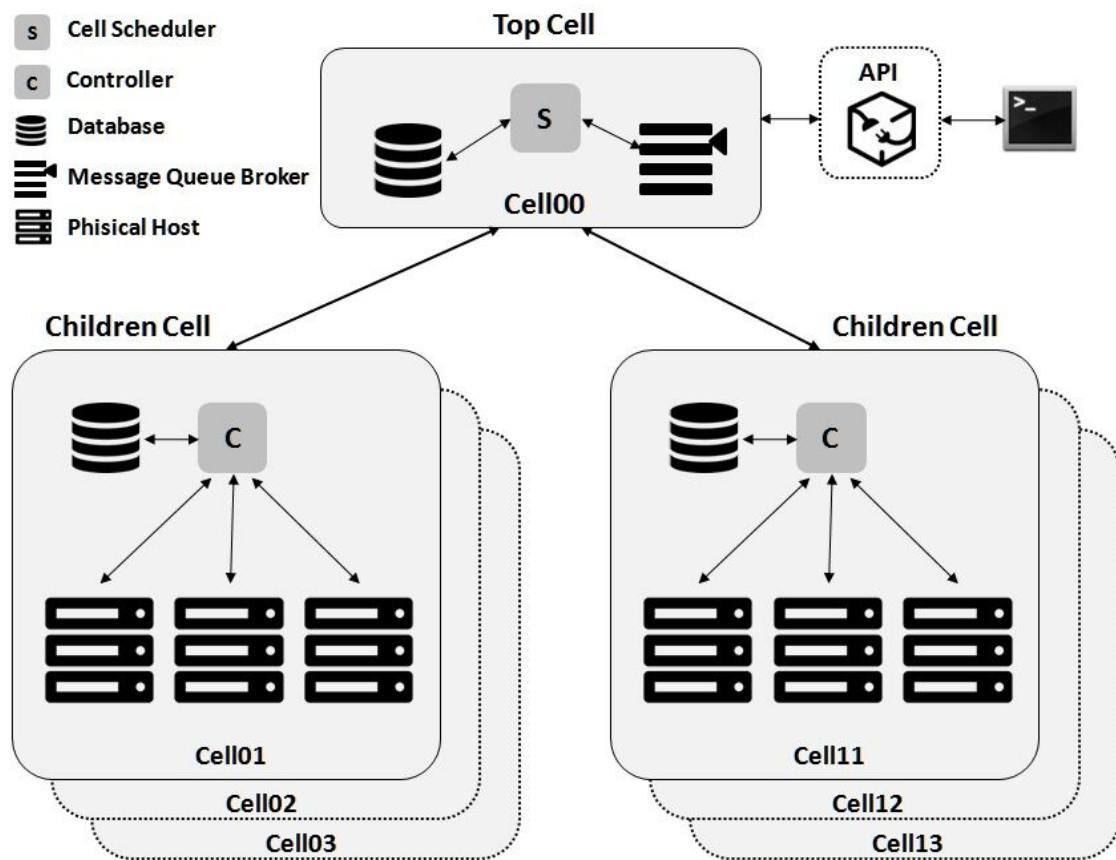


Figure 2.5: CERN's OpenStack architecture.

The Top cell (Parent node, Cell0) runs specific API services, with whom the user has to interact through a command line interface in order to set all the parameters for the VMs to be created. The essential specifications are the virtual machine flavor and the operating system image.

The flavors represent a list of hardware configuration templates which can be set for a VM (Fig. 2.6). A flavor specifies the number of vCPUs, the available memory, the swap area, the number of disks attached and the ephemeral disk dimension. Table 2.2 summarises the main configuration parameter of a flavor.

ID	Name	Memory_MB	Disk	Ephemeral	Swap	VCPUs	RXTX_Factor	Is_Public
1	m1.tiny	512	0	0		1	1.0	True
12076	m2.large	7500	40	0		4	1.0	True
17895	m2.small	1875	10	0		1	1.0	True
2	m1.small	2048	20	0		1	1.0	True
3	m1.medium	4096	40	0		2	1.0	True
38242	m2.medium	3750	20	0		2	1.0	True
4	m1.large	8192	80	0		4	1.0	True
50	win.small	2048	61	0		1	1.0	True
51	win.medium	4096	81	0		2	1.0	True
52	win.large	8192	121	0		4	1.0	True

Figure 2.6: CERN Nova flavors list updated in December 2016.

Field	Description
Name	Descriptive name that is typically not required, though some third party tools may rely on it.
Memory_MB	Instance memory in megabytes.
Disk	Virtual root disk size in gigabytes of the ephemeral disk where the base image is copied. When booting from a persistent volume it is not used. The "0" size is a special case which uses the native base image size as the size of the ephemeral root volume.
Ephemeral	Specifies the size of a secondary ephemeral data disk. This is an empty, unformatted disk and exists only for the life of the instance. The default value is 0.
Swap	Optional swap space allocation for the instance. The default value is 0.
VCPUs	Number of virtual CPUs present in the instance.
RXTX_Factor	Optional property that allows the servers to have a different bandwidth cap with respect to the one defined in the network they are attached to. This factor is multiplied by the <i>rxtx_base property</i> of the network. Default value is 1.0.
Is_Public	Boolean value. Whether or not flavor is available to all users or is private in the project it was created in. The default value is true.

Table 2.2: Description of the attributes of a flavor configuration.

The other component specifying a virtual machine is the operating system image that represents the list of operating system's templates which can be set for a VM. Table 2.3 summarises the publicly available images for the OpenStack CERN updated in December 2016.

Operating system	Architecture
CC7 Base	x86_64
CC7 Extra	x86_64
SLC5 CERN Server	i386 x86_64
SLC5 Server	i386 x86_64
SLC6 Base	i386 x86_64
Windows	i386 x86_64
SLC6 CERN Server	i386 x86_64

Table 2.3: List of the operative system's images updated in December 2016.

In particular, Scientific Linux 6 is a free and open source Linux distribution built by Fermilab, CERN and other universities, based on the source code of Red Hat Enterprise Linux version. The purpose of the project, born in 2003, was to provide a customized x86-based operating system that could be used in the HEP community in order to overcome the problem of sharing applications and results between different operating systems. Thus implying the re-compilation of the software and validation of results. Nowadays, Scientific Linux is the most commonly used OS in scientific and academic environments. CERN released a customized version of the operating system introducing, for example, additional repositories and software useful for CERN's purposes. This distribution has been named Scientific Linux CERN.

Once a virtual machine is instantiated, the image and flavor configurations are saved into the parent cell MySQL database, together with other specifications like the Cinder storage volumes (see Section 2.4.2) and the availability zone. The latter represents a group of network nodes in which the user can create and instantiate new virtual machines accordingly with the resources' availability. If the availability zone is not set, a default one is given, based on the best zone available for instantiating that virtual machine at that time. The subsequent creation of virtual machines follows strict rules thanks to the parent cell's scheduler component: every VM must be created inside a project, each of which is mapped to a specific child cell. The scheduler forwards the request of new virtual machines creation to the correct child cell controller. The dispatching occurs in regard to the order in which the requests are queued in the message queue broker. The creation request waits in the parent's queuing list until one child reports to the parent the resources availability for one of its hosts (also called compute nodes) together with the target host number; the information about its hosts' status are saved in the private child cell's database. Once the child cell takes charge of the creation of the new virtual machines, they

are then submitted to processes that take care of specific configurations, such as the IP service.

At CERN, the number of children cells is about 40 and each of these consists of about 200 physical machines, for a total number of approximately 8 000 physical machines in the Meyrin and Wigner data centres. CERN cells tend to be heterogeneous among each other in terms of hardware, network and operating system, in order to dedicate special resources to the project that requires specific computational power.

Resources	Quantity
Cells	> 40
Physical host per cell	~ 200
Cores	~ 155 000
RAM	~ 350 TB
VMs	~ 18 000
Images and snapshots	~ 2 700
Volumes	~ 2 300
Hypervisors in production	~ 5 800

Table 2.4: Amount of resources for the CERN OpenStack Cloud in December 2016 [62].

2.4.2 Other Main Components

Glance [63] is the OpenStack image service, responsible for storing and maintaining VMs' template images or private images modified for the users' need. CERN's private cloud offers a set of public images such as: Windows, Scientific Linux CERN (SLC) and CentOS with different architectures and packages for CERN specific customisation. Glance is composed of the Glance API and the Glance registry; the former is responsible for delivering images to computing nodes and updating them regularly, the latter maintains metadata information regarding VM images. This extra information exposes the architecture, the operating system family, the distribution of the version and release date. When a new image is released the previous one is set as old and made only visible to the users that use it.

Cinder [64] is the OpenStack's block storage service, its role is to provide volumes to the VMs instances. Volumes life is persistent since it is not tied to the life time of an image, but it can be associated to more than one virtual machine at a time.

Virtual machines can be created through the command line interface thanks to Nova services as explained in Section 2.4.1, but the same kind of operations plus the management of the resources can be carried out with Horizon [65]. This component provides a dashboard that allows for interaction with the underlying services in order to list images, flavors and instances for the sake of checking their status and setting the required metadata. Users can perform standard operations on VM, such

as: creating a virtual machine snapshot, start, pause, suspend, stop or reboot VMs; as well as the possibility to create key-pairs for the SSH connection to the instance.

All OpenStack's operations can be performed only with definite security policies, provided by Keystone [66]. This component manages the identification services and the authentication protocols across other OpenStack services.

2.5 Tier-0 Batch System

The goal of the majority of the virtual resources instantiated on the physical servers of the CERN data center is to expose the computational power for the Tier-0 batch system. The role of a batch system is to queue workloads ensuring a fair-share policy between different users and groups within the system and guaranteeing that the utilisation of the system is maximised at all times. Each user of the batch system can have many submitted jobs at the same time, that are queued and executed independently from each other. The order of the dispatching of jobs to the resources is determined by a fairshare algorithm. The latter ensures that, on average, the resources between experiments are equally shared among users balancing the forecast utilisation with respect to the current usage level.

Often, the outcome of searches is strongly linked to the computing throughput, referred to the rate of processing data per unit of time. A solution has been found in offering an effective and centralized environment known as High Throughput Computing (HTC). It is defined as a computing paradigm suited to efficiently run multiple loosely-coupled tasks on many different computing resources across multiple administrative boundaries at the same time. The events gathered from the LHC's experiments are loosely-coupled by nature, so they can be computed following the HTC paradigm. The key to HTC is effective management and exploitation of all available computing resources, with the goal of accomplishing high-throughput over relatively long periods of time. HTC is, by design, a system based on unreliable components, being that if some nodes fail in performing the job, the latter must be restarted on a different system. Therefore, the HTC community is concerned with robustness and reliability of jobs over a long-time scale. The concept of High Throughput Computing appeared in 1997 during the presentation of Condor, a software system developed at the University of Wisconsin that promised to expand computing capabilities through efficient capture of cycles on idle machines.

The average CPU utilisation of the CERN batch resources is around 70%, considering all kinds of workload, some of those being I/O bound (simulation, reconstruction and analysis). For pure CPU jobs the efficiency goes above 90% and for this reason, LHC computing needs are often identified as CPU intensive tasks. At CERN, the events to be processed are independent from one another and therefore imply an intrinsic parallelism of the jobs. The current batch system for the high throughput computing is LSF, an acronym for Load Sharing Facility Platform [67]. The batch system is being gradually migrated from LSF to HTCondor. HTCondor provides traditional functions such as job queueing mechanism, scheduling policy, priority scheme, resource monitoring and resource management. It also provides a flexible framework for matching job requirements and job preferences with exposed resources and their requirements about the jobs they are willing to run.

Chapter 3

Benchmarking Computing Resources

The introduction of new computing resources is a common practice in data centres that have to satisfy an increasing demand of computational power. Furthermore, the cycles of renewing the already installed resources is part of the management of the data centre. Both the new and the already installed resources need to be continuously monitored in order to verify their performance. The project of this thesis deals with the evaluation of the computing performance of a new set of 240 servers introduced in the spring of 2016 in the Wigner data center. The purpose was to extend the computing capacity of the Tier-0 batch system.

This Chapter summarises the key concepts of the work that was done. The new resources, their hardware components, the virtual machine configurations as well as the benchmarking tools and the scheduling of the tests used to evaluate the performance of the system is described. Finally, how the results are collected, stored and analysed is explained.

3.1 Performance Benchmarking

The performance monitoring of the computing infrastructure is crucial to quantify the quality of the service (QoS) and to control the behaviour of the components. The role of the monitoring tools is to observe the systems performance by collecting and analysing performance statistics, with the aim of identifying problems and suggesting possible remedies. The goal is to reach the best balance between the needed performance and the associated costs. In this process, it is fundamental to compare the perceived and the presumed performance of a service in order to detect issues, to choose the most effective resources and to define the possible bottlenecks. In the CERN data centre, for instance, to enhance the reliability of the service, the physical resources alongside the virtual resources are monitored to gather metrics from the running service.

There are many aspects that can interfere with the performance monitoring process. For example, the types of applications are so numerous that it is not possible to have a standard measure of performance for all the use cases. In addition, it is difficult to identify the set of goals and the system parameters, since they can significantly influence the performance when changed. To reach a good level of accuracy of results, it is important to find the best evaluation techniques and the

right measures of performance. Finally, to obtain useful insights, the resources must be profiled with respect to realistic workloads.

3.1.1 Benchmarking Process

The benchmarking process is the comparison of performances between two or more systems by means of reference workloads, the so called benchmark measurements. In order to produce a correct performance analysis, the measurements should be executed under stable conditions and the test workload must be as representative as possible of the average behaviour of the real workload.

As explained in [68], the term workload stands for the amount of the actual work that a system instance or a set of instances are going to perform. Two main categories can be identified with respect to their roles. The *real workload* is the amount of work observed on a system; it cannot be easily reproduced for benchmarking purposes, and therefore, is generally not suitable to be used as a test case. The *synthetic workload* simulates a workload whose characteristics are similar to the real one, but that can be applied repeatedly in a controlled manner. The evaluation of the test workloads have evolved together with the complexity of the computer architectures. Initially, the most frequent instruction in computers was addition, therefore the performance of the system was synonymous to the performance of the processor, which was the most expensive and used component. Thus, as an early approximation, the computer with the fastest addition instruction was considered to be the highest performing one. As the number and complexity of instructions grew, more detailed workload descriptions were required. Therefore, measurements of the relative frequencies in instruction mixes on real systems have been introduced. Furthermore, together with the introduction of complex instruction sets that represent high level functions, new workload types, known as kernels, have been defined. This generalization of the instruction mix was due to the introduction of pipelining, instruction caching and address translation mechanisms, which caused the individual instructions to no longer be considered as isolated. However, most of the kernels do not take into consideration the input/output (I/O) operations, which are an important part of the real workload. Therefore, in order to measure I/O performances, simple loop codes, known as synthetic programs, have been developed to simulate a specified number of service calls or I/O requests. Finally, in order to compare computer systems used for a specific application, a representative subset of functions can be used to simulate its behaviour. Benchmarks are generally described as a set of programs that behave like a real workload that should be performed on the system, which uses almost all resources including processors, I/O devices, networks, and databases.

For the purpose of this project, the resources have been tested, before the release of the new servers, during their commissioning phase, by three types of benchmarks that emulate the HEP workloads. The identification and validation of those benchmark workloads have already been conducted in the WLCG context [69] thus justifying the choice described in Section 3.4.2.

3.2 Hardware Characteristics

The 240 servers, installed and commissioned in the Wigner data center, are characterized by homogeneous hardware, whose components has been chosen managing a balance between performance requirements and hardware cost. In the following section are described the main components of the hardware: CPU, RAM and storage.

3.2.1 CPU

Each server is equipped by 2 Intel Xeon Processor E5-2630 v3 CPUs [70]. The Xenon brand has been introduced by Intel to identify x86 microprocessors targeted for non-consumer workstation, servers and embedded systems. In particular, this CPU derives from the Haswell-EP (E5-16xx/26xx v3) family, which defines the x86 eleventh generation of Intel architecture, developed from 2013 starting from the Sandy Bridge architecture. Table 3.1 reports the most important specifications of the CPU Xeon E5-2630 v3.

Characteristic	Value
Architecture	x86_64
Processor Base Frequency	2.40 GHz
Max Turbo Frequency	3.20 GHz
Cache	20 MB SmartCache
L1d cache	32k
L1i cache	32k
L2 cache	256k
L3 cache	20480k
Number of cores	8
Number of threads	16

Table 3.1: Intel Xeon Processor E5-2630 v3 specifications [70].

In the production set-up for the Tier-0 batch system, each CPU has enabled the simultaneous multithreading (SMT). Therefore, each CPU, equipped with 8 physical cores, can host 16 logical cores by enabling 2 threads per core. As a result, the total number of logical cores for each server is 32. Figure 3.1 shows a pictorial representation of the logical threads in a server.

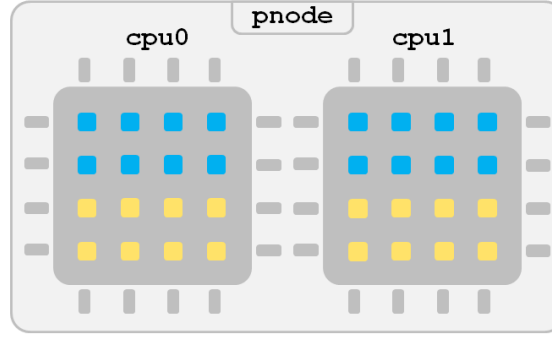


Figure 3.1: Each server, also called physical node - pnode - is composed by two CPUs, each of which has enabled 16 threads on 8 physical cores logical cores thanks to SMT.

The CPU supports the NUMA architecture. NUMA stands for Non-Uniform Memory Access [71] and identifies an architecture in which each processor is provided with its own local memory, whereas the memory provided to the other CPU on the same socket is considered remote. Therefore, depending on which core the process is running, the access to the memory can take different amounts of time: faster if it accesses the local memory, slower if it accesses the remote memory. Each server is characterised by two NUMA nodes. Therefore, the 32 logical cores belong to two different NUMA nodes. Numbering the logical cores from 0 to 31, the logical cores pinned to each NUMA node are reported in Table 3.2.

NUMA node	Logical core
node 0	0-7 16-23
node 1	8-15 24-31

Table 3.2: Pinning of the logical cores of the two CPUs on the NUMA nodes.

When a VM is instantiated, the NUMA architecture implies that each logical core of a NUMA node is associated to a virtual CPU (vCPU) of the VMs with the rule that vCPU of the same VM cannot be pinned to cores belonging to different NUMA nodes. For example, on the same server, the instantiation of two VMs with 16 logical cores each implies the first VM to pin in the interval [0-7] and [16-23], the second one to pin in the interval [8-15] and [16-23]. Whereas, in the case of four VMs on the same server with 8 logical cores each, 2 virtual machines are placed on one CPU and the other two on the sibling CPU. Therefore, two VMs randomly pick the cores from both partitions of the first NUMA node [0-7] and [16-23], the other two pin the cores for the second NUMA node in a similar way.

3.2.2 RAM

The server RAM is 8 x 8GB of DDR4 @ 1866MHz. DDR4 is a SDRAM, whose abbreviation stands for Double Data Rate fourth generation Synchronous Dynamic Random Access Memory. It has a greater range of available clock speeds and timings and reduced latency with respect to its predecessors DDR3 and DDR2. DDR4 chips are expected to support transfer rates between 2133 - 4266 MT/s (Million Transfers per second), whereas, DDR3 technology supports up to 800 - 2133 MT/s. One more enhancement with respect to its ancestors, is less power consumption: 1.2 Volts compared to 1.65 Volts of DDR3 chips.

3.2.3 Storage

Each server is composed of two Samsung MZ7KM960HAHP-00005 SSDs (Solid State Drive) [72] in RAID-0 configuration. Table 3.3 reports the main characteristics of the selected SSD.

The preference of SSDs to HDDs is mainly due to the type of workload that the servers should manage, and the need to reduce power and space consumption while increasing performance and energy efficiency. In particular, the analysis performed by the procurement team can be sustained by many external factors. According to Samsung [73], SSDs can increase performance to 60% and the energy efficiency of about 30%, with 26% of additional costs. HDDs require more time to speed up when booting and when it performs I/O operations due to the mechanical parts of the device. For example, the SSDs can perform from 20 000 to 30 000 IOPS compared to the 100 IOPS of an HDD. Furthermore, SSDs do not suffer from a seek-time penalty that manifests mainly in file fragmentation in HDD. Storage requirements for computing nodes are driven by high throughput and low latency in accessing local data for running jobs. Typically those data are temporary, within the lifetime of the job. Indeed, computing nodes do not store persistent copies of data and the job related data are purged at the end of the job itself. For this reason, the storage of the computing nodes is configured as RAID-0. RAID, an acronym for Redundant Array of Independent Disks, is a data storage virtualisation technology that combines multiple physical disks into a single logical unit. The RAID classification is defined with regard to how the data are distributed across the drives, for the purpose of providing different balances among reliability, availability, performance and capacity. In particular, RAID-0 consists of distributing data among disks without mirroring, causing no redundancy for handling disk failures. Benefiting the speed in read/write and disk occupancy without any overhead caused by parity controls. It should be noted that RAID-0 is only adopted for the computing nodes of the batch system, since for the CERN storage resources the data redundancy is largely guaranteed.

Characteristic	Value
Model	SM863
Sequential write (128KB)	485 MB/s
Random write IOPS (4KB)	28K IOPS
Sequential read (128KB)	520 MB/s
Random read IOPS (4KB)	97K IOPS

Table 3.3: Samsung MZ7KM960HAHP-00005 SSD specifications [72].

3.3 Server Virtualisation

Once the physical resources have been settled into place, the servers have been configured to run OpenStack and the QEMU/KVM hypervisor. The resources have been deployed as a new Nova cell. For the purpose of this work, the operating system that has been installed on the VMs is Scientific Linux CERN release 6.8 (SLC6) [74], since it is similar to the one used by the CERN batch systems. Once the instantiation request is submitted and accepted by the OpenStack child cell controller, QEMU/KVM takes care of downloading the requested image and saving it in a file. It should be noticed that for optimisation purposes, the operating system image is downloaded only once in order to save space on the hypervisor. Each VM points to the same image file and the differentiation is done by a specific storage space that contains the files, packages and executables of each given virtual machine.

In order to perform a comprehensive analysis of the performance of each VM configuration, different VM flavors in terms of virtual cores, memory size and disk space have been provisioned. Table 3.4 summarises the properties of the tested configurations. It reports the name of the configuration used during the analysis phase, the characteristics in terms of virtualised hardware and a pictorial representation of the core allocation. Moreover, in the pictures, the external rectangle, flagged as pnode, represents the physical host and the two internal squares represent the CPUs with 16 logical cores. The VMs, instantiated on top of the hardware components, are visualised by a dotted line, each of which owns the virtual CPUs depicted as black squares. As shown in the pictures, the vCPUs of a single virtual machine belong to the same physical CPU only, with the exception of VM-32 which uses all the available logical cores from both CPUs. Core pinning is made possible by configuring the OpenStack scheduler to be NUMA aware, so that the allocation of logical cores is limited to a single NUMA node. It is important to emphasise that the pinning of vCPUs and logical cores shown in the pictures is only a pictorial representation since there was no organised pinning among vCPUs and logical cores. Finally, the RAM size of each VM is proportional to the number of vCPU.

Another configuration applied to the servers in order to improve the performance is connected to the memory pages. When a process uses the memory, the RAM is

allocated by the CPU in default chunks of 4K bytes, known as pages. Since the process address space is virtual, the CPU and the operating system need to keep track of which page belongs to which process, and where it is stored. When the system needs to access a virtual memory location, it uses the page tables inside the Memory Management Unit (MMU) to translate the virtual address to a physical address. Obviously, the amount of time needed to find where the address in the memory is mapped is proportional to the number of pages. For example, when a process uses 1GB of memory, in the worst case, 262.144 entries must be checked ($1\text{GB} / 4\text{K}$). Most CPU architectures support pages bigger than 4 KB, known as huge pages, which enable the kernel to load fewer mapping table into the Translation Lookaside Buffer (TLB). The latter is the cache of page tables that speeds up the translation of virtual addresses to physical addresses reducing the accessing overhead. The smaller number of pages reduces the overhead involved in performing memory operations, and also reduces the likelihood of a bottleneck when accessing page tables. For this reason, this approach can improve system performance by reducing the cycles required to access page table entries. Huge pages size can vary from 2 MB to 256 MB, depending on the kernel version and the hardware architecture. For the purpose of this project, the servers have been configured with a huge page dimension of 2 MB. Memory for huge pages is allocated during the system startup, and it remains allocated until the configuration is changed. This increases the possibility of getting physically contiguous memory; if the kernel is unable to allocate huge pages in a node of a NUMA system, it will attempt to make up the difference by allocating extra pages on other nodes with sufficient available contiguous memory.

3.3.1 VM Provisioning Approach

In order to collect the performance metrics under similar conditions, the number of provisioned virtual machines per server in a given configuration has been fixed to 32 divided by the number of vCPUs of the VM in that configuration. This approach allows to fully load the host resources during the benchmarking process. The virtual machines are contextualised by means of a cloud-init script that is injected at provisioning time. This script prepares the environment to run the benchmarks (see Appendix A). The provisioning action was managed by a Python script using the Nova API to interact with the OpenStack infrastructure. The script takes care of scaling the number of virtual machines up to the expected values for each configuration and server, verifying the number of alive virtual machines and deleting those that are in state of error, shut-off or not active. Due to not being able to target a specific server to provision, a given virtual machine, all the servers were filled up and, if needed, were scaled down to the configured value. This process was iterative, until the correct number of alive virtual machines per server was reached in the requested configuration. Moreover, the script took care of scheduling the benchmark run on the virtual machines (see Appendix A).

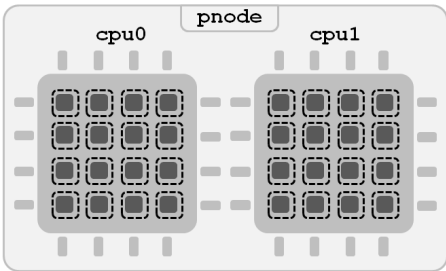
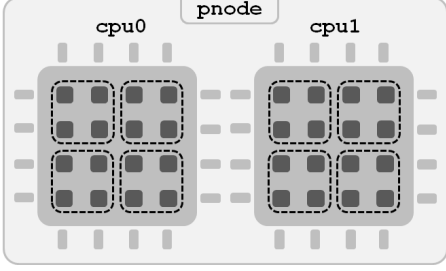
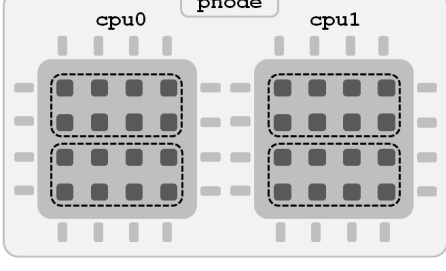
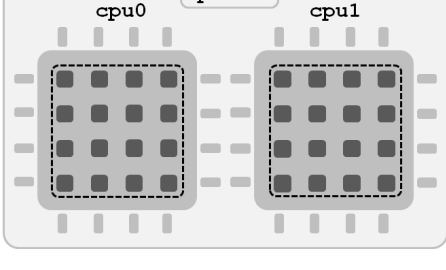
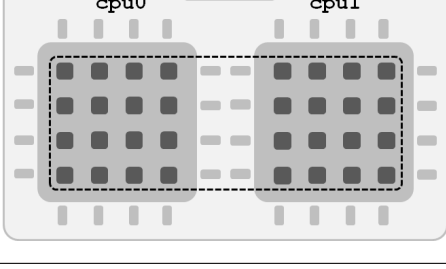
Scenario	Flavor properties	Graphical representation
VM-1	VMs on physical node: 32; vCPU per VM: 1; Memory size per VM: 1 874 MiB; Disk space per VM: 40 GB;	
VM-4	VMs on physical node: 8; vCPU per VM: 4; Memory size per VM: 7 500 MiB; Disk space per VM: 120 GB;	
VM-8	VMs on physical node: 4; vCPU per VM: 8; Memory size per VM: 15 000 MiB; Disk space per VM: 240 GB;	
VM-16	VMs on physical node: 2; vCPU per VM: 16; Memory size per VM: 30 000 MiB; Disk space per VM: 480 GB;	
VM-32	VMs on physical node: 1; vCPU per VM: 32; Memory size per VM: 60 000 MiB; Disk space per VM: 960 GB;	

Table 3.4: Virtual machines configuration for each scenario. The pictures show the logical cores distribution among the virtual machines in a pictorial way, because no pinning was applied inside a given CPU.

3.4 Benchmarking Tools

3.4.1 Benchmark Suite

To cope with the benchmarking process and to ease the systematic collection of measurements, CERN built the so called Benchmark Suite, which is a scalable tool able to run HEP based benchmarks on computing resources. The suite is an open-source tool characterized by a light installation process, that takes place during the VM contextualisation phase, a configurable set of benchmark jobs to run and a JSON-based measurement report. The JSON document contains the results and metadata information such as the name of the virtual machine, the IP address, the operating system and the CPU type. The files are transferred through the AMQ [75] transportation layer and sinked in with Logstash into Elasticsearch, where subsequently the data is monitored with Kibana or extracted and analysed with other analytic tools, such as iPython and Pandas libraries. The transportation layer Elasticsearch and Kibana belong to the Elastic family tool (see Appendix B). Whereas iPython and Pandas are described in Section 3.4.4.

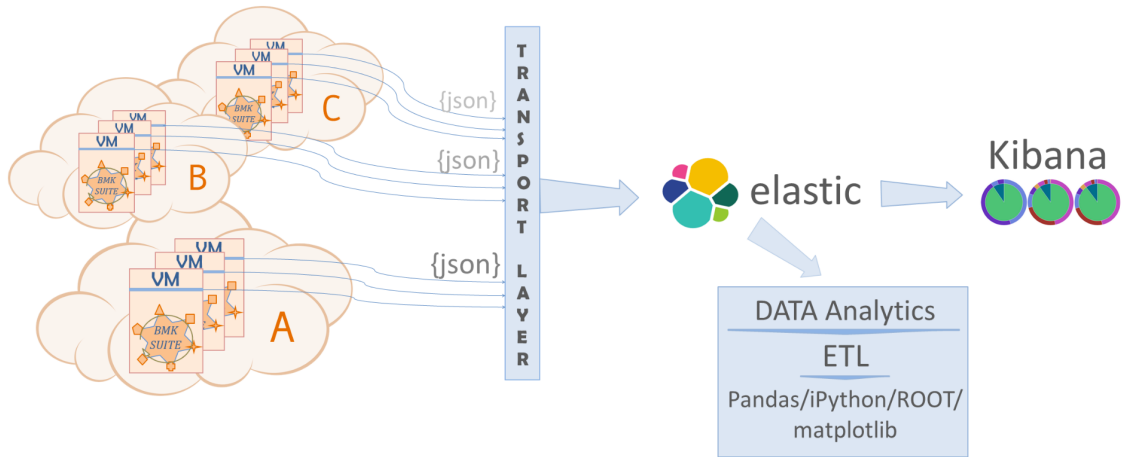


Figure 3.2: CERN Cloud Benchmarking Suite [76].

The benchmarks adopted are single threads used to reflect most of the LHC experiments software applications. In order to fully load the computing resources and reproduce the worst case load scenario for a batch node, where all the computing slots is running jobs and the CPU is not idle, the benchmarks procedure is configured to run in parallel as many threads as the number of vCPUs of each VM. The benchmark result is calculated as the arithmetic mean of values collected from each thread.

Given that a physical server generally hosts more than one virtual machine at the same time, in order to fully load the server, a synchronisation of the benchmarks among the virtual machine is needed to fully load the server. It is possible to generate load running benchmark tests in two modes, referred to as sequential and synchronised. Referring to Fig. 3.3a, the sequential mode allows for the execution of a scheduled benchmark sequence, for instance the sequence ABC, repeatedly in time with no break interval between two consecutive processes. The sequence repetitions

are configurable and usually occur until the VM is destroyed. This running mode is useful to understand how a generic workload in a VM can affect other workloads in other VMs hosted on the same server. The duration of each benchmark job, which varies either with respect to the VM or with respect to the execution itself, adds up over time making different benchmarks overlap with regard to different VMs. The synchronised mode, instead, allows the benchmark job to start in a specific moment t_i , as in Fig. 3.3b, thanks to a system task scheduler typically implemented as a cron job. For this reason, a specific benchmark job starts at the same moment on different VMs allowing the study of the performance of a given virtual machine.

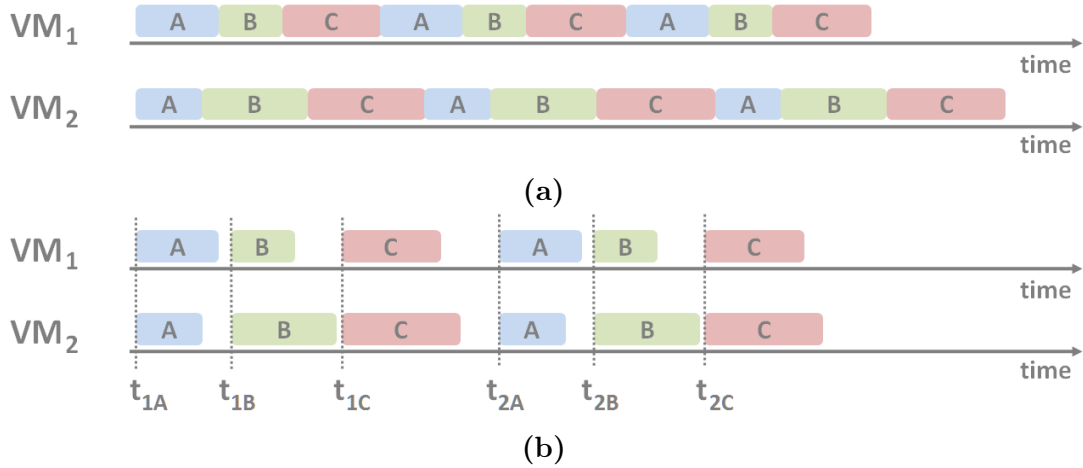


Figure 3.3: Example of benchmark job sequences ABC executed in sequential mode (a) and in synchronised mode (b) in two different VMs.

3.4.2 Benchmark Workload

The benchmark jobs included in the suite focus on the speed in accomplishing a given computing task by the central processor unit. Those tasks are mainly random number generations, or the simulation of HEP event execution inside a LHC detector. The benchmark jobs are respectively called ATLAS Kit Validation, Dirac Benchmark 2012, Whetstone Benchmark and HEP-SPEC06. In the following a description of them is provided.

ATLAS Kit Validation, referred to as KV in the following, is a tool-kit representative of a real LHC experiment workload: it configures and runs the simulation of the ATLAS detector response to the propagation of high-energy particles through the ATLAS detector material. The simulated environment is loaded at the start of the KV benchmark, defining the detector geometry, the system's configuration, the list of particles and the configuration for the Monte Carlo generator. In a single KV run, N independent events of a single muon going through the detector are simulated in sequence. The CPU time needed to simulate each event is recorded and the average over the N events is computed and represents the benchmark result. The random seed used by the random number generator is fixed at configuration level in order to guarantee the reproducibility of the pseudo-random number sequence across different tests. This approach makes sure that the comparison of tests is not

also affected by the randomness of the sequence, which would reduce the resolution of the measurements increasing the spread. Moreover, in order to exclude from the measurement the overhead due to the initial load of the software libraries and the configuration data, the first event in the sequence is omitted from the results. This is the event where most of the execution time is spent in setting up the simulation environment. A single KV run lasts about 5 minutes.

Dirac Benchmark 2012, addressed in the following as DB12, identifies a python code that generates a sequence of Gaussian pseudo-random numbers [77]. This approach has been chosen since random numbering is a typical task in the Monte Carlo simulation, which is one of the most frequent kind of jobs running on the WLCG. The number of random generations can be configured to keep the CPU busy for enough time to make the benchmark measurements reliable. The number of iterations is fixed at $12,5 \times 10^6$, which corresponds to 250 HS06 seconds. However, in order to also have a fast measurement, the running time should not be longer than the minimal interval needed to have a reliable measurement. The duration of its run is about 1-2 minutes. Even if it is a simple concept, it has been proved to scale well with the experiment job duration and with the HEP-SPEC06 benchmark for most of the CPU models [78].

Whetstone [79], referred to as WSN in the following, is a synthetic benchmark introduced in 1972 that contains several modules meant to represent a mix of "typical" operations executed by scientific calculators, such as C functions including: sin, cos, sqrt, exp, and log as well as integer and floating-point math operations, array accesses, conditional branches, and procedure calls. Despite its synthetic mix of operations, the goal of this benchmark is to measure the performance of both integer and floating-point arithmetic and is mostly representative of small engineering/scientific applications that fit into cache memory. At the beginning, this program gave a measure of the machine's speed in thousands of Whetstone instructions per second (kWIPS), but the growing increase in computer speeds changed the unit of measure into Millions of Whetstone Instructions Per Second (MWIPS). The duration of its run is about 1 minute. The choice of selecting this benchmark as one player in the CERN Benchmarking Suite has been driven by the historical role it played, used as a control in the analysis for the other benchmarks.

HEP-SPEC06 Benchmark, addressed as HS06 in the following, is the official CPU performance metric used by WLCG sites in order to assess their computing power. It has been defined by the HEPiX Benchmarking Working Group [80]. Its aim is to provide performance measurements that can be used to compare compute-intensive workloads on different computer systems; it consists of reproducible CPU benchmark jobs to describe experiment requirements, measure existing compute resources and procure new hardware. HEP-SPEC06 is based on the benchmark subset of the industry standard SPEC¹ CPU2006 benchmark suite. These benchmarks stress the performance of computer processors, floating point units and memory architectures.

¹ The SPEC (Systems Performance Evaluation Cooperative) is a corporation of leading computer vendors that developed a standardised set of benchmarks drawn from various engineering and scientific applications.

The CPU2006 standard [81] contains two benchmark suites: the SPECint2006 for compute-intensive integer performance and the SPECfp2006 for compute-intensive floating point performance. Experiment workloads usually perform, a much higher number of integer operations compared to the floating point ones. However, the SPECint2006 has a 0.1% percentage of floating point operations compared to the experiment workloads with a percentage of floating point operation of about 10% [82]. To avoid this discrepancy a different benchmark set has been adopted, which represents the C++ benchmarks contained in CPU2006 and it is identified by the flag ALL_CPP. There are seven benchmarks in the ALL_CPP suite, 3 of them are from SPECint2006, while the other 4 are from SPECfp2006 suite. The final result is computed as the geometric mean of the 3 integer benchmark results and of the 4 floating-point benchmark results.

Integer		Floating point	
Test name	Description	Test name	Description
444.namd	molecular dynamics	471.omnetpp	discrete event simulation
447.dealII	finite element analysis	473.astar	path finding algorithms
450.soplex	linear programming and optimization	483.xalancbmk	XML document processing
453.povray	image ray tracing and image rendering		

Table 3.5: ALL_CPP benchmark set.

SPEC can run benchmark jobs on multi-core machines in two ways, identified as SPECspeed and SPECrate. SPECspeed runs a single copy of the benchmark on the machine, using only a single core. The SPECrate benchmark runs in multi-thread configuration with as many benchmarks as the core number and it calculates the results as a function of the total elapsed time (Fig. 3.4 above). In this case the idle time of the fastest and the slowest is included in the performance result, which then appears to be lower compared to what is delivered to the WLCG jobs. In order to correctly emulate the WLCG job behaviour consisting of multiple independent single-threaded applications sequentially scheduled on a core, for the HEP-SPEC06 another approach has been followed, called Multiple Speed (Fig. 3.4 below). Multiple independent SPECspeed runs, one for each core, are run parallel in the system. Then, the single results are summed up obtaining the total result for the system.

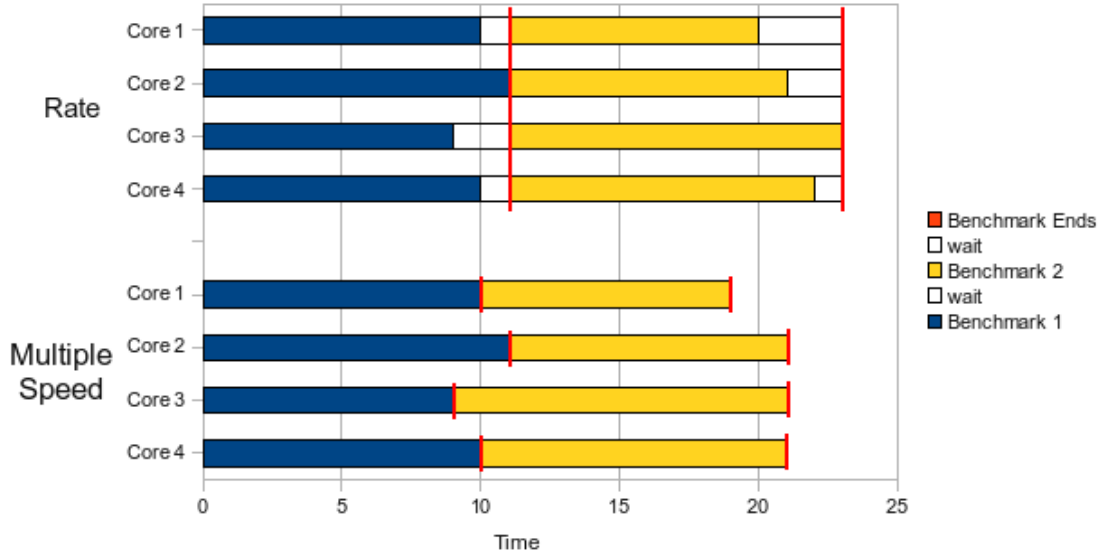


Figure 3.4: SPECrates vs HEP-Spec06 Multiple Speed [82].

3.4.3 Outline of the Benchmarks in the Configurations

The performance measurements have been conducted for every use case defined in Table 3.4, through the described Benchmark Suite, running synchronised sequences of Whetstone, DB12 and KV. In addition, for the VM-8 and VM-16 configuration the HEP-SPEC06 benchmark has also been run. VM-8 and VM-16 have been selected since they are the most interesting and most used configurations for the batch system. That choice was driven by time constraints for the commissioning phase with respect to the duration of about 6 hours of HEP-SPEC06 benchmark. Given that the latter is not included in the Benchmark Suite, it has been launched separately following the same synchronised approach adopted for the other benchmark runs. Table 3.6 summarises the benchmarks done for each configuration.

	DB12	WSN	KV	HS06
VM-1	X	X	X	
VM-4	X	X	X	
VM-8	X	X	X	X
VM-16	X	X	X	X
VM-32	X	X	X	

Table 3.6: Summary of the benchmarks performed for each scenario.

3.4.4 Analysis Tools

Kibana supplies a powerful real-time tool that allows for the monitoring and validation of the data acquisition via predefined statistical plots. Nevertheless the analysis process requires adequate tools for in depth investigations that, as of today, go beyond the capabilities of Kibana. Our choice is Python and the Pandas tool-kit. Python is a dynamic and flexible object-oriented programming language having powerful libraries for data manipulation and an ecosystem of tools for data analysis. It provides many advantages such as its simplicity. Analysts can easily learn how to use it due to its syntax, as well as its nature as a scripting language. Moreover, it integrates well with existing IT infrastructure since it is platform independent. For the purpose of this project, the iPython notebooks [83] have been used with the support of numerical and graphical modules such as Pandas [84], an acronym for Python Data Analysis. The latter provides simple data transformation tools and it offers data structures for easily manipulation numerical tables and time series. In order to be able to work with data from the suite using iPython, it is necessary to retrieve data in JSON format, from Elasticsearch, transforming and saving them into a suitable data structure. This data extraction workflow is pictured in Fig. 3.5 and detailed in Appendix C.



Figure 3.5: The benchmark results are collected into Elasticsearch and analysed with Kibana. Moreover, they can be retrieved manually as JSON files, saved into a repository and transformed into a suitable data structure to enable analysis with Python and the Pandas libraries.

Chapter 4

Analysis of Benchmark Data

The performance of computing nodes which run benchmarks representative of the HEP simulation workloads has been studied for different VMs configurations. These configurations have been provisioned partitioning the compute resources in such a way that each CPUs' server was fully loaded during the tests. This chapter details the analysis of the benchmark results for each configuration. The uniformity of the data collected along time is studied, as well as the main properties of the measurements distributions. From the latter, statistics, such as mean, variance and peak values, are extracted. Lastly, the relative performance of the benchmarks across configurations is studied, also providing a comparison with the HEP-SPEC06 benchmark.

4.1 Data Handling

For the purpose of this project, the benchmarking of the CPUs involved many servers, each of which has been studied repetitively over time under different configurations. The precondition to the analysis is a data quality and handling phase, during which the content of the data is assessed, corrupted data and missing information are identified and derived quantity, such as time aggregations and averages, are produced. The raw data are preliminary stored in a suitable Pandas DataFrame [85], which is a tabular data structure that enable slicing, grouping and fitting operations over the dataset. Statistics, such as a mean, median and percentile, are extracted from data with the goal of summarising large amounts of data and to understand the average behaviour of the full population. In more details, the performance of a VM is calculated as the average of the parallel tests on its vCPU. Therefore, the distribution of the averages of the parallel results is investigated to verify the assumption of homogeneous and reproducible results over time and across different servers. This assumption can be broken down by several effects that need to be investigated and explained. For instance, the non-uniformity of the performance of the different servers can break the hypothesis of equally distributed variables. In addition, due to changing in the work conditions of the servers, the repeated measurements can be affected by variations that need to be identified and taken into account. In addition to the study of the distribution, the analysis is sustained by the identification of patterns and relationships in the raw

data through data visualisation tools. For the purpose of the analysis process of this thesis statistical graphs such as histograms, boxplots, scatter plots and error bars plots have been employed.

4.2 Configurations

In this section, each configuration (see Table 3.4) is described starting from the introduction of the benchmark execution, up to the analysis of the results.

4.2.1 VM-32

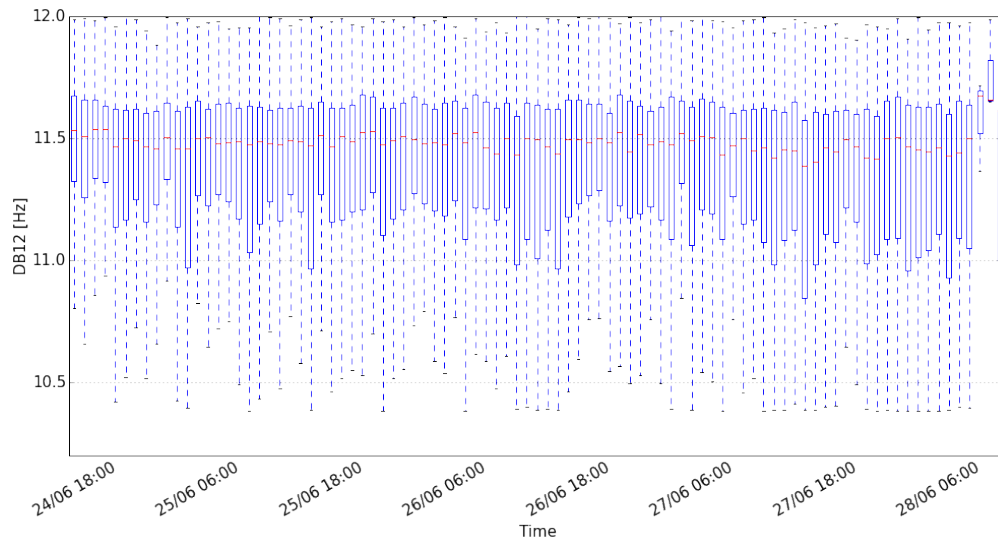
VM-32 defines the configuration in which one VM with 32 vCPUs is provisioned in each physical node. Given that the overcommit of the resources is not possible, the VM is the single user of the the resource. Table 4.1 summarizes the total running time and the overall number of tests performed. More than 270 tests per server have been collected in an interval of 93 hours.

Running period	93 hours
Tests per hour	3
Number of physical nodes	237
Total number of benchmark executions	~ 62 000

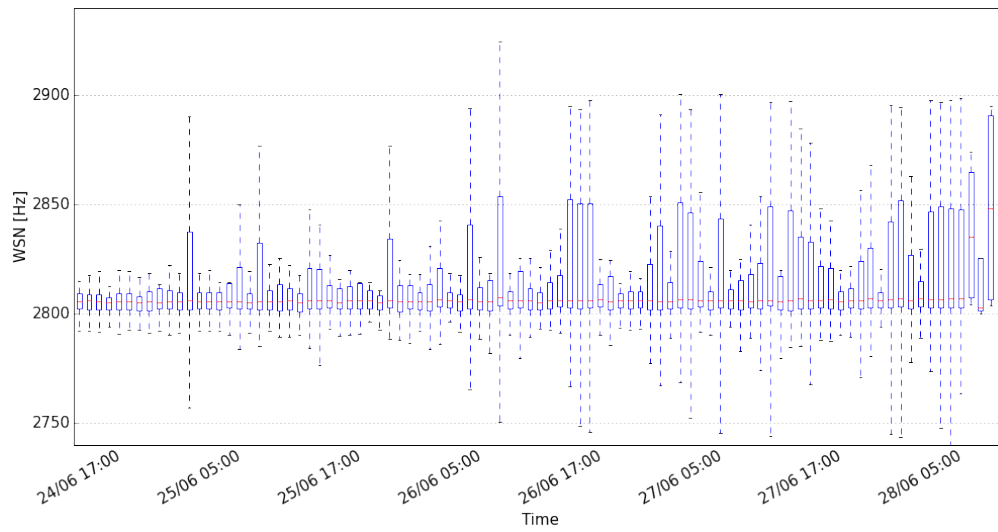
Table 4.1: Test summary for configuration VM-32.

Figure 4.1 shows the distribution of the benchmark results for all of the VMs, summarising as boxplots the dispersion of the values for each hour. The measurements were uniform over time with the presence of outliers.

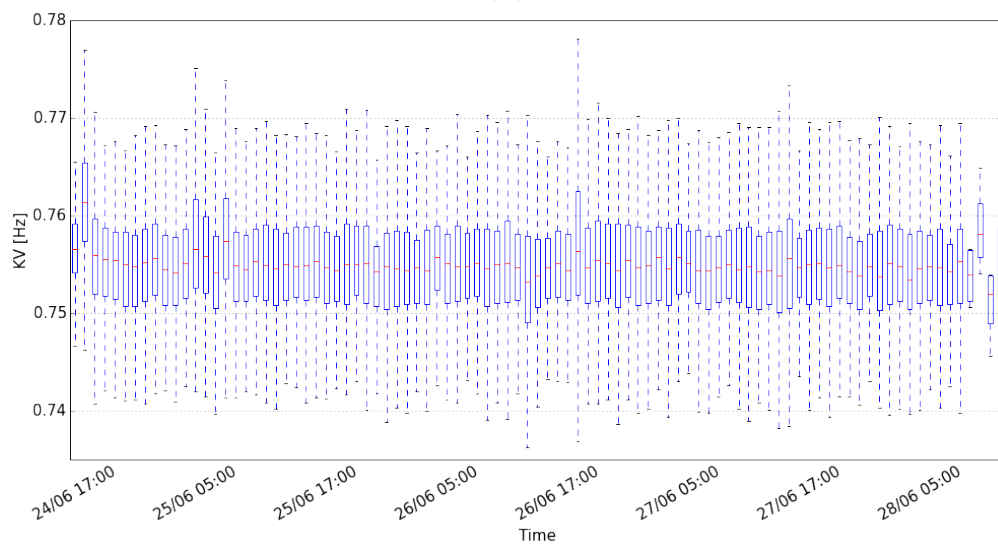
Figure 4.2, Fig.4.3 and Fig. 4.4 visualise the distribution of benchmarks results, for DB12, WSN and KV respectively. Moreover, the tables next to the plots summarise the mean (m) and its error, calculated as the ratio between the root mean square (s) and the root square of the entries, the root mean square, the most probable value (mpv), as well as the Gaussian fit parameters. These last values have been calculated through an optimisation algorithm, whose goal is to determine the best convergence of the fit varying the fit range. In order to evaluate the quality of the fit, the χ^2 test of goodness has been adopted, fixing the p-value at 0.05. When the fit range is tight, it means that the underlying distribution endures non-Gaussian tails. For the purpose of evaluating the best fit for the underlying function, the Gaussian fit has been performed in the neighborhood of the maximum peak value. Figure 4.2, for the DB12 values, clearly shows the presence of a double substructure in the distribution. The WSN results in Fig. 4.3 are distributed symmetrically around a central value, with the exception of clustered outliers. Figure 4.4 shows KV values distributing as a Gaussian.



(a)



(b)



(c)

Figure 4.1: Hourly dispersion of the test results summarised as boxplot, for the DB12 (a), for the WSN (b) and for the KV (c) respectively in the VM-32 configuration.

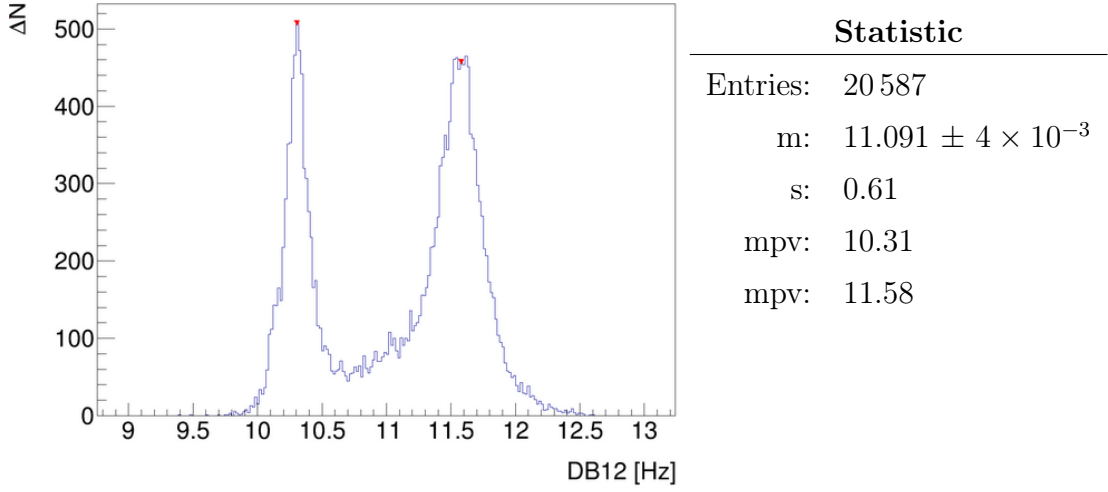


Figure 4.2: Distribution of the DB12 measurements in the VM-32 configuration.

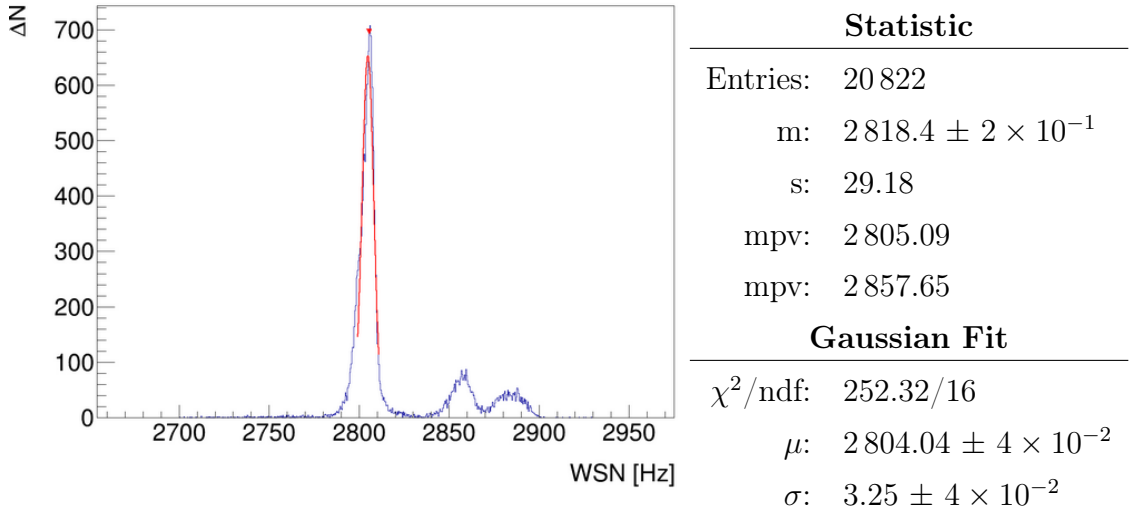


Figure 4.3: Distribution of the WSN measurements in the VM-32 configuration.

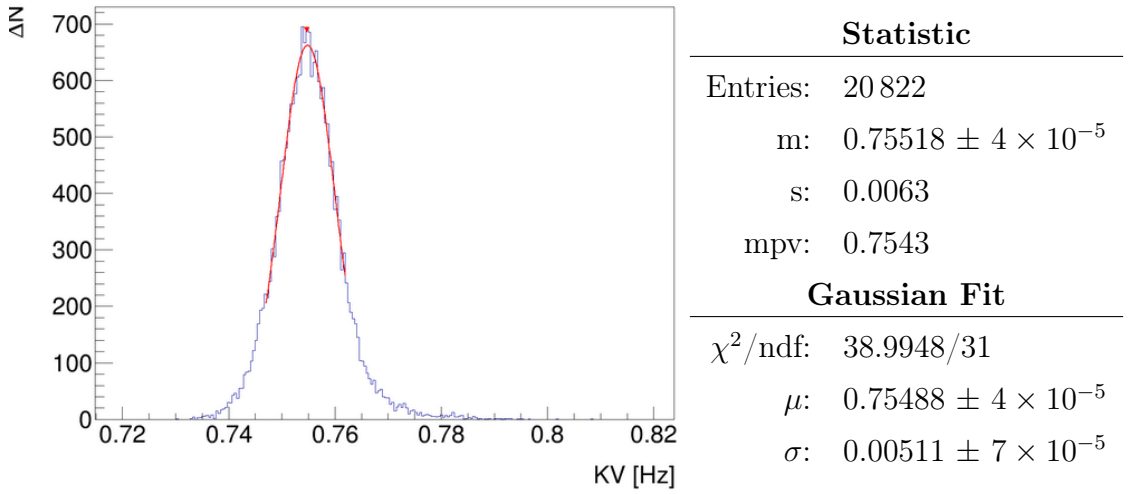


Figure 4.4: Distribution of the KV measurements in the VM-32 configuration.

The unusual DB12 profile in Fig. 4.2 reveals the presence of a group of tests performing better than other. It is important to remark that a single VM runs on a physical node, implying that each execution of the benchmark in one VM cannot be influenced by the run in another VM.

Verified that the double behaviour does not rely on variation over time of VMs performance, since no patterns are evident in Fig. 4.1a, Fig. 4.5 shows the dispersion of measurements for each VM using boxplots. The plot does not show a clear division between VMs that systematically perform better than others. However, the plot indicates that the overall set of 237 VMs are dominated by a large dispersion of values.

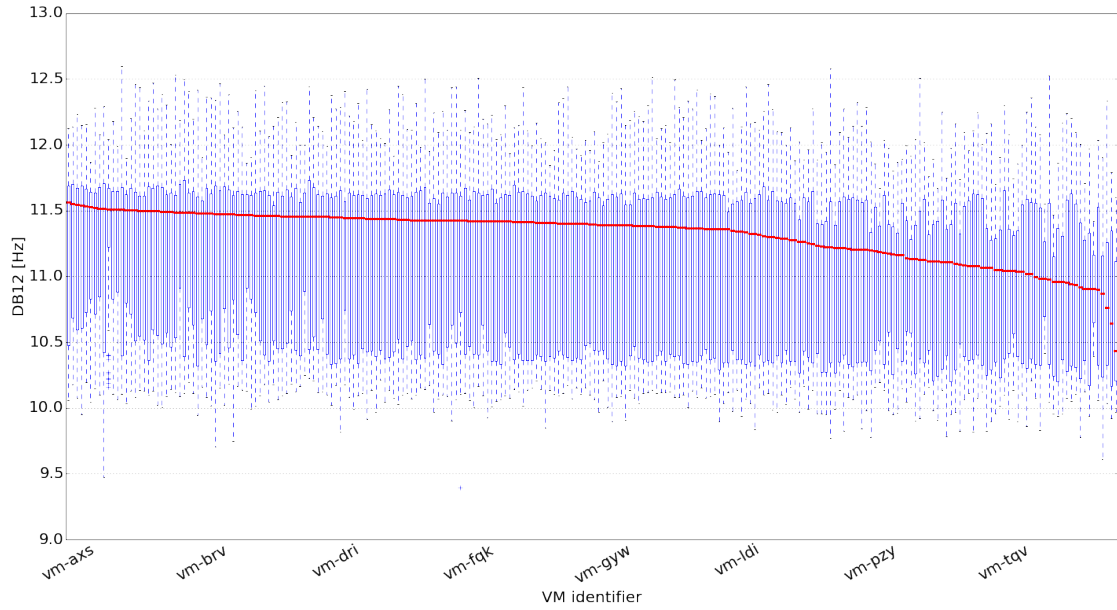


Figure 4.5: The boxplots show the dispersion of the results of the DB12 with respect to the VM in which they run. Candles are ordered by median value.

In order to understand the reason for the double profile, the measurements are analysed on a test VM over time and on a test time interval over VMs. Then the procedure is generalised at the entire set of VMs and time intervals. Firstly, Fig. 4.6 shows DB12 values over time for a specific VM. The plot visualises two distinct clusters separated at a benchmark value of approximately 11 (which corresponds to the DB12 distribution mean) and each cluster averages near the peaks of the bimodal distribution in Fig. 4.2. Therefore the VM, depending on the hour at which DB12 is running, has two different performances. In addition, Fig. 4.7 shows, for each hour, the dispersion of DB12 results over VMs. This plot also clearly displays that DB12 values can be clustered into two groups, separated at approximately 11, which corresponds to the DB12 distribution mean (Fig. 4.2). The particular cases in Fig. 4.6 and Fig. 4.7 highlight that the effect for the DB12 workload is driven nor by the time interval nor by the physical node, but it influences every physical node in every time interval.

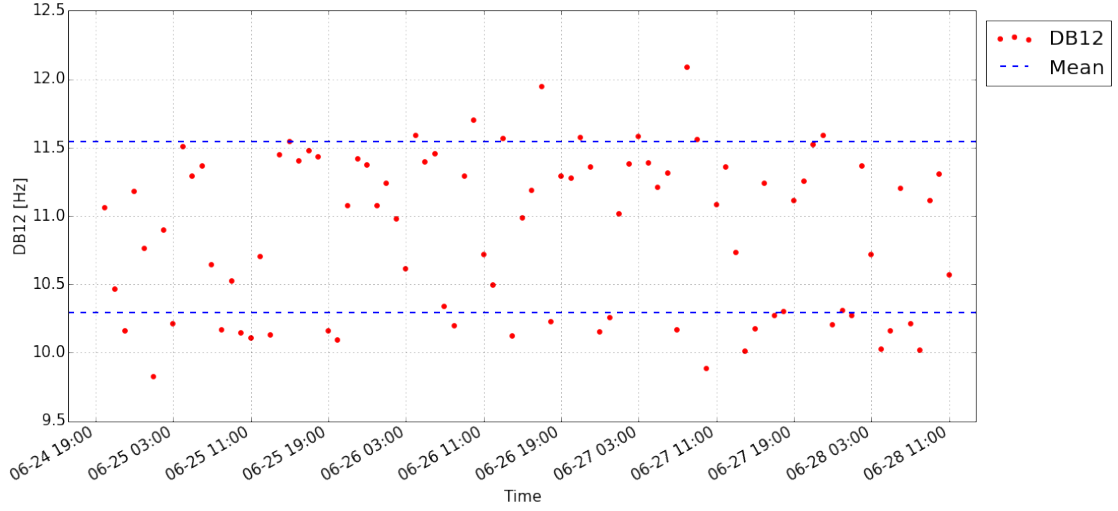


Figure 4.6: The boxplots show the dispersion of the results of the DB12 in one VM with respect to the half hour at which they run.

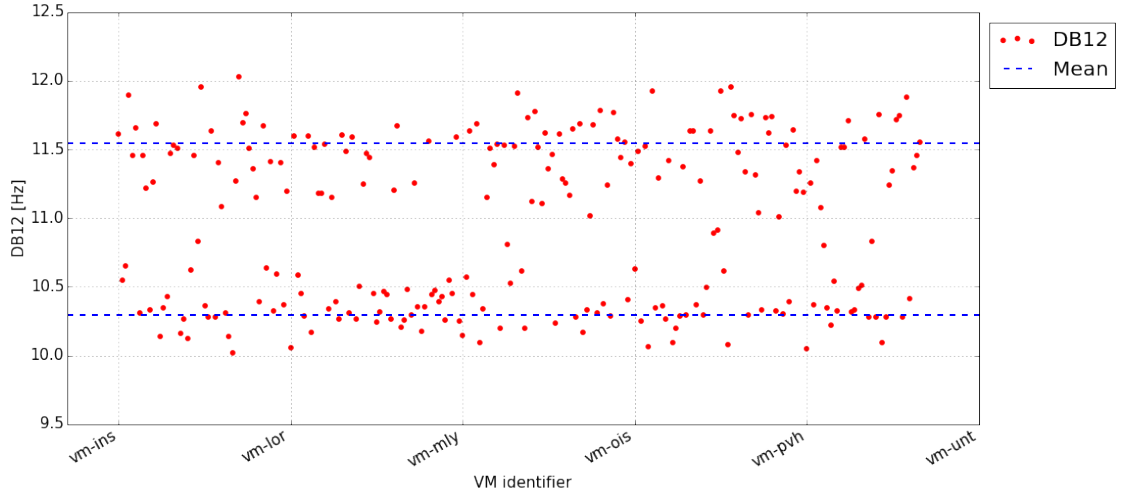


Figure 4.7: The boxplots show the dispersion of the results of the DB12 in a specific time interval with respect to the VM at which they run.

In order to verify if the peculiar results apply to the overall set of VMs, Fig. 4.8 relates the DB12 values with a lag plot. Four regions are evident as vertices of a rectangle, whose orthogonal projection retrace the distribution in Fig. 4.2. Therefore, the shape of the lag plot confirms the fluctuation of DB12 results over time: when a VM's result falls in the 10 range, its successor can fall again in the 10 or in the 11.5 range. The specular behaviour is depicted at the right part of the plot.

The reason for the VM's double profile has not been further analysed, but can be due to an additional process running on the physical node. The same consideration has been applied to WSN, due to the presence of a structure towards higher values in Fig. 4.3. All of the 237 VMs fluctuate between higher and lower performance. However, if for the DB12 benchmark fluctuation happens with 50% probability, for

the WSN fluctuation occurs only 3% of the times. Therefore, it is not interesting for the ongoing analysis.

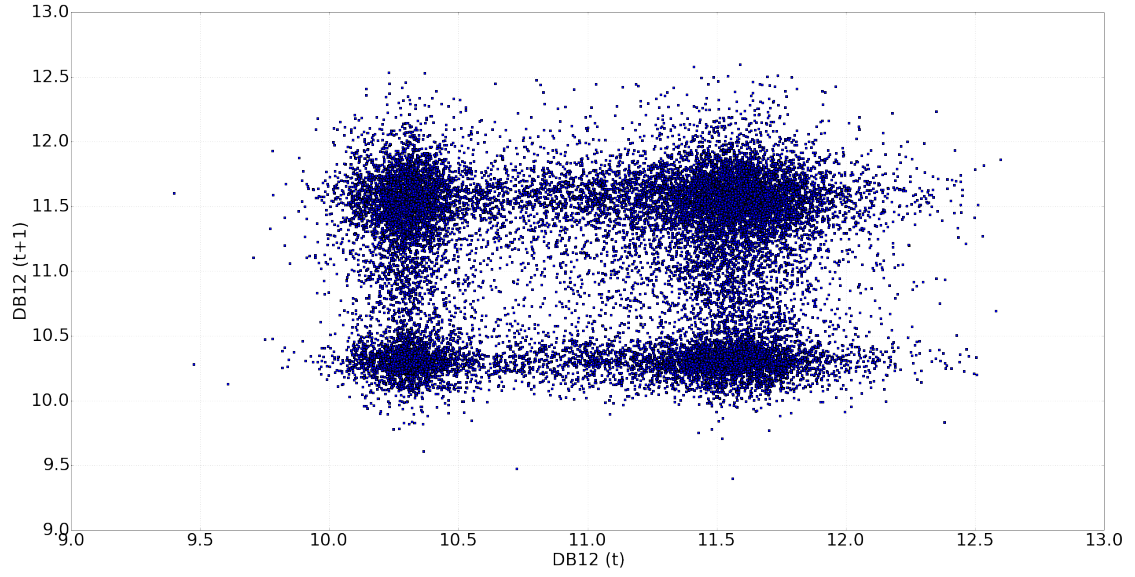


Figure 4.8: The lag plot shows the dispersion of the DB12 results correlating an entry with its successor in the VM-32 configuration.

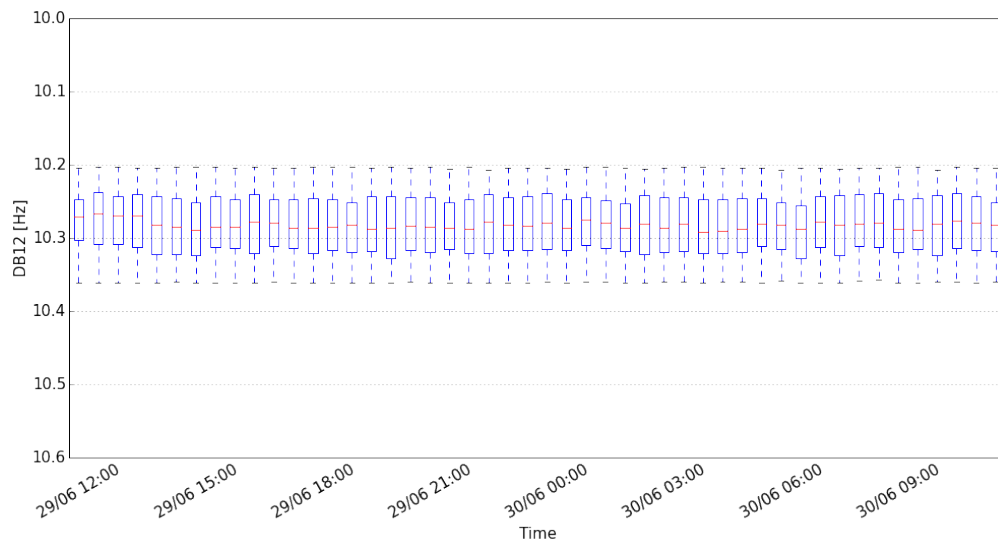
4.2.2 VM-16

VM-16 represents the configuration in which two VMs with 16 vCPUs each are provisioned in each physical node. Table 4.2 summarizes the total running time and the overall number of tests performed.

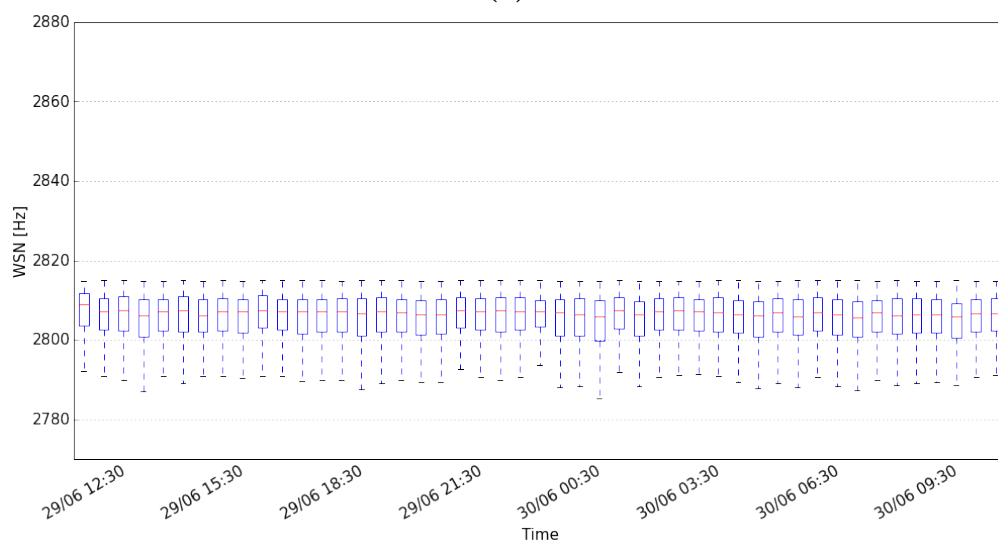
Running period	45 hours
Tests per hour	6
Number of physical nodes	238
Total number of benchmark executions	~ 78 000

Table 4.2: Test summary for configuration VM-16.

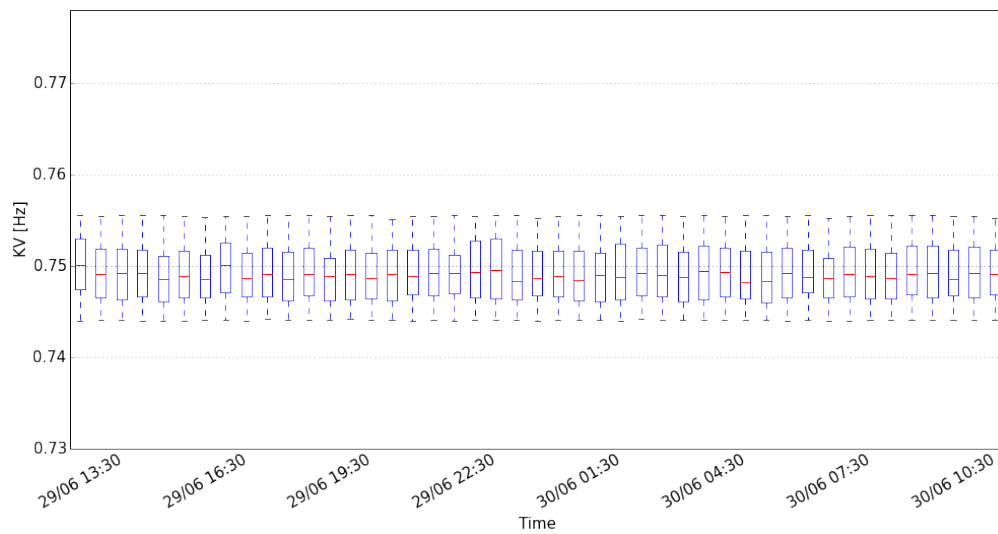
Figure 4.9 shows the time distribution of benchmark results for all of the VMs, summarising as boxplots the dispersion of values for each half hour. The measurements were uniform over time, apart from some outliers for the WSN benchmark, as suggested by the pronounced whiskers in Fig. 4.9b. The uniformity over time of benchmark results motivates the usage of the full set of data for further analysis.



(a)

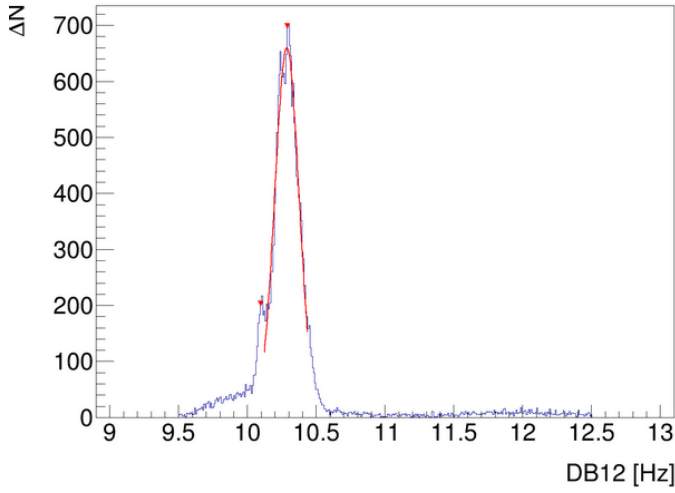


(b)



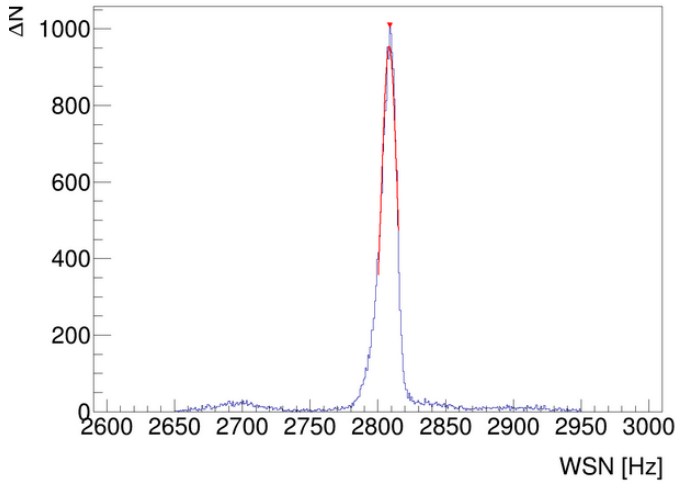
(c)

Figure 4.9: Hourly dispersion of the test results summarised as a boxplot, for the DB12 (a), for the WSN (b) and for the KV (c) respectively in the VM-16 configuration.



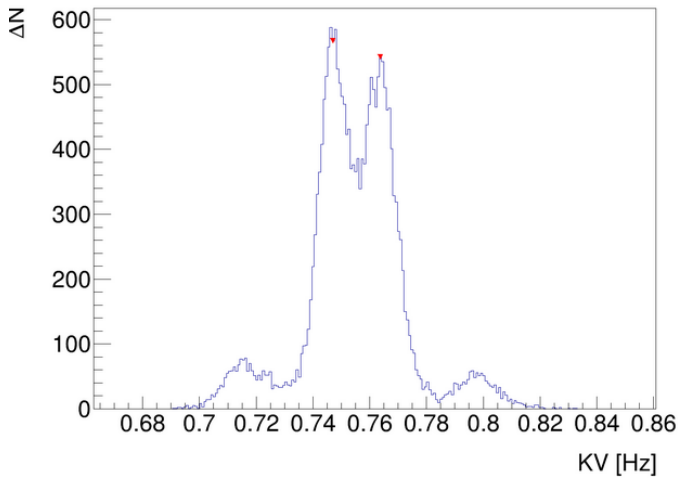
Statistic	
Entries:	21 898
m:	$10.337 \pm 3 \times 10^{-3}$
s:	0.39
mpv:	10.34
Gaussian Fit	
χ^2/ndf :	118.61/34
μ :	$10.2876 \pm 8 \times 10^{-4}$
σ :	$0.0878 \pm 9 \times 10^{-4}$

Figure 4.10: Distribution of the DB12 results in the VM-16 configuration.



Statistic	
Entries:	20 954
m:	$2809.6 \pm 2 \times 10^{-1}$
s:	35.64
mpv:	2809.26
Gaussian Fit	
χ^2/ndf :	40.06/15
μ :	$2808.57 \pm 8 \times 10^{-2}$
σ :	$5.8 \pm 1 \times 10^{-2}$

Figure 4.11: Distribution of the Whestone results in the VM-16 configuration.



Statistic	
Entries:	22 144
m:	$0.7548 \pm 1 \times 10^{-4}$
s:	0.017
mpv:	0.747
mpv:	0.763

Figure 4.12: Distribution of the KV results in the VM-16 configuration.

Figure 4.10, Fig. 4.11 and Fig. 4.12 visualise, for the DB12, WSN and KV respectively, the distribution of the benchmark results. Moreover, the annexed table reports the mean (m), the root mean square (s), the most probable value (mpv) and the Gaussian fit parameters of the distribution. The DB12 (Fig. 4.10) and WSN (Fig. 4.11) results are symmetrically distributed around a central value, apart from outliers irrelevant for the ongoing study. Instead, the KV benchmark results in Fig. 4.12 show two substructures symmetric with respect to the mean value. The double substructure implies a difference over the performance of the VMs or a hidden non-uniform profile over time. In order to understand the double behaviour, Fig. 4.13 shows the dispersion of KV benchmark results grouped by VM. The plot clearly visualises a performance difference between VMs: the candles above the gap identify the set of VMs whose benchmark tests perform better, whereas the candles below the gap identifies the set of VMs whose benchmark tests perform worse. Therefore, the overall set of VMs has been split into two different classes with respect to their profile: the class of VMs that performs better is referred to as VM_{fast} , whereas the class of VMs that performs worse is identified by the keyword VM_{slow} .

Figure 4.14 shows the overlapped distribution of KV results for the VM_{fast} and VM_{slow} categories. Each category is also plotted in Fig. 4.15 and Fig. 4.16 where statistical values and Gaussian fit results are reported.

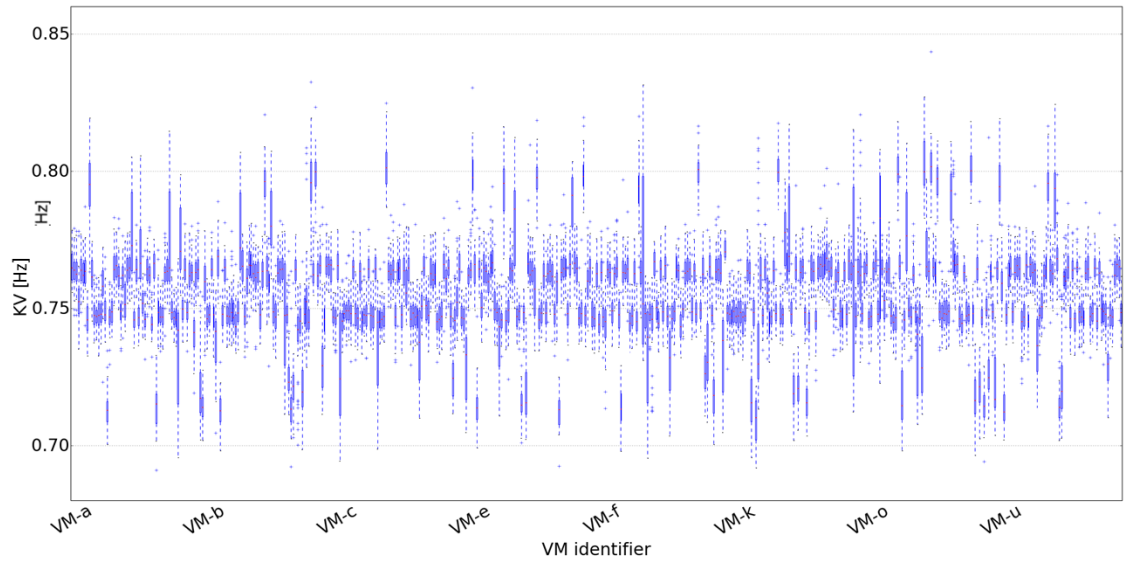
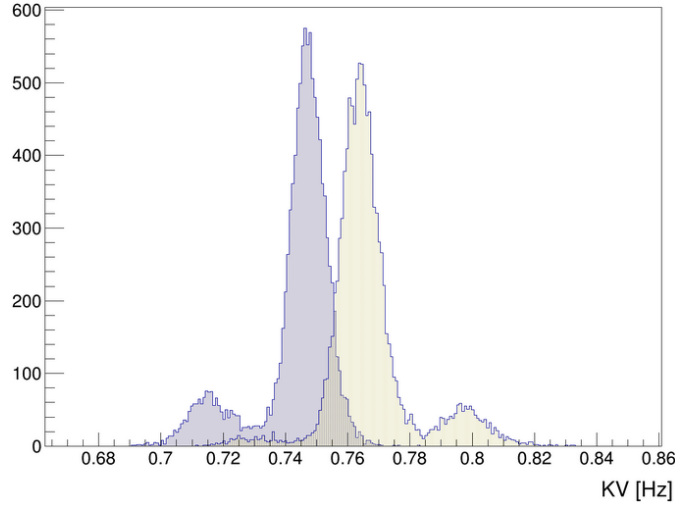
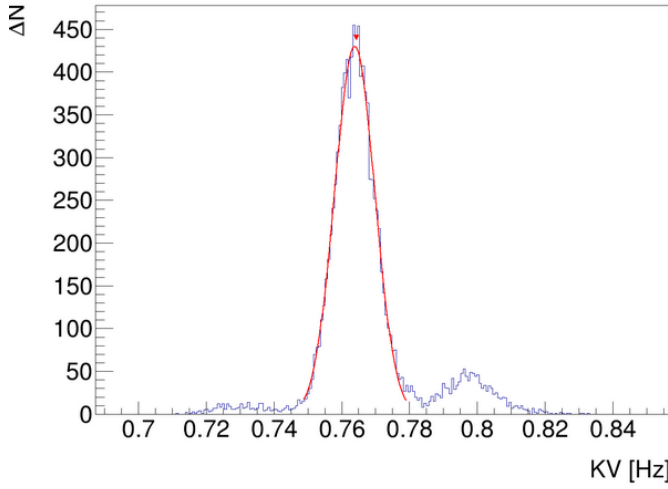


Figure 4.13: Dispersion of the KV benchmark's results with respect to the VM in which they run summarised as a boxplot. The plot clearly highlights the existence of the VM_{fast} and VM_{slow} classes.



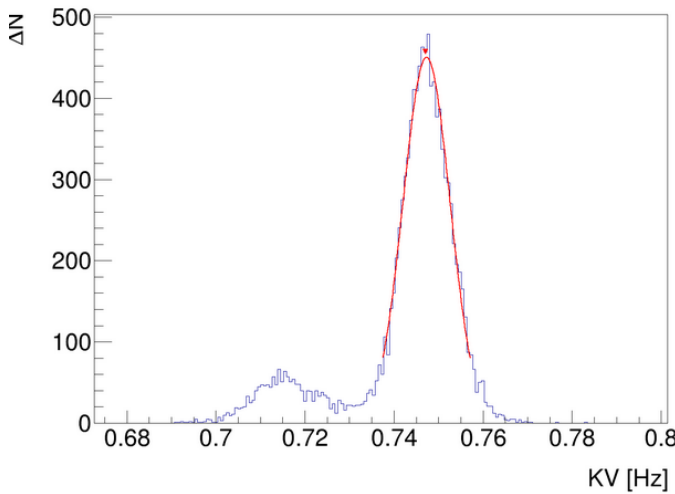
Statistic	
Entries:	22 390
m:	$0.7548 \pm 1 \times 10^{-4}$
s:	0.018
mpv:	0.747
mpv:	0.764

Figure 4.14: Distribution of the KV benchmark's results for the VM_{fast} (light shadow) and VM_{slow} (dark shadow) classes in the VM-16 configuration.



Statistic	
Entries :	9 314
m:	$0.76378 \pm 6 \times 10^{-5}$
s:	0.00607
mpv:	0.7598
Gaussian Fit	
χ^2/ndf :	50.4614/41
μ :	$0.76389 \pm 6 \times 10^{-5}$
σ :	$0.00591 \pm 5 \times 10^{-5}$

Figure 4.15: Distribution of the KV results in the VM-16_{fast} class.



Statistic	
Entries:	9 518
m:	$0.74732 \pm 5 \times 10^{-5}$
s:	0.0057
mpv:	0.7445
Gaussian Fit	
χ^2/ndf :	37.40901/27
μ :	$0.74734 \pm 7 \times 10^{-5}$
σ :	$0.00527 \pm 7 \times 10^{-5}$

Figure 4.16: Distribution of the KV results in the VM-16_{slow} class.

The relative performance increase of VM_{fast} with regard to VM_{slow} in each physical node is reported, for each test performed, in Fig. 4.17. The average value of the distribution, excluding the outliers at approximately 8%, is about a factor of 2%. Figure 4.18 shows that the performance difference happens in approximately 97% of the performed tests. It should be remembered that, in VM-16 configuration, each physical node hosts two VMs, whose vCPUs has been configured via libvirt to run on a single NUMA node. Therefore, due to the configuration of the VM-16 scenario, a systematic performance difference on VMs implies a systematic performance difference on CPUs. Furthermore, it has been verified that each server hosts a VM that belongs to the VM_{fast} class and the other that belongs to the VM_{slow} class. In particular, an in depth analysis verified that the VMs in the VM_{slow} class were systematically running on the second physical CPU of the dual-socket server.

Further analysis, done by CERN-IT, directly on the bare-metal server, at the level of the CPU logical threads, has highlighted a slower performance on the threads belonging to the first physical core of the second CPU, vCPUs 8 and 24 [69]. In particular, the threads show an increase of 16% in the final average simulation time, that explains 2% spread between VMs when averaged over 8 virtual processors. The analysis has been conducted firstly, on the bare-metal server with SMT disabled setting scheduling affinity sequentially to each physical core, then confirmed enabling SMT. To verify the performance issues, the CERN team focused on the code executed by the KV benchmark using the Linux perf [86] profiler tool¹. It is a profile application used to trace the performance counters² for the dynamic control flow. Details about the investigation process are available in a CERN note [87].

¹perf is a tool included in the Linux kernel capable of lightweight profiling that provides generalized abstractions over hardware specific capabilities.

²Performance counters are CPU hardware registers that count hardware events such as instructions executed, cache-misses suffered, or branches mispredicted.

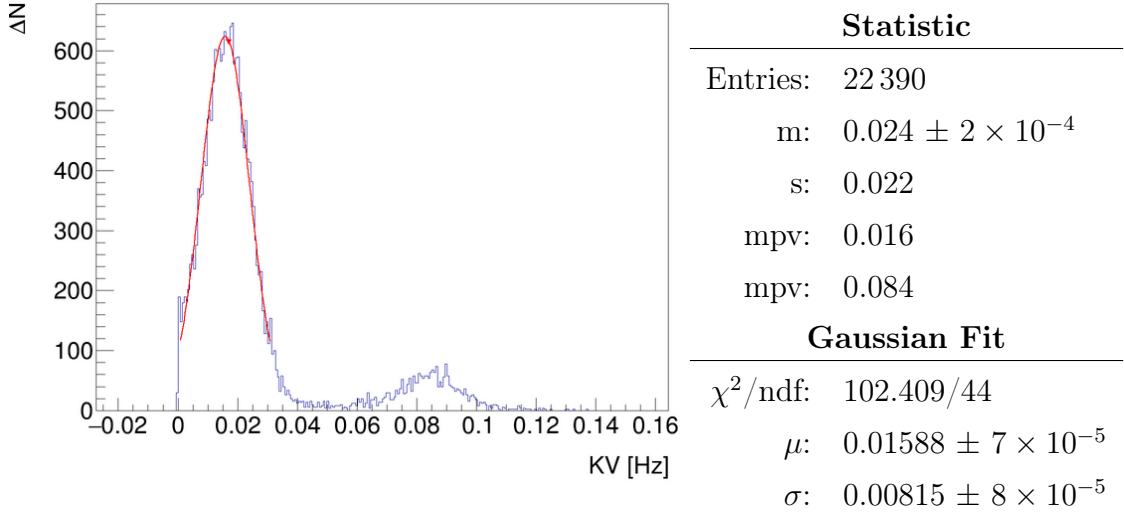


Figure 4.17: Difference between the benchmark results of the VM_{fast} and VM_{slow} classes in the VM-16 configuration.

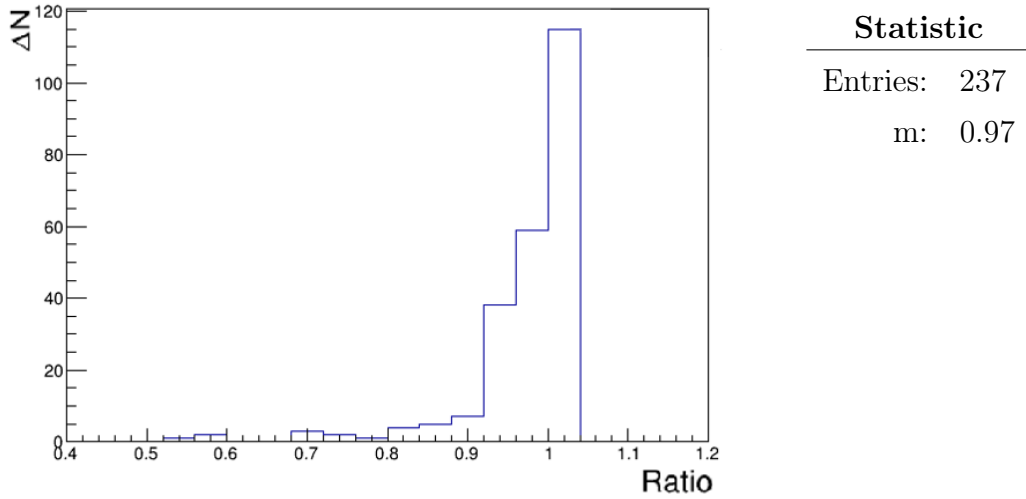


Figure 4.18: Distribution of the percentage of benchmark tests in which the same VM performs faster than its sibling in the VM-16 configuration.

In the case of DB12 and WSN, the bimodal Gaussian effect was negligible. The absence of different classes of performance has been verified using the same approach of the KV benchmark and the results are shown in Fig. 4.19 for the DB12 benchmark and Fig. 4.20 for the WSN benchmark. For both benchmarks, the performance difference between CPUs is lower compared to the KV workload. In particular, Fig. 4.19b shows a performance difference of about 0.8% and Fig. 4.19c shows that the percentage of times in which a CPU is systematically faster than its sibling is only about 65%. The same consideration can be made for the WSN workload in Fig. 4.20. The performance difference between CPUs is about 0.4% (Fig. 4.20b), whereas the frequency at which one CPU is systematically faster than the sibling is about 67% (Fig. 4.19c).

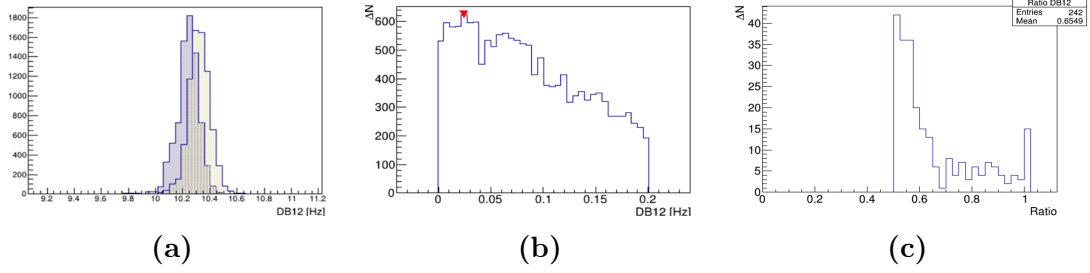


Figure 4.19: Distribution of the DB12 results, in the VM-16 configuration, for the VM_{fast} and VM_{slow} classes (a). Distribution of the relative performance improvement of VM_{fast} compared to VM_{slow} (b). Histogram (c) shows the fraction of tests having a VM faster than the sibling one.

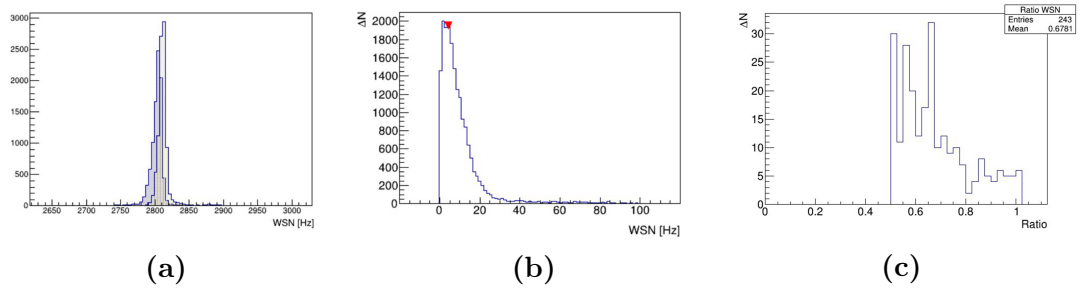


Figure 4.20: Distribution of the WSN results, in the VM-16 configuration, for the VM_{fast} and VM_{slow} classes (a). Distribution of the relative performance improvement of VM_{fast} compared to VM_{slow} (b). Histogram (c) shows the fraction of tests having a VM faster than the sibling one.

4.2.3 VM-8

VM-8 represents the configuration in which 4 VM with 8 vCPUs are each provisioned in each physical node. Table 4.3 summarizes the total running time and the overall number of tests performed.

Running period	147 hours
Tests per hour	6
Number of physical nodes	239
Total number of benchmark executions	~ 230 000

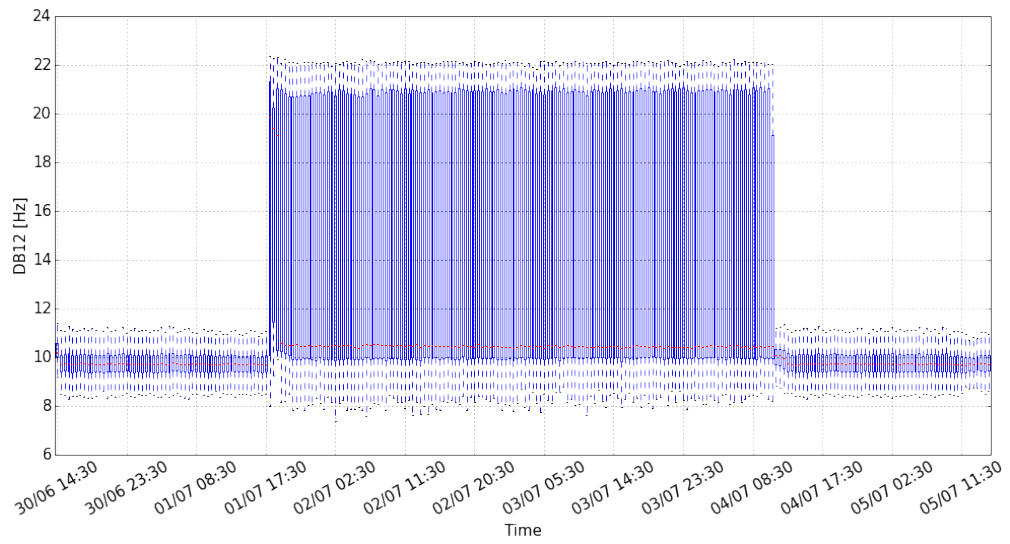
Table 4.3: Test summary for configuration VM-8.

Figure 4.21 shows the time distribution of benchmark results for all of the VMs, summarising as boxplots the dispersion of values for each half hour. Each plot is characterised, on average, by a constant profile over time that can be split into three categories. The initial and final phases visualise candles characterised by a small difference between quantiles and symmetric whiskers. Between the initial and final

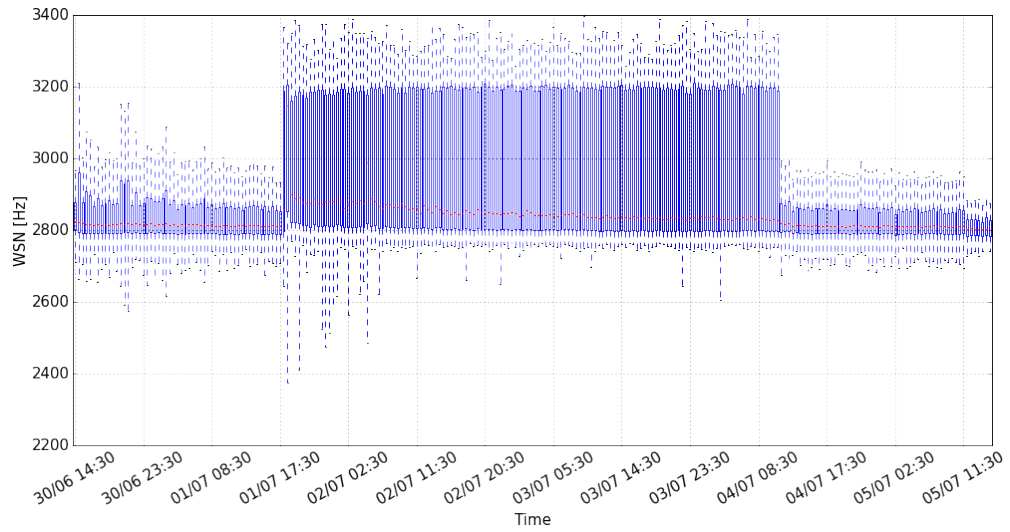
phase there is a transition phase, characterised by a significant difference between quantiles. In particular, the 25th percentile is close to the average behaviour of the benchmarks during the initial and final phase, whereas the 75th percentile is significantly higher. The difference between quantiles in the transition phase is justified by a variation of the scenario. During the run, the deletion of a single VM on the physical nodes introduced a new configuration, as a category of the VM-8 configuration, called VM-8₃ during the analysis phase. The VM-8₃ configuration is characterised by the presence of 3 VMs of 8 vCPUs on each physical host. One CPU hosts only one VM, which can benefit from the CPU resources without the influence of its sibling, improving, consequently, its performance during the benchmark execution. Instead, the other CPU hosts two VMs that must share the CPU resources, causing lower performance results. Consequently, the new scenario introduces a performance difference between the two CPUs that is reflected in the significant variability between quantiles during the transition phase. In order to simplify the nomenclature adopted during the analysis phase, the keywords used to identify the two sets of VMs in the VM-8₃ scenario are introduced. The VM_{single} is the flag used for the cluster of VMs running alone on the CPU, while the VM_{couple} defines the flag for the VMs that work with a sibling on the CPU.

Figure 4.22, Fig. 4.23 and Fig. 4.24 visualise, for the three types of benchmarks, the distribution of the benchmark results. The plots share a common structure with two distributions. The distribution on the left identifies the set of VMs that perform worse due to the presence of a sibling on the CPU. Instead, the curve on the right, with higher results, identifies the set of VMs that run alone on the CPU. It is interesting to notice the scaling factor between the peak of the two curves in Fig. 4.22 and Fig. 4.24 for the DB12 and KV benchmarks respectively. It is about a factor of 2, indicating that the VM alone on the server performs approximatively two times better than the one with a sibling on the socket. Also the WSN benchmark in Fig. 4.23 identifies a pattern characterized by a curve with two peaks, but the scaling factor between the two clusters is less than a factor of 2. Furthermore, interesting consideration can be introduced for the KV benchmark (Fig. 4.24). The curve on the right, which defines the VMs performing alone on the socket, shows a double substructure. It implies a double behaviour for the VMs executing alone on the CPU, that has the same justification found for VM-16.

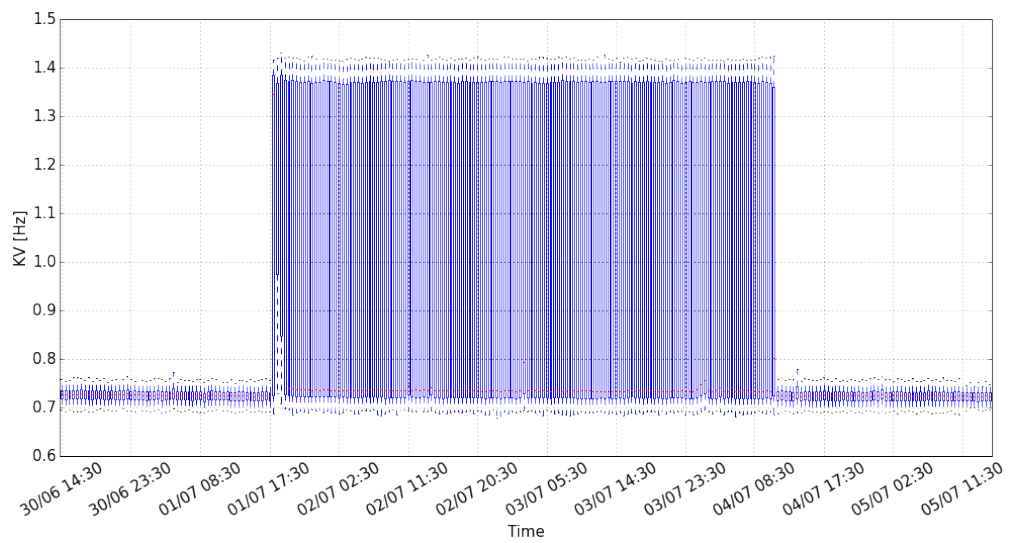
In order to correctly evaluate the distribution of the VM-8 configuration, the dataset has been divided into two different clusters. One that represents the set of VMs belonging to the configuration with four VMs on each physical node, defined as VM-8₄, the other that represent the configuration with three VMs on each physical node, defined VM-8₃. The division algorithm has been performed on the number of VMs on each server at each half hour.



(a)



(b)



(c)

Figure 4.21: Hourly dispersion of the test results summarised as boxplot, for the DB12 (a), for the WSN (b) and for the KV (c) respectively in the VM-8 configuration.

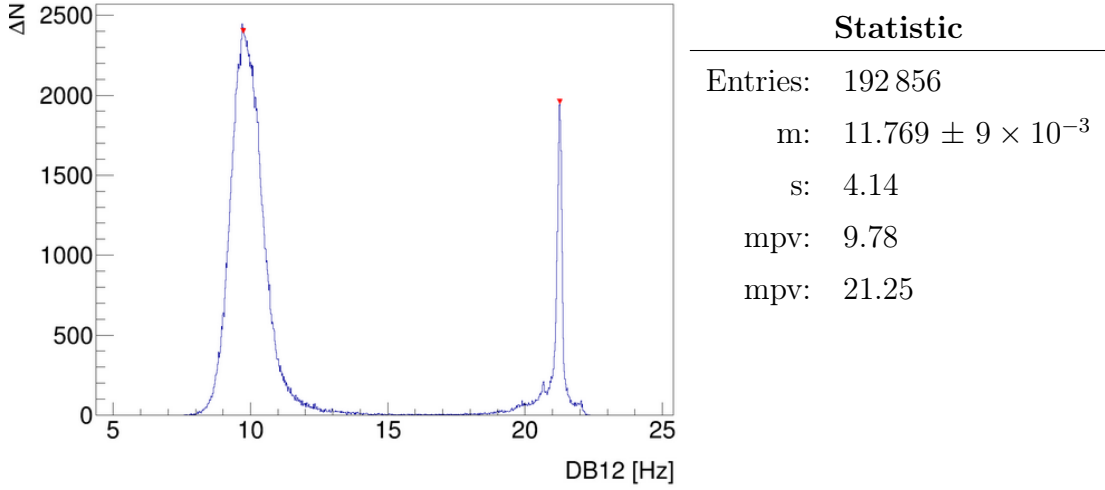


Figure 4.22: Distribution of the DB12 results in the VM-8 configuration.

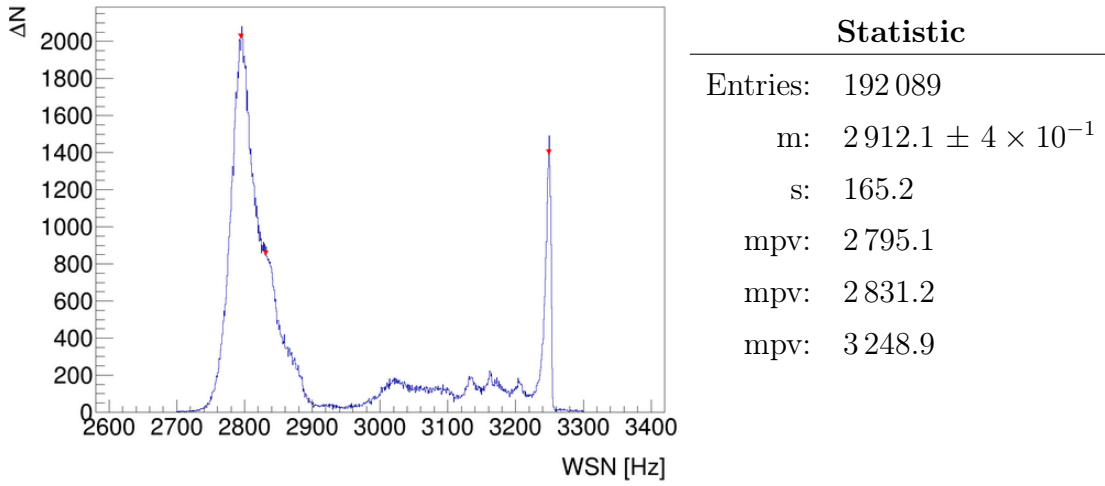


Figure 4.23: Distribution of the WSN measurements in the VM-8 configuration.

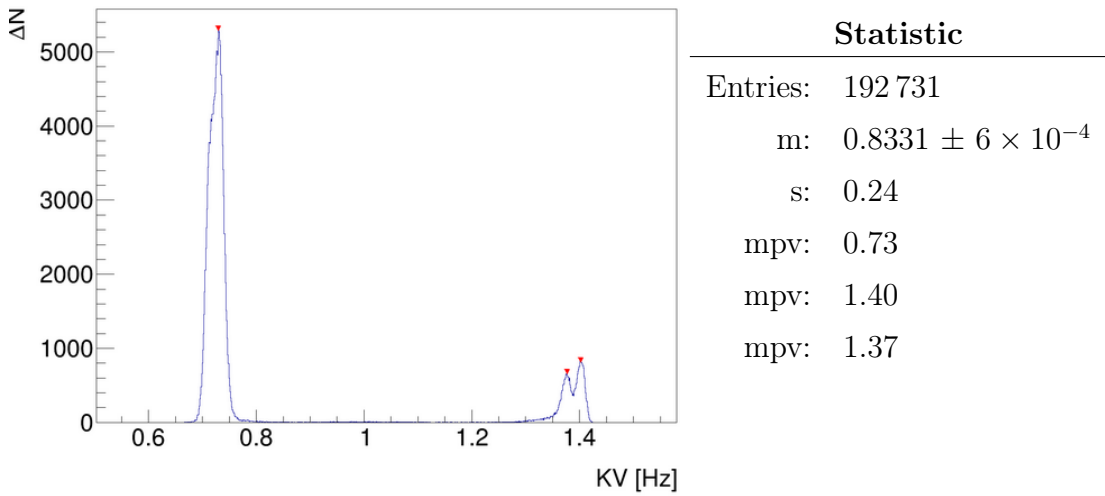


Figure 4.24: Distribution of the KV measurements in the VM-8 configuration.

VM-8₄ configuration The VM-8₄ configuration identifies the configuration with 4 VMs with 8 vCPUs each. Graphically, the configuration operates during the initial and final phase (Fig. 4.21c) where the spread between quantiles is small.

Figure 4.25, Fig. 4.26 and Fig. 4.27 visualise, for the three types of benchmarks, the distribution of the tests results. The DB12 results (Fig. 4.25) are distributed as a Gaussian, the WSN (Fig. 4.26) measurements distribute as a Gaussian apart from whiskers not relevant for the ongoing study. Finally, the KV workload (Fig. 4.27) shows a curve with two substructures, as in the VM-16 scenario (see Section 4.2.2). Therefore, following the consideration made about the CPUs performance difference for the VM-16 configuration, it is worth verifying if the performance discrepancy of the CPUs also applies for the VM-8₄ configuration. The analysis is shown only for the KV workload, but it has been performed also for the DB12 and the WSN.

As for the VM-16 scenario, in order to understand the reason for the double behaviour, the tests' results have been grouped by VM. Hence, Fig. 4.28 shows the dispersion of the KV benchmark results for each VM. The plot clearly visualises a performance difference between VMs: the boxplots above the gap identify the set of VMs whose tests perform better (VM_{fast}), whereas the set of VMs below the gap identify the set of VMs that perform worse (VM_{slow}). Therefore, it is possible to divide the overall set of VMs into two different clusters with respect to their profile. In order to split the clusters, the performed strategy is based on the position of the VMs' quantiles with respect to a threshold. If the VMs' 25th percentile is above the threshold it belongs to the VM_{fast} cluster, otherwise it belongs to the VM_{slow} cluster. In order to find the correct KV value to be defined as a threshold, the distribution of the 25th percentile has been plot in Fig. 4.29. The threshold is identified approximatively at about 0.72, that separates the two sub-distributions. The result of the clustering procedure for the KV benchmark is shown in Fig. 4.30. As for the VM-16 scenrio, the KV benchmark shows two substructures, each of which distributes as a Gaussian. Figure 4.31 defines the distribution of the set of VMs that run the tests on the fastest CPU, whereas Fig. 4.32 represents the distribution of the set of VMs that run the tests on the slowest CPU. In particular, also in this configuration, the fastest gains a factor of about 2% in performance over the slowest CPU. For what concerns the DB12 and the WSN, the analysis does not reveal any double substructure confirming that, on average, the VMs of different sockets perform in the same way for those benchmarks.

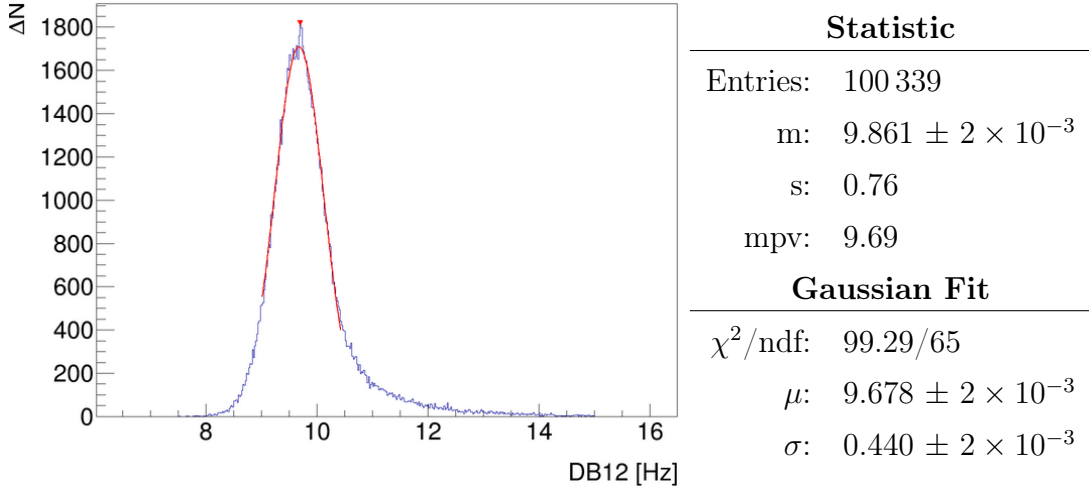


Figure 4.25: Distribution of the DB12 results in the VM-8₄ configuration.

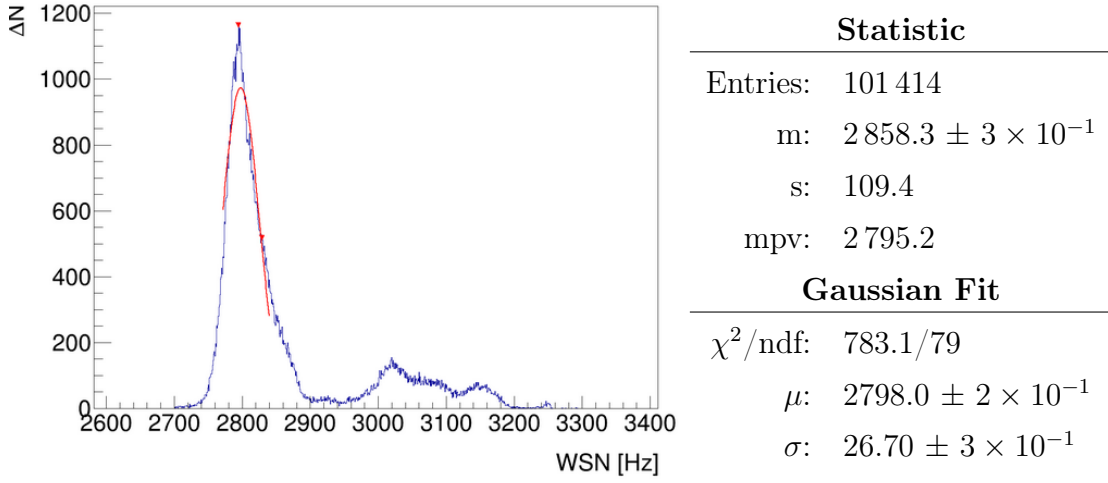


Figure 4.26: Distribution of the WSN results in the VM-8₄ configuration.

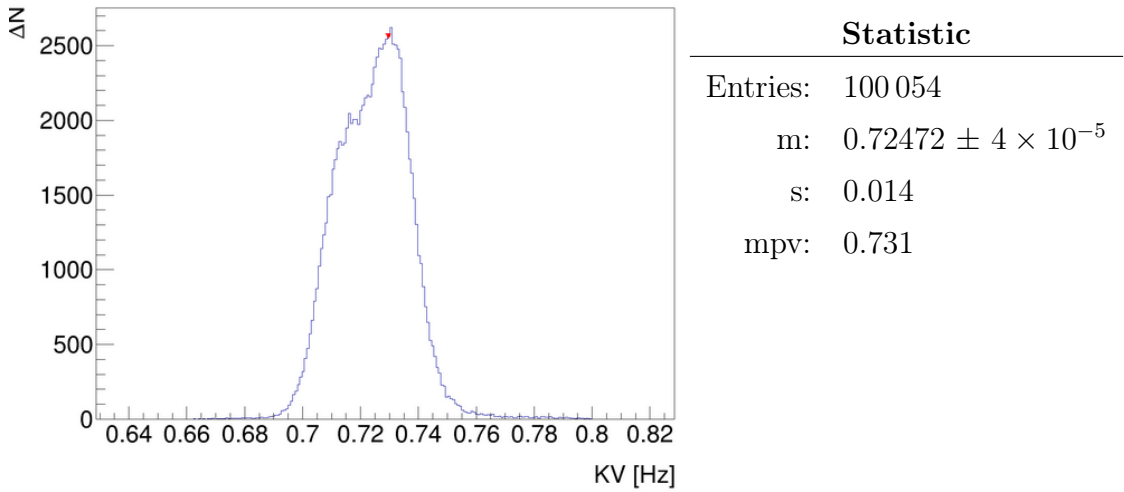


Figure 4.27: Distribution of the KV results in the VM-8₄ configuration.

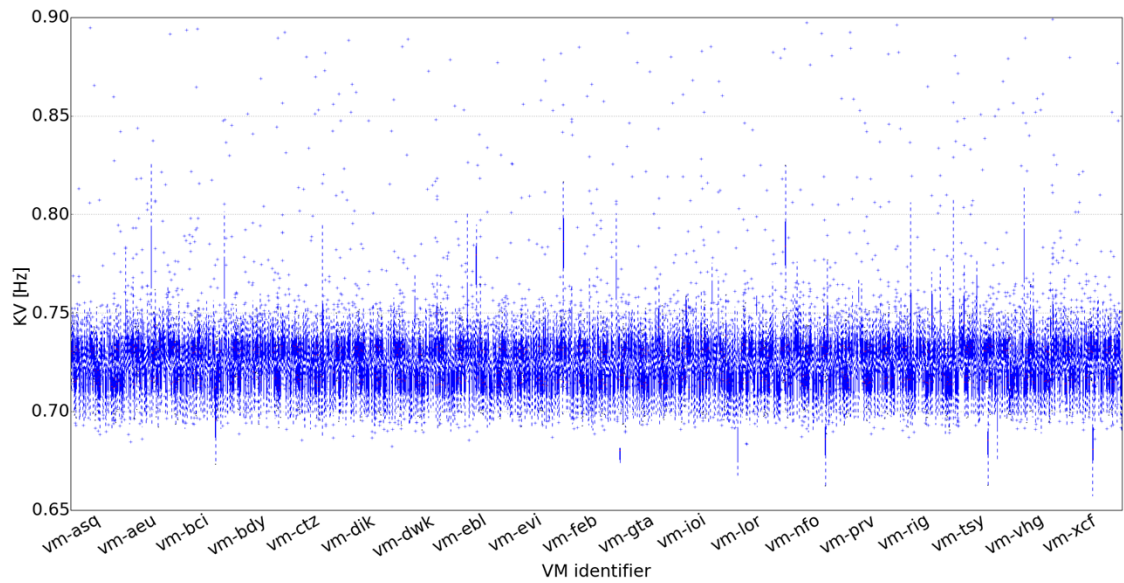


Figure 4.28: Dispersion of the KV results summarised as boxplots with respect to the VM in which they run.

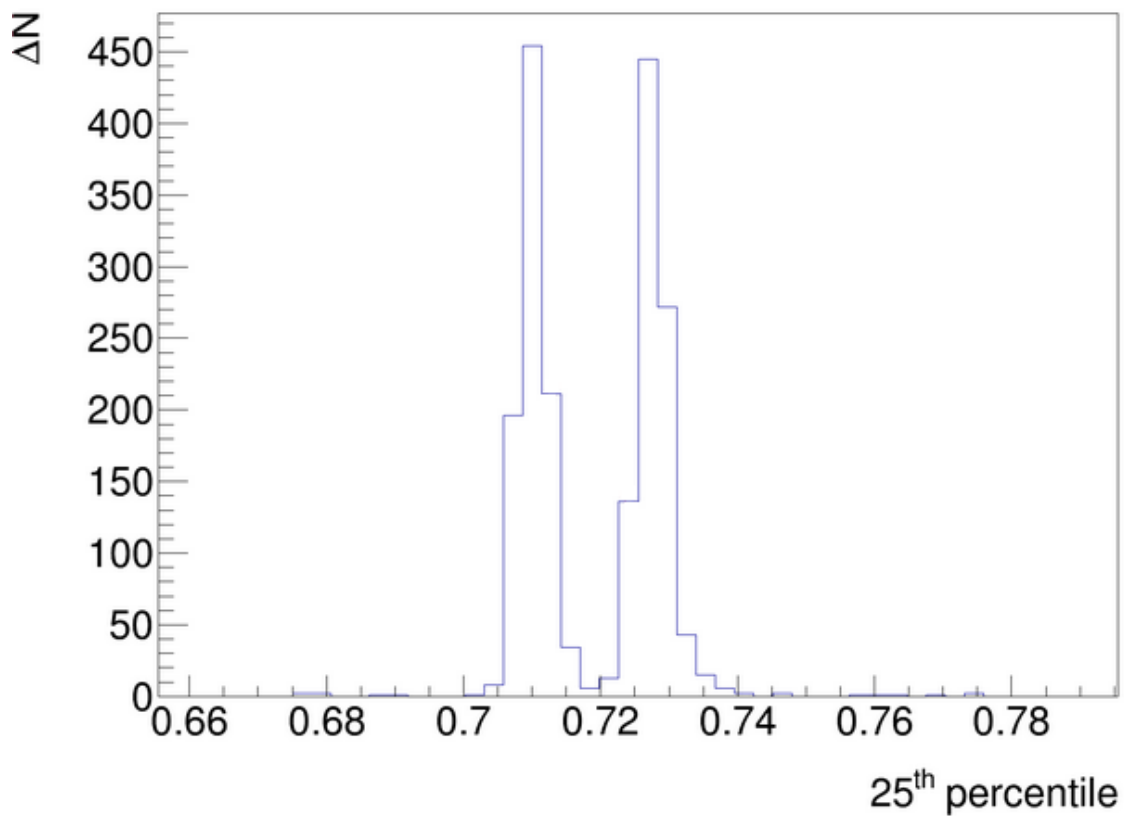


Figure 4.29: Two separate classes are visible and are discriminated using the distribution of the 25th percentile of the KV benchmark.

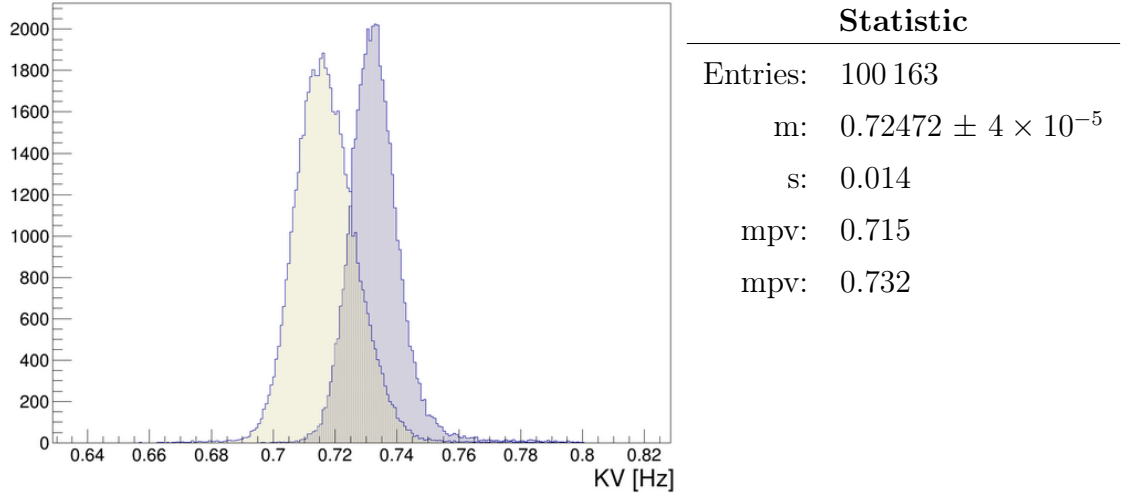


Figure 4.30: Distribution of the KV benchmark's results for VM_{fast} (dark shadow) and VM_{slow} (light shadow) classes in $VM-8_4$ configuration.

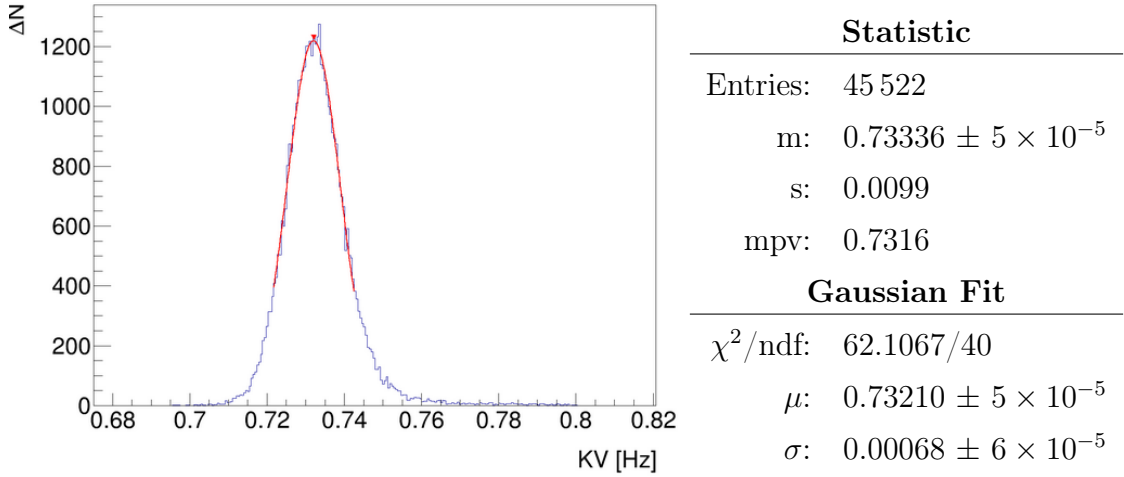


Figure 4.31: Distribution of KV results for the fast cluster in the $VM-8_4$ configuration.

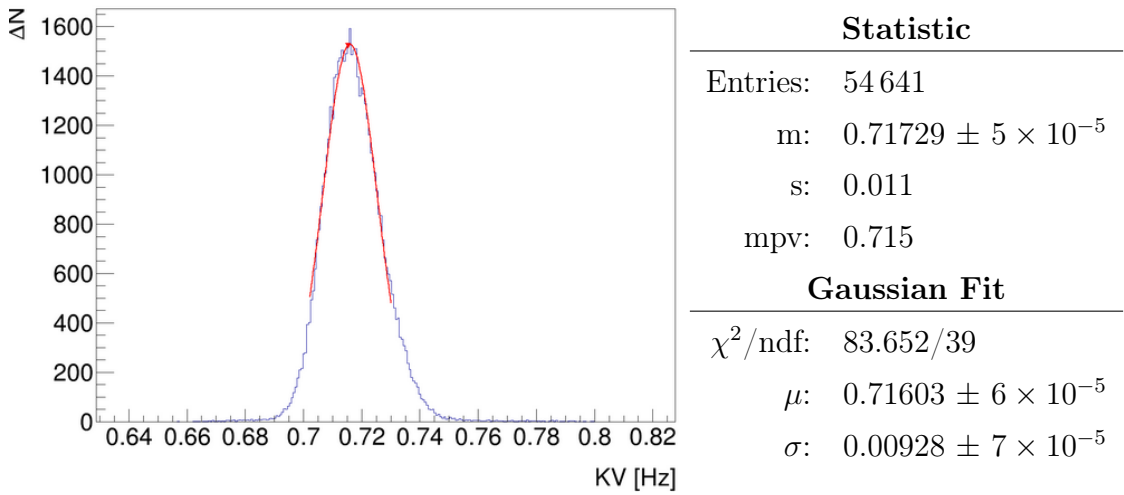


Figure 4.32: Distribution of KV results for the slow cluster in the $VM-8_4$ configuration.

VM-8₃ configuration The VM-8₃ scenario identifies a sub-configuration of the nominal VM-8 configuration. It identifies the configuration in which each server hosts 3 VMs, one that runs alone on a CPU (VM_{single}) and the other two that share the resources of the CPU (VM_{couple}). Figure 4.33 gives a pictorial representation of a possible distribution of the VMs on the physical node; the alternative configuration is characterised by VM_{single} on CPU1.

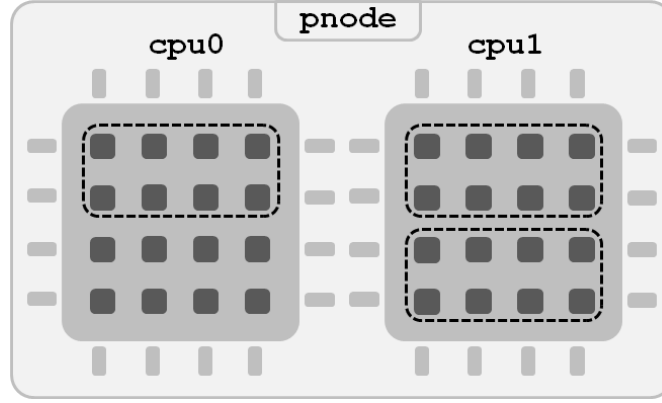
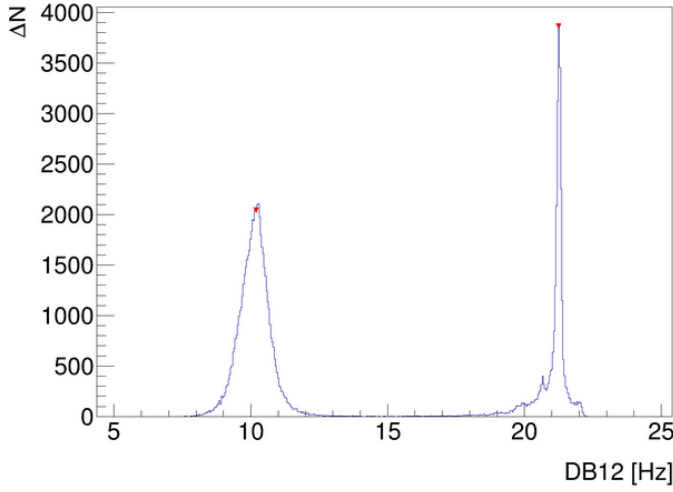


Figure 4.33: Pictorial representation of the VM-8₃ configuration.

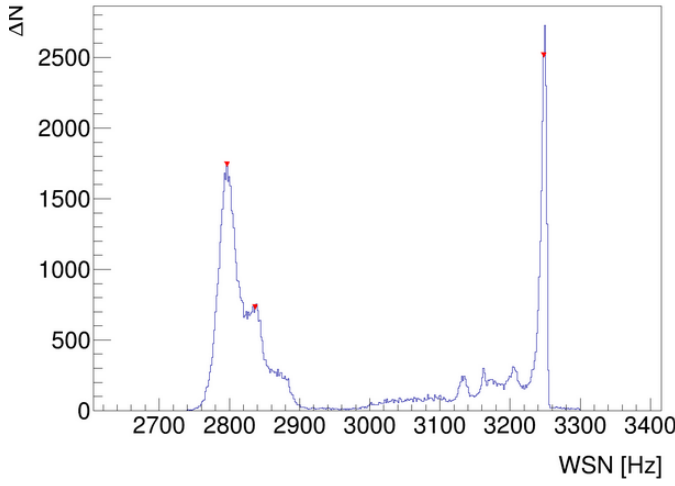
The distributions of the CPUs performance are visualised in Fig. 4.34, Fig. 4.35 and Fig. 4.36, for DB12, WSN and KV respectively. The plots recall, with a lesser number of entries, the distribution plots in Fig. 4.22, Fig. 4.23 and Fig. 4.24. The curve on the left gather the values derived from the set of VMs that execute benchmarks with a sibling on the CPU; whereas, the right curve shows the distribution of the benchmarks results of the VM running alone on the CPU. As for the general distribution, it is particularly interesting to verify that for the DB12 and for the KV benchmark, the ratio between the average performance of the fastest and the slowest CPUs is approximatively a factor of 2. Whereas for the WSN benchmark the proportionality does not apply. In order to perform the analysis, the VM_{single} and VM_{couple} clusters must be studied separately. Since the separation between the curves is clear, the threshold is chosen as a middle value between the two peaks. In the following study only the KV benchmark is reported, even if the study has been performed, with the same approach, for the DB12 and the WSN.



Statistic

Entries:	89 442
m:	$13.83 \pm 2 \times 10^{-2}$
s:	5.18
mpv:	21.26
mpv:	10.25

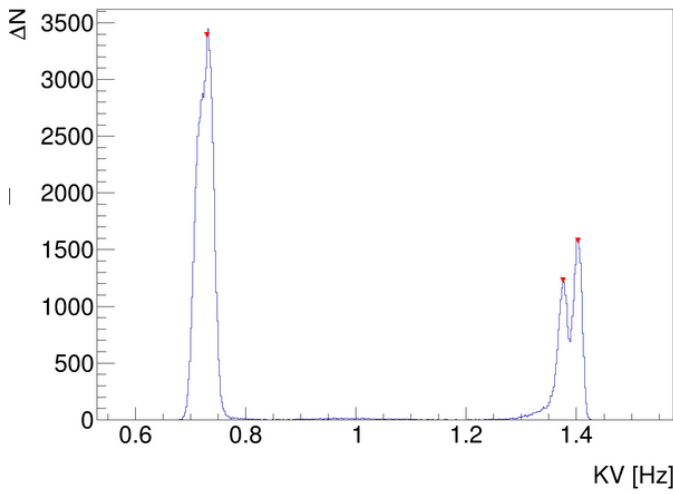
Figure 4.34: Distribution of the DB12 results in the VM-8₃ configuration.



Statistic

Entries:	89 241
m:	$2972.0 \pm 6 \times 10^{-1}$
s:	193.2
mpv:	3 248.9
mpv:	2 795.6
mpv:	2 838.4

Figure 4.35: Distribution of the WSN results in the VM-8₃ configuration.



Statistic

Entries:	89 442
m:	$0.947 \pm 1 \times 10^{-3}$
s:	0.31
mpv:	0.73
mpv:	1.40
mpv:	1.37

Figure 4.36: Distribution of the KV results in the VM-8₃ configuration.

VMs running alone on the CPU - VM_{single} Figure 4.37 shows the distribution of the KV benchmark, highlighting a substructure not evident in the DB12 (Fig. 4.34) and the WSN (Fig. 4.35) benchmarks. The double peak supports the considerations obtained from the VM-16 and VM-8₄ scenarios: the second CPU performs worse than the first CPU when stressed with the KV workload. The reason for such behaviour is due to the random provisioning of VMs on the fast or slow CPU. Obviously, the provisioning on a CPU of the VM_{single} is reflected on the performance of the VM_{couple} set. In order to split the distributions, the same quantile approach of the VM-8₄ configuration has been followed. The cluster approach is again based on a cut over a threshold with respect to the 25th percentile. Figure 4.38 shows the distribution of the KV results for each VM. Two distinct clusters are clearly defined and the threshold is identified approximatively at the value that symmetrically divides the histograms defined by the 25th percentile. In particular, the set of VMs whose 25th percentile is higher with respect to the threshold is the set of VM_{fast} , whereas the set of VMs whose 25th percentile is lower with respect to the threshold is the set of VM_{slow} . The distribution of the two clusters are shaped as two Gaussian in Fig. 4.39 for the VM_{fast} distribution and in Fig. 4.40 for the VM_{slow} . As for VM-16 and VM-8₄, the ratio between the mean of the distributions is approximatively a factor of 2%.

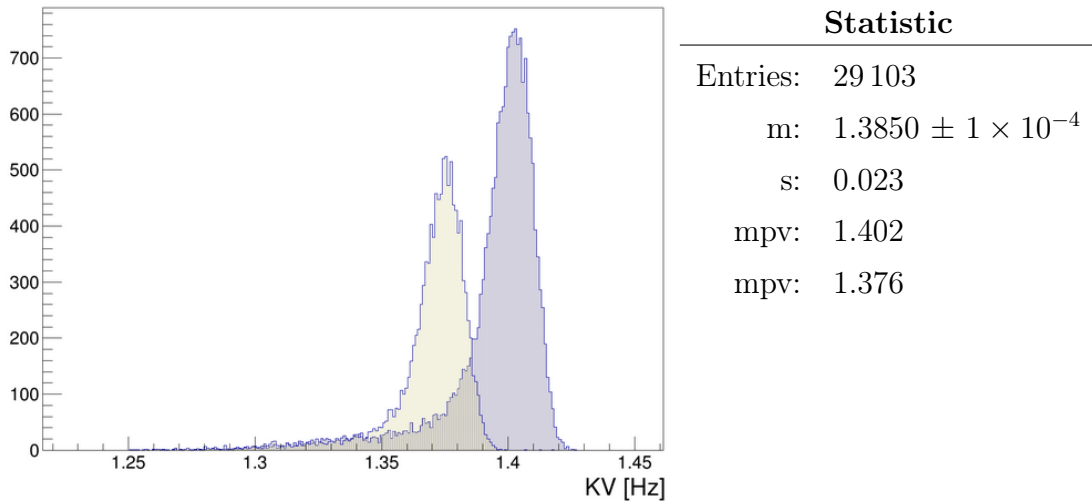


Figure 4.37: Distribution of the KV benchmark's results for VM_{slow} (light shadow) and VM_{fast} (dark shadow) classes in VM-8₃ - single configuration.

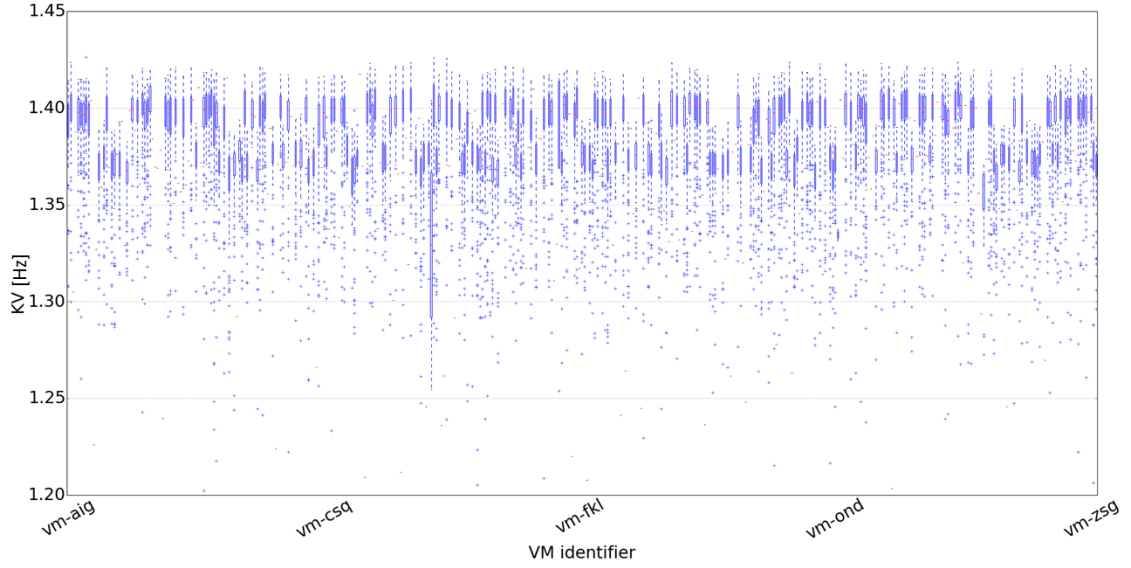


Figure 4.38: Dispersion of the KV results with respect to the VM on which they run.

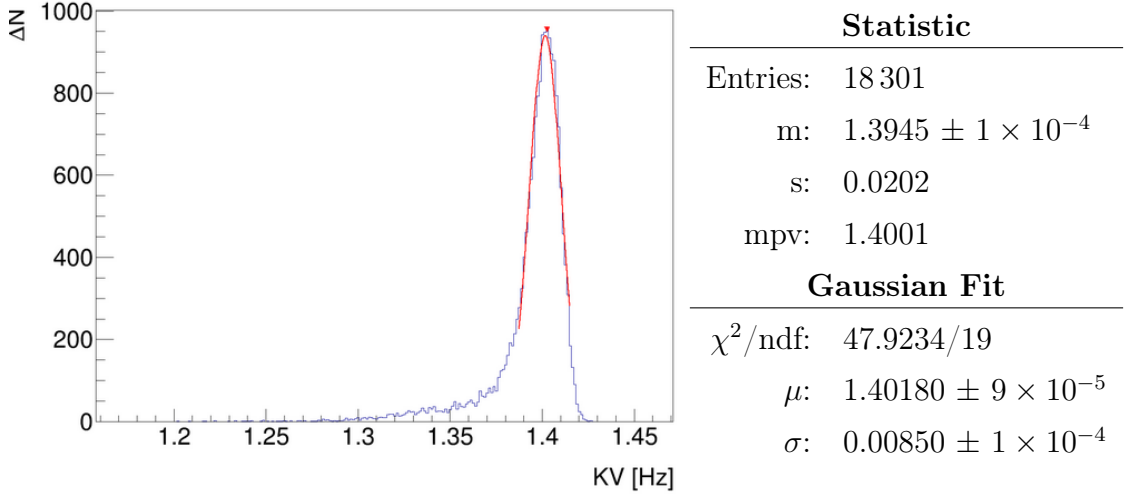


Figure 4.39: Distribution of KV results for the fast cluster in VM-8₃ - single configuration.

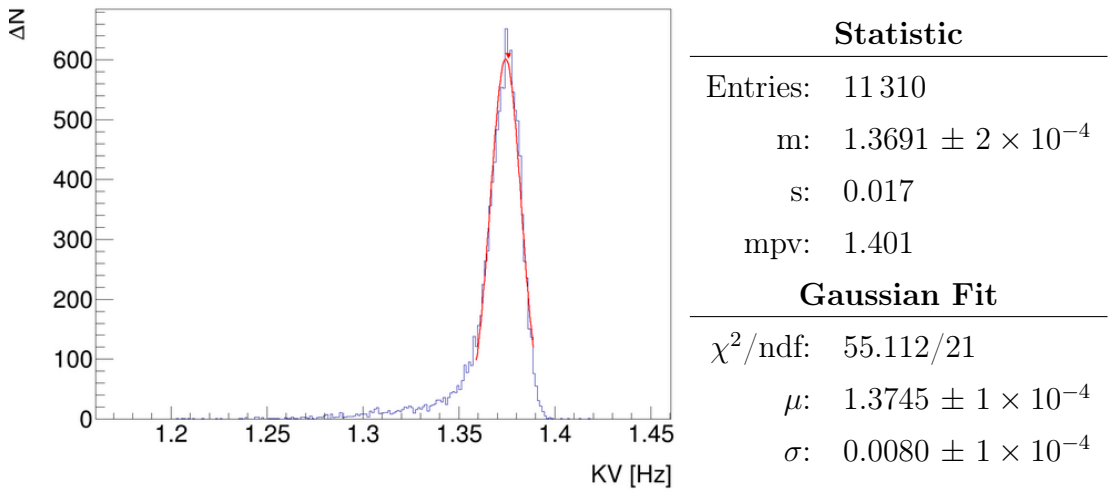


Figure 4.40: Distribution of KV results for the slow cluster in VM-8₃ - single configuration.

VMs running with a sibling on the CPU - $\text{VM}_{\text{couple}}$ The VM-8_3 - couple cluster identifies the VMs that perform on the same CPU in the VM-8_3 scenario. Figure 4.41, Fig. 4.42 and Fig. 4.43 show the distribution of the benchmarks results for the DB12, WSN and KV respectively,. As expected, the distribution profile is similar to the profile of the VM-8_4 distribution. Moreover, Fig. 4.43 for the KV the distribution has a double structure that has been analysed.

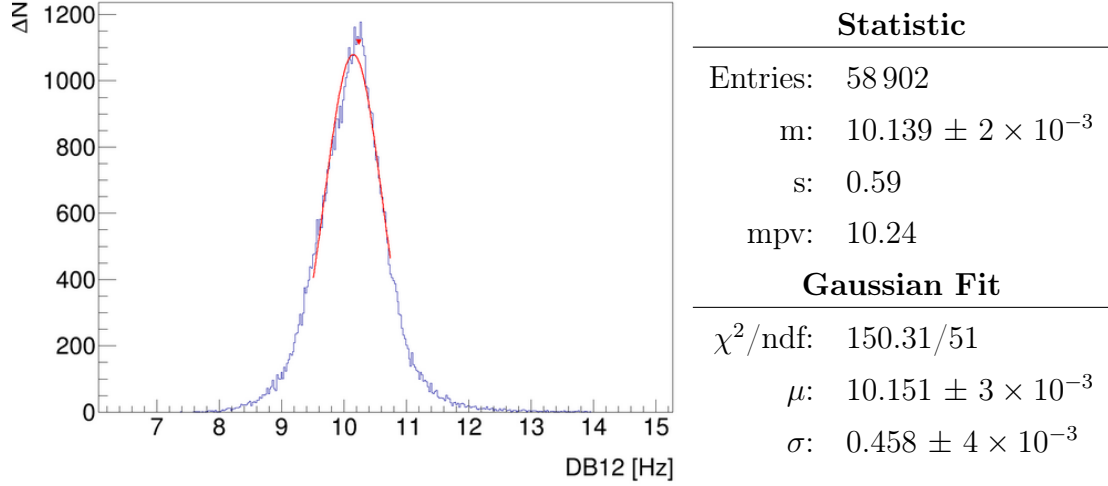


Figure 4.41: Distribution of the DB12 results in the VM-8_3 - couple configuration.

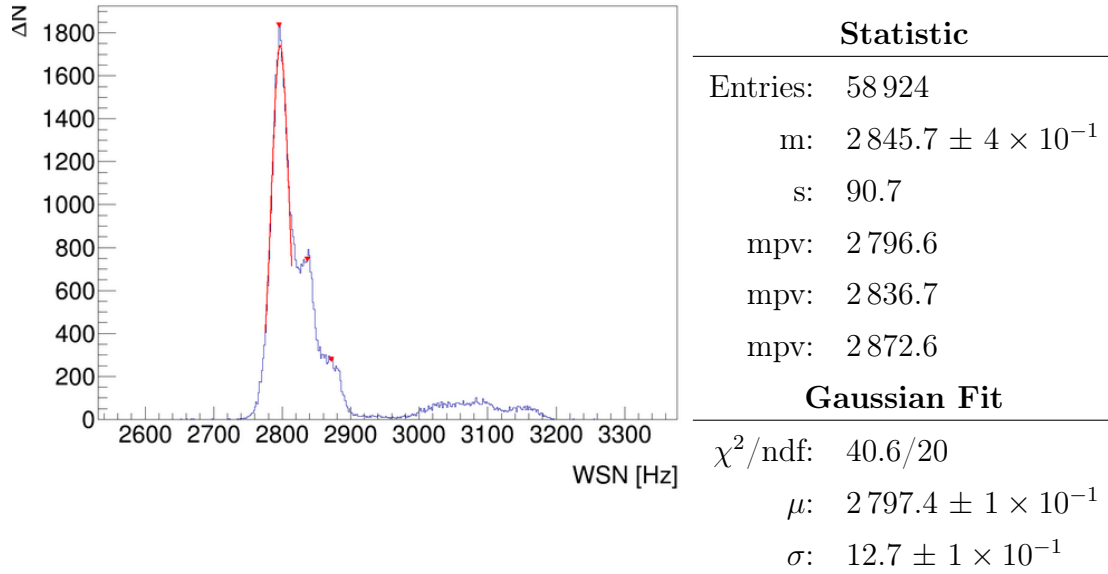


Figure 4.42: Distribution of the WSN results in the VM-8_3 - couple configuration.

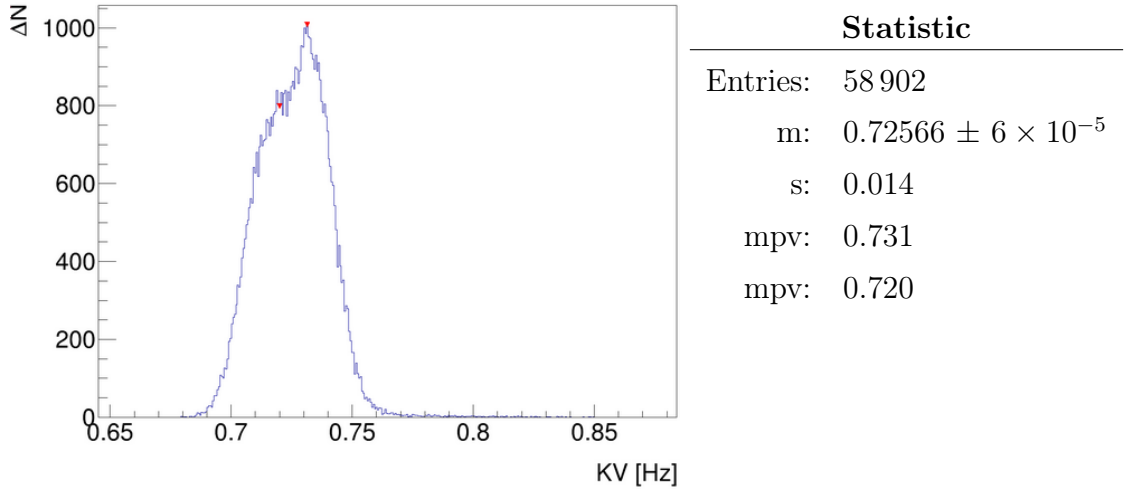


Figure 4.43: Distribution of the KV results in the VM-8₃ - couple configuration.

In order to verify the CPUs performance difference for the KV benchmark, the same approach as in VM-16 and VM-8₄ is used. As before, only the KV workload is reported, the DB12 and the WSN benchmark have been studied as well but they do not reveal any interesting results. Figure 4.44 shows the KV benchmark results for each VM, where a clear distinction between VMs performances is evident. In order to divide the distribution into two sub-structures (Fig. 4.45), the quantiles algorithm has been employed. In particular, Fig. 4.47 identifies the executions of the set of VM_{slow}, whereas Fig. 4.46 refers to the set of VM_{fast}. As for the previous cases, the performance discrepancy between the CPUs is about a factor of 2.

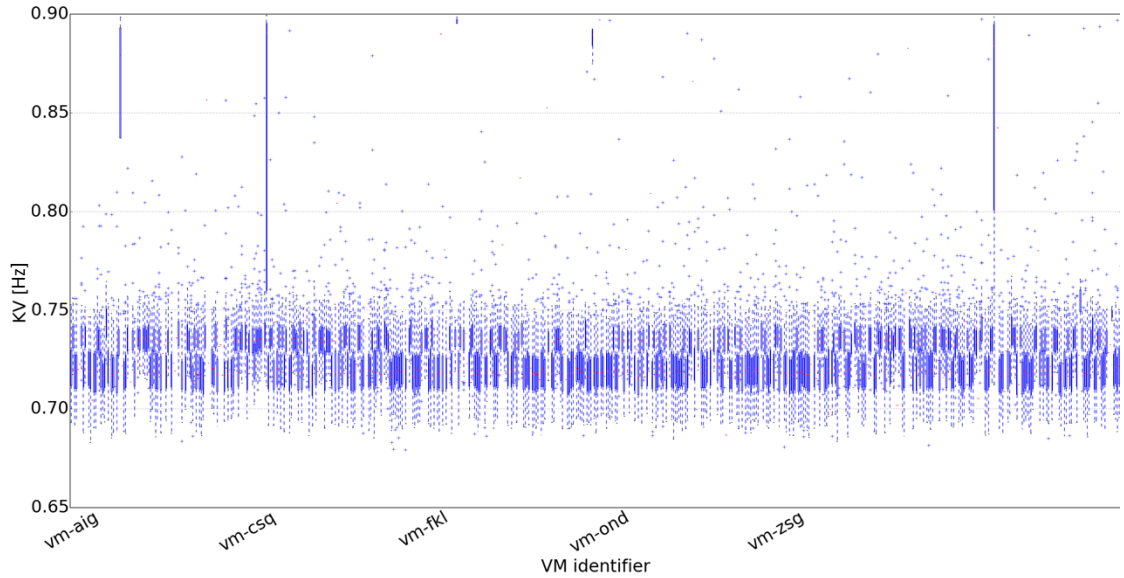


Figure 4.44: The boxplots show the dispersion of the KV results with respect to the VM on which they run. It highlights the difference between the two clusters.

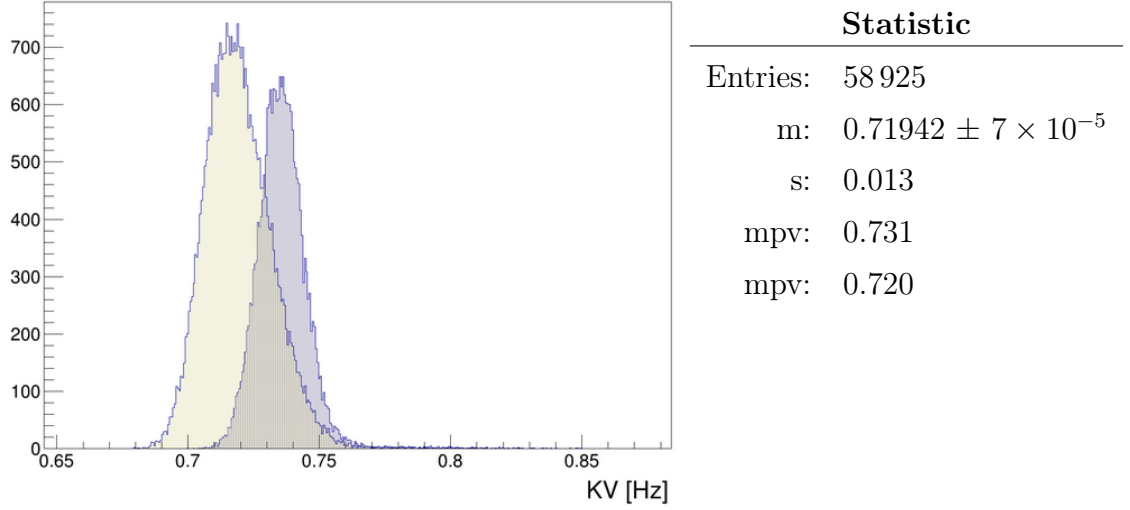


Figure 4.45: Distribution of the KV benchmark results for VM_{fast} (dark shadow) and VM_{slow} (light shadow) classes in $VM-83$ - *single* configuration.

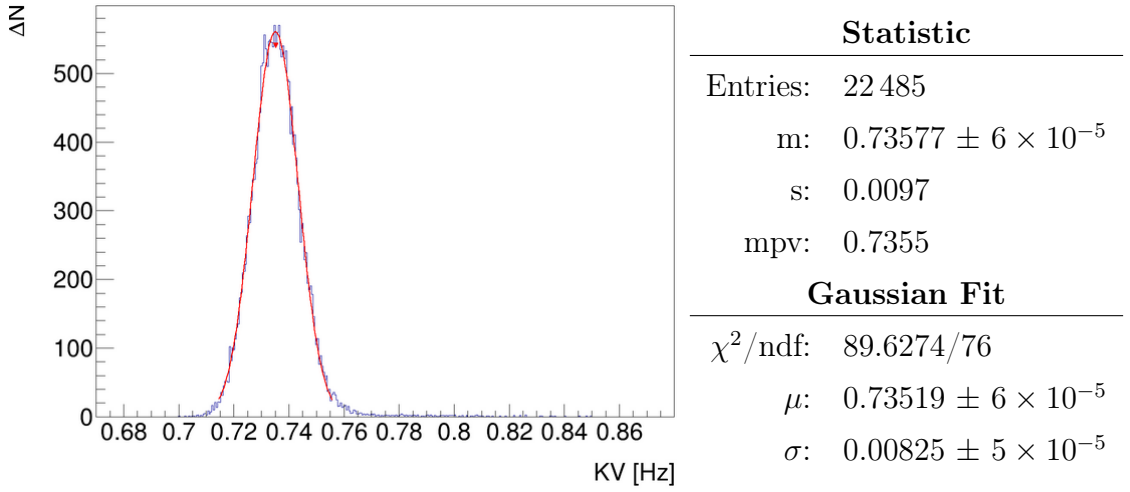


Figure 4.46: Distribution of KV results for fast cluster in $VM-83$ - *couple* configuration.

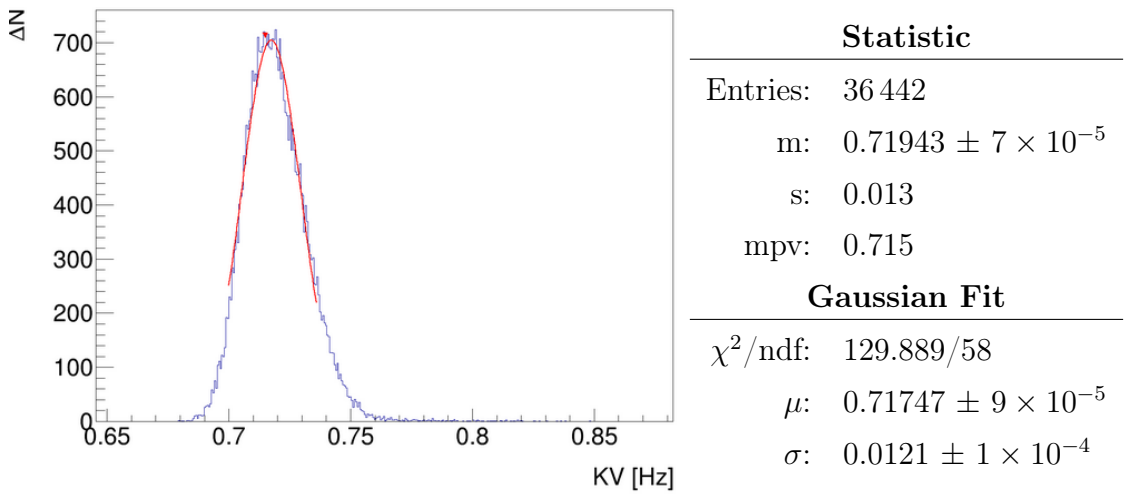


Figure 4.47: Distribution of KV results for slow cluster in $VM-83$ - *couple* configuration.

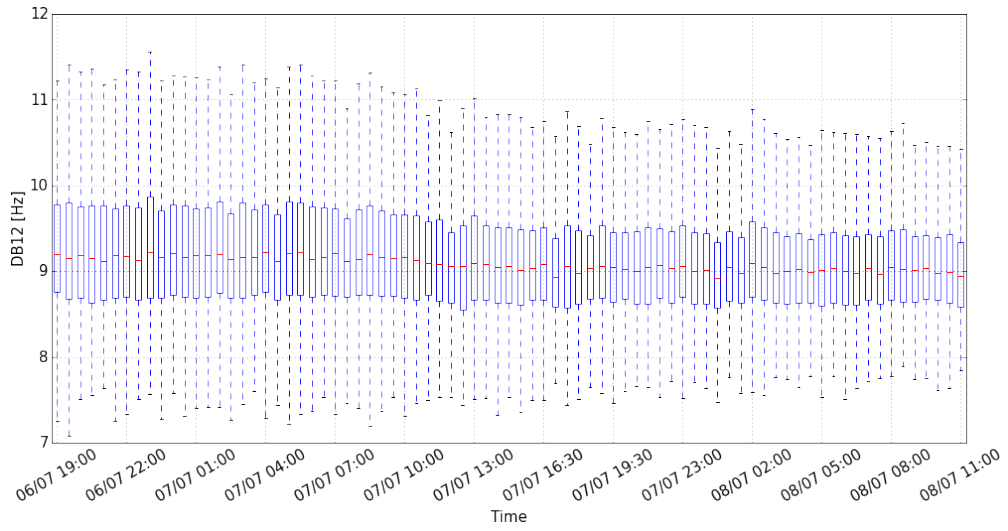
4.2.4 VM-4

VM-4 represents the configuration in which 8 VM with 4 vCPUs each are provisioned in each physical node. Table 4.4 summarizes the total running time and the overall number of tests performed. It should be noted that only 100 servers over the 240 physical nodes have been used to provision the VMs. The restriction is due to the dimension of the pool of public IP addresses available for the VMs associated to those servers, which was equal to 1000. To cope with this restriction a total number of 800 VMs in the physical servers have been provisioned, each of which with its own public IP address.

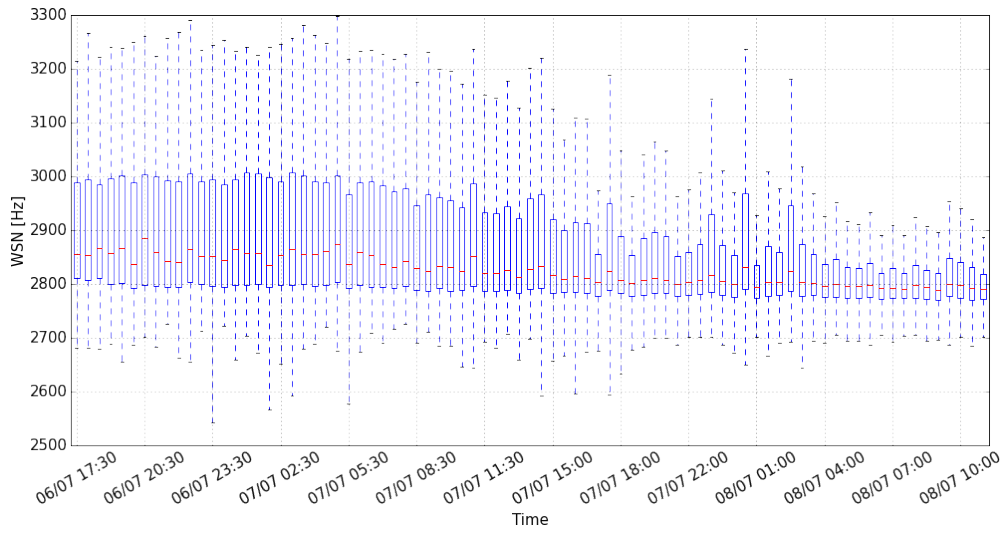
Running period	57 hours
Tests per hour	6
Number of physical nodes	100
Total number of benchmark executions	$\sim 580\,000$

Table 4.4: Test summary for configuration VM-4.

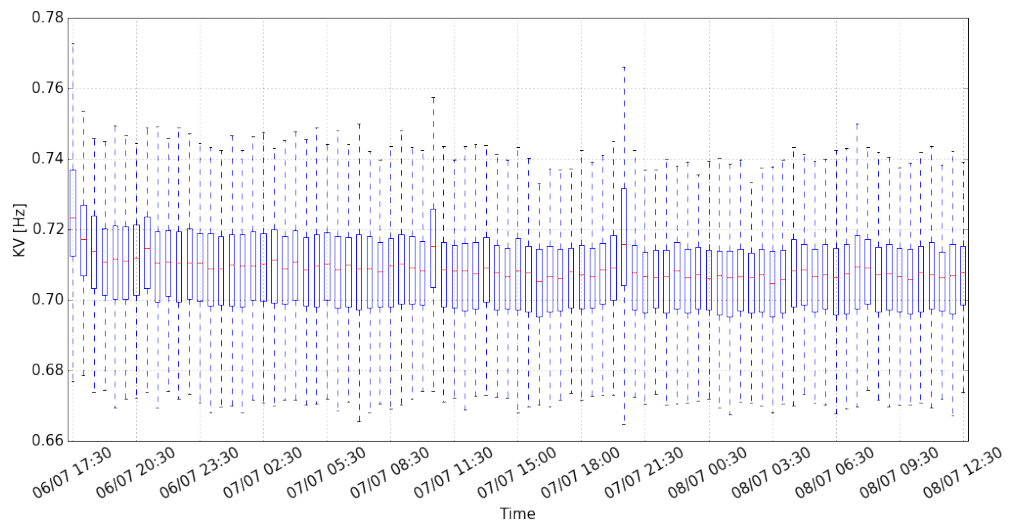
Figure 4.48 shows the time distribution of benchmark results for all the VMs, summarising as boxplots the dispersion of values for each half hour. The measurements were uniform over time, apart from WSN. The uniformity over time of benchmark results motivates the usage of the full set of data for further analysis. Figure 4.49, Fig. 4.50 and Fig. 4.51 visualise, for the DB12, WSN and KV respectively, the distribution of the benchmark results. Moreover, the annexed table reports the mean (m), the root mean square (s), the most probable value (mpv) and the Gaussian fit parameters of the distribution. The DB12 benchmark results in Fig. 4.49 are defined by a Gaussian distribution. The WSN results, in Fig. 4.50, are symmetrically distributed around a central value, a part from outliers that cause a fat tail on the right side. Instead, for what concerns the KV benchmark, following the study of the VM-16 and VM-8 configurations, the VMs have been categorised by a slower or faster performance. In order to perform the clustering algorithm the approach based on quantiles is followed. The procedure is the same as it was for the VM-8 configuration, so it is not described but only the results are shown. Figure 4.51 shows the distribution of KV results for the VM-4_{fast} and VM-4_{slow} overlapped categories. Each category is also plotted in Fig. 4.52 and Fig. 4.53, respectively VM-4_{fast} and VM-4_{slow}, where statistic values and Gaussian fit results are also reported. As for the VM-16 and VM-8 studies, the difference in the CPUs' performance in the KV benchmark is about 2%. In particular, the number of VMs in the VM-4_{fast} class and the VM-4_{slow} class are equivalent at approximatively 400 VMs.



(a)



(b)



(c)

Figure 4.48: Hourly dispersion of the test results summarised as boxplot, for the DB12 (a), for the WSN (b) and for the KV (c) respectively in the VM-4 configuration.

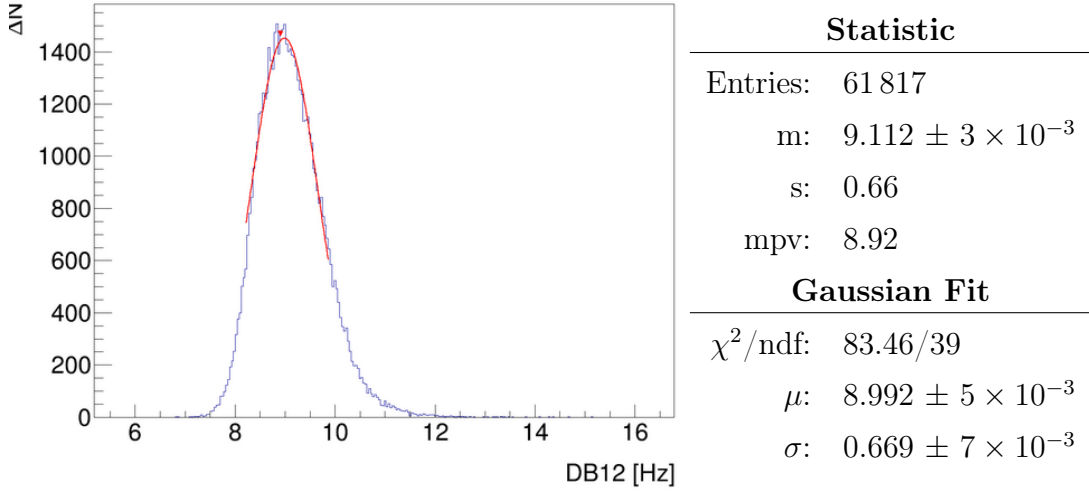


Figure 4.49: Distribution of the DB12 results in the VM-4 configuration.

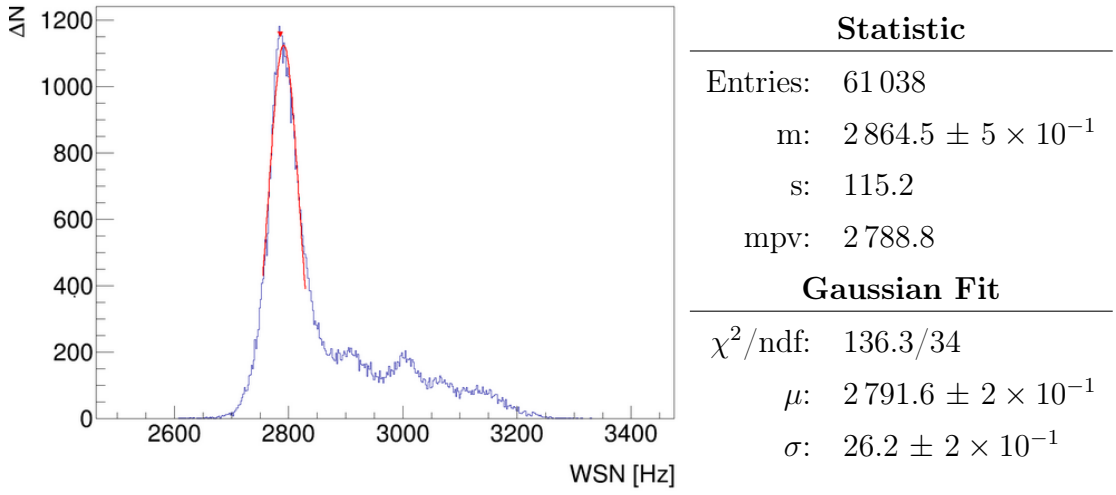


Figure 4.50: Distribution of the WSN results in the VM-4 configuration.

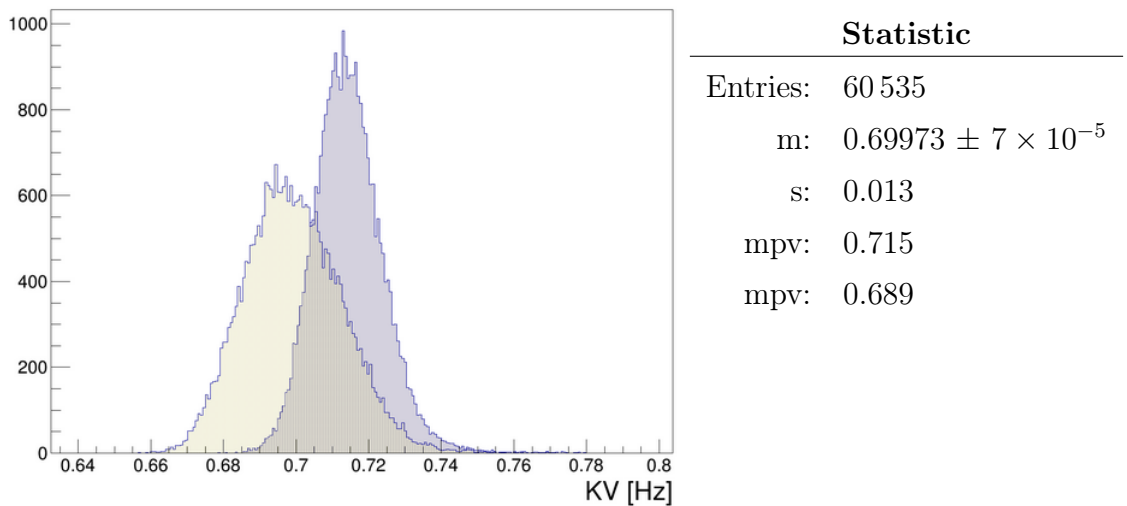


Figure 4.51: Distribution of the KV benchmark's results for VM_{slow} (light shadow) and VM_{fast} (dark shadow) classes in VM-4 configuration.

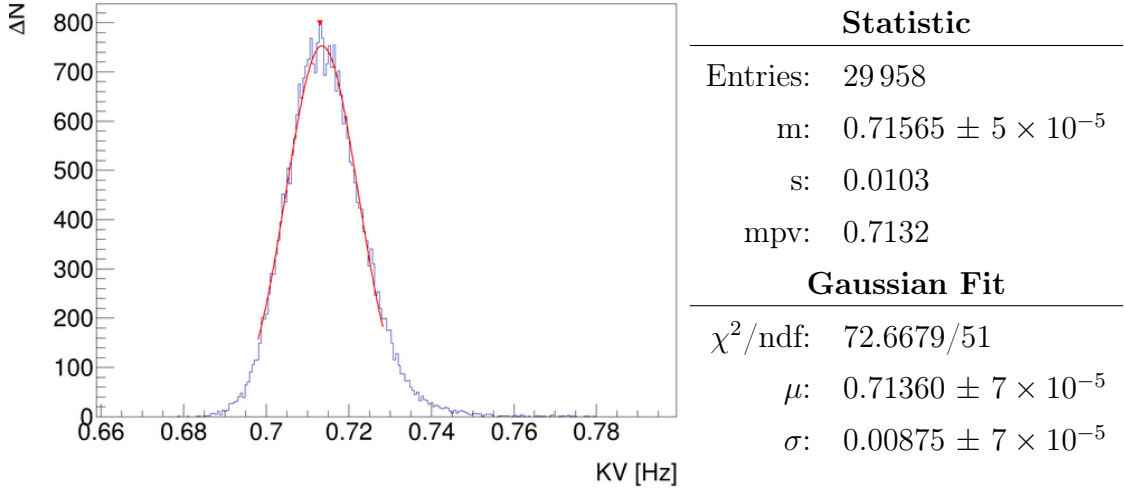


Figure 4.52: Distribution of KV results for VM-4_{fast} class.

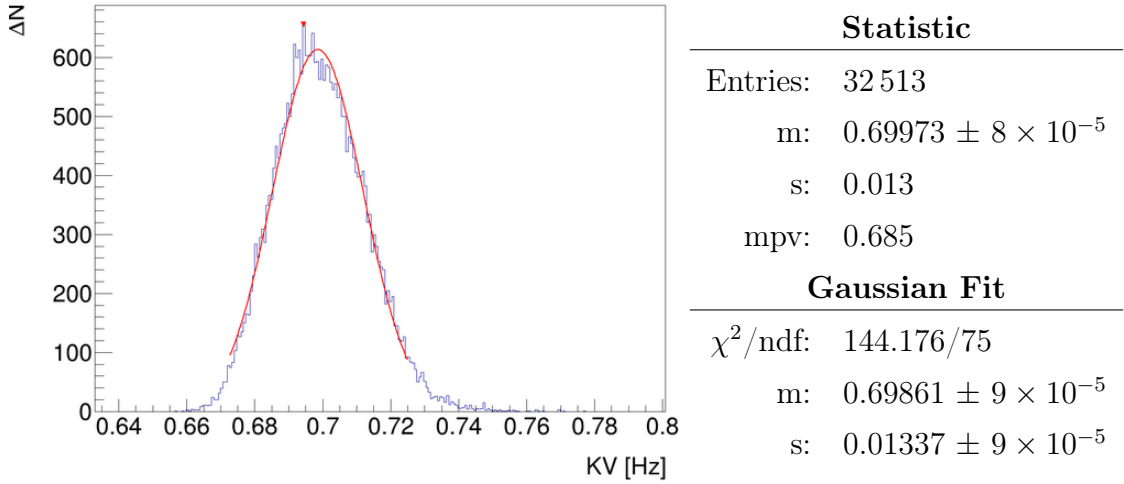


Figure 4.53: Distribution of KV results for the VM-4_{slow} class.

4.2.5 VM-1

VM-1 represents the configuration in which 32 VM with 1 vCPUs each are provisioned in each physical node. Table 4.5 summarizes the total running time and the overall number of tests performed. As for the VM-4 configuration (see Section 4.2.4), also this configuration had the public IP address limitation. For this reason only 30 physical nodes have been tested, hosting a total number of 960 VMs each of which with its own IP. The total number of benchmark executions is very high compared to the other configurations. This being not only because the running period is wider but also because, from configurations rules, each server produces a number of tests equal to the number of vCPUs, while in the other configurations each test result is the average over the vCPUs of the VM.

Running period	245 hours
Tests per hour	6
Number of physical nodes	30
Total number of benchmark executions	$\sim 1\,400\,000$

Table 4.5: Test summary for configuration VM-1.

Figure 4.54 shows the time distribution of benchmark results for all the VMs, summarising as boxplots the dispersion of values for each half hour. The measurements were uniform over time. The presence of symmetric outliers reflects the variability of results caused by the high total number of benchmark executions. The uniformity over time of benchmark results motivates the usage of the full set of data for further analysis. Figure 4.55, Fig. 4.56 and Fig. 4.57 visualise the distribution of the benchmark results, for the DB12, WSN and KV respectively. The effect seen in the previous configurations is not expected to be relevant here since the VMs get dynamically cycled among different CPU logical threads.

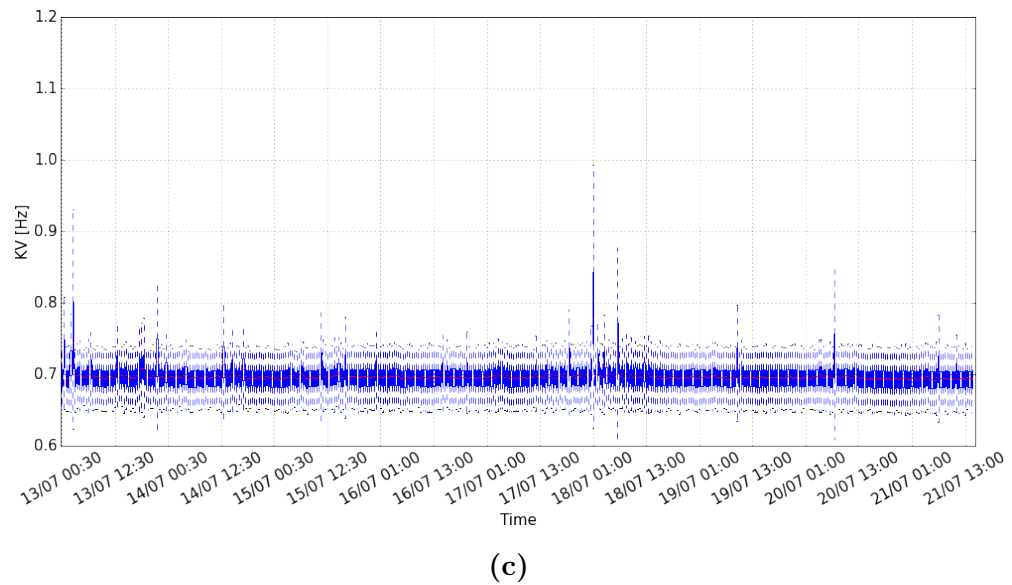
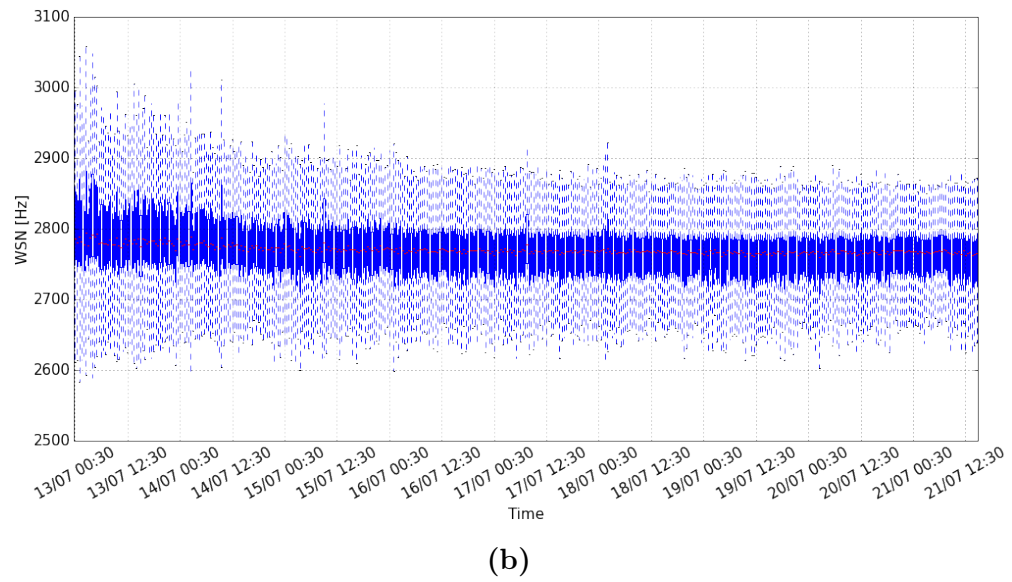
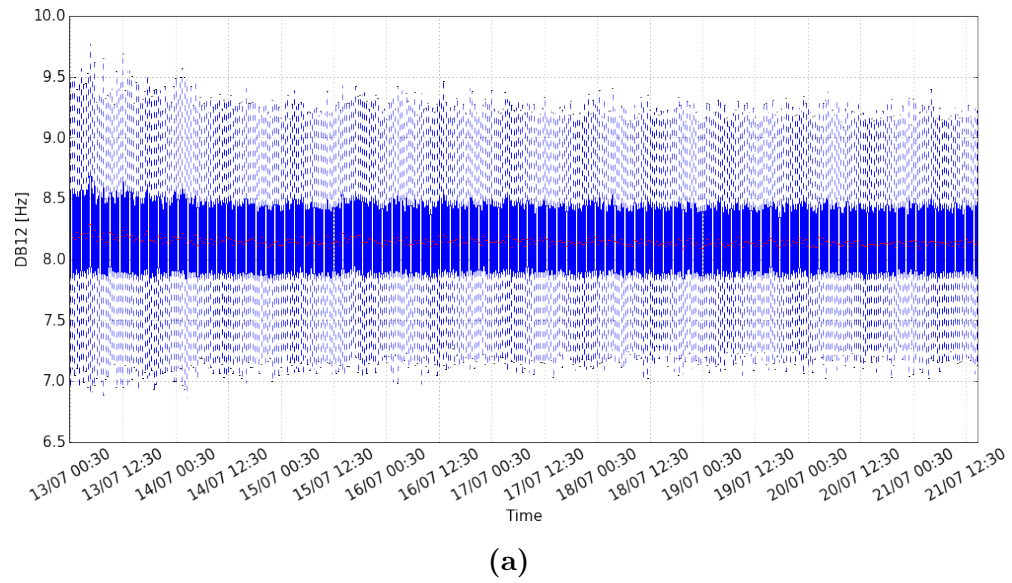


Figure 4.54: Hourly dispersion of the test results summarised as boxplot, for the DB12 (a), for the WSN (b) and for the KV (c) respectively in the VM-1 configuration.

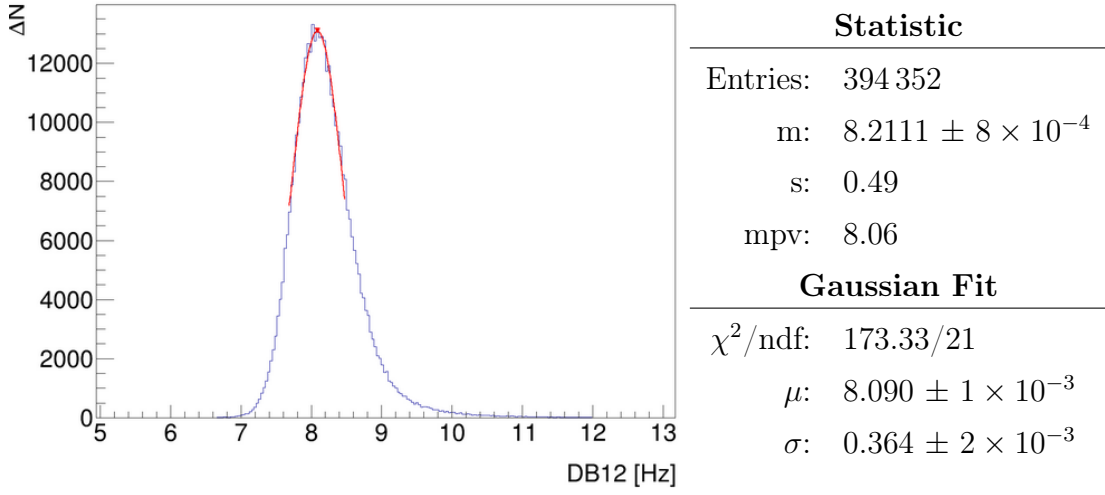


Figure 4.55: Distribution of the DB12 measurements in the VM-1 configuration.

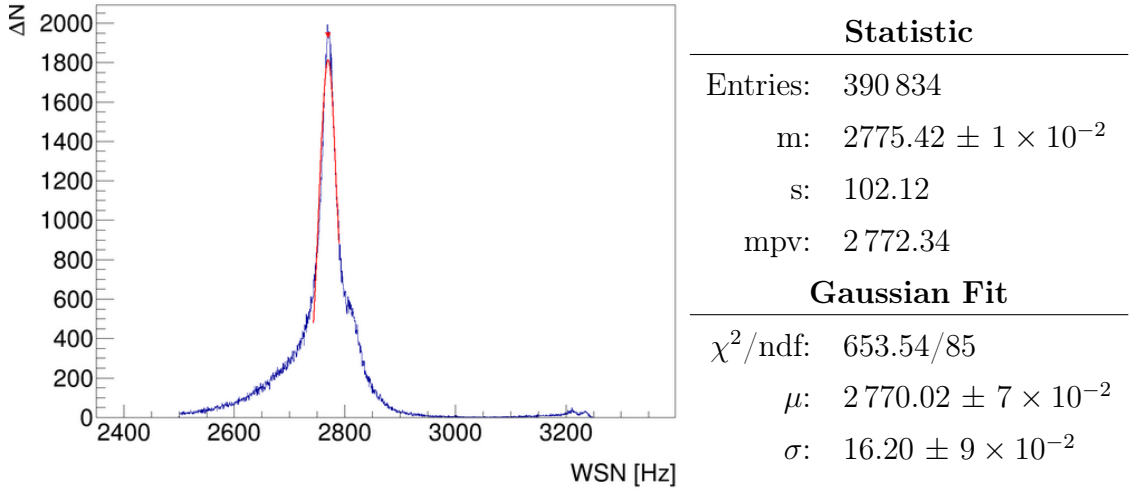


Figure 4.56: Distribution of the WSN measurements in the VM-1 configuration.

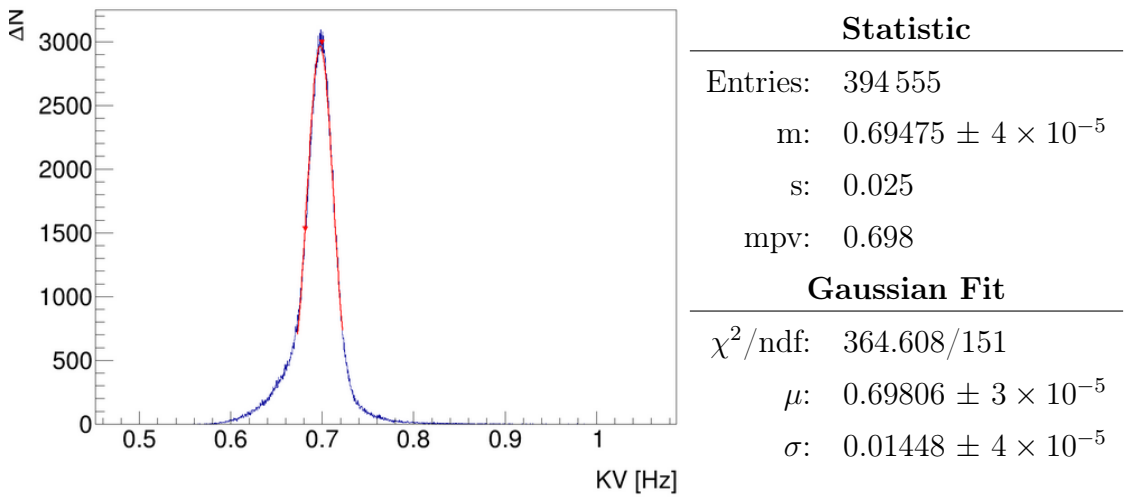


Figure 4.57: Distribution of the KV measurements in the VM-1 configuration.

4.3 HEP-SPEC06 Measurements

The HEP-SPEC2006 benchmark has been studied by the CERN team in addition to the fast benchmarks described in the previous sections. The result of the HEP-SPEC benchmark for each physical node has been extracted from the procurement DB, while the virtual environment has been tested on the VM-8 and VM-16 configuration since they are the two configurations mainly used as batch resources. Even if the HEP-SPEC2006 is not included in the Suite it has been injected with custom cron-based scripts similar to the one used by the DB12, WSN and KV.

Figure 4.58 compares the performance of the VM-8 in green, that of the VM-16 in red and that of the procurement in blue. The scaling factor between the VMs and the bare-metal server values is proportional to the number of VMs inside the physical node as expected, being that the HS06 value is the sum of the benchmark value per thread. This is different compared to the other benchmarks where the average over threads is reported. In particular, the benchmark values for VM-16 is about $\frac{1}{2}$ with respect to the benchmarking of the full bare-metal server; while the HS06 results for VM-8 is approximately $\frac{1}{4}$ with respect to the HS06 of the full bare-metal server (table in Fig. 4.58).

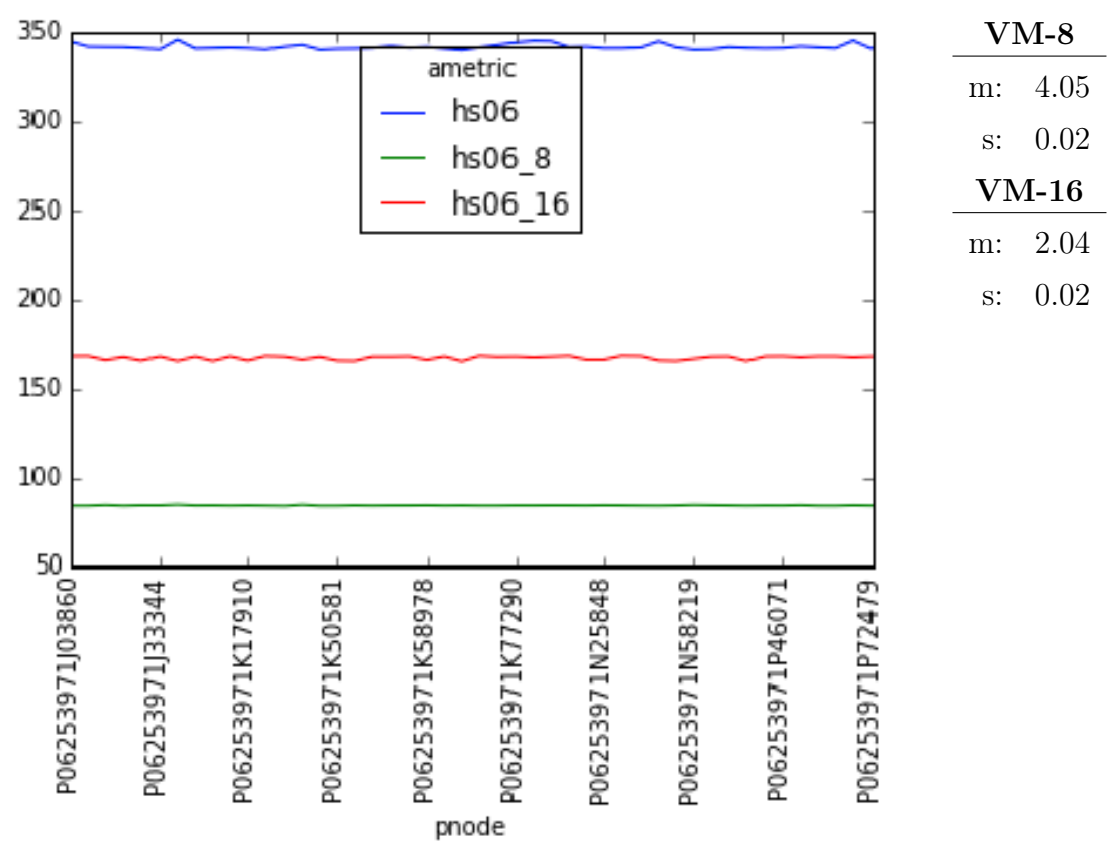


Figure 4.58: Performance comparison for the HEP-SPEC06 between the VM-8 configuration (green line), the VM-16 configuration (red line), and bare-metal (blue line). The table reports mean and standard deviation evaluated over the tested servers.

4.4 Scaling Factors across Configurations

This section summarises the results obtained from the measurements of the four benchmarks in each configuration. The mean value of the statistical distribution in the configuration has been used. In order to compare the different scenarios with different scales, the values have been normalised taking as a reference the performance of the VM-8₃ - alone as follows. The factor of 2 takes into account the double number of threads running with SMT enabled.

$$Ratio_{(bmk_A, VM_X)} = 2 * \frac{\mu(bmk_A, VM_X)}{\mu(bmk_A, VM_{ref})}$$

The configurations summarised are the VM-32, VM-16, VM-8, VM-8₃ - alone (referred to as VM-8_{3A}), VM-8₃ - couple VM-8_{3T} and VM-1. It should be noted that the VM-8 scenario has been divided into its sub-configurations, since the performance of other scenarios do not have significant substructures. Figure 4.59 shows the performance across configurations.

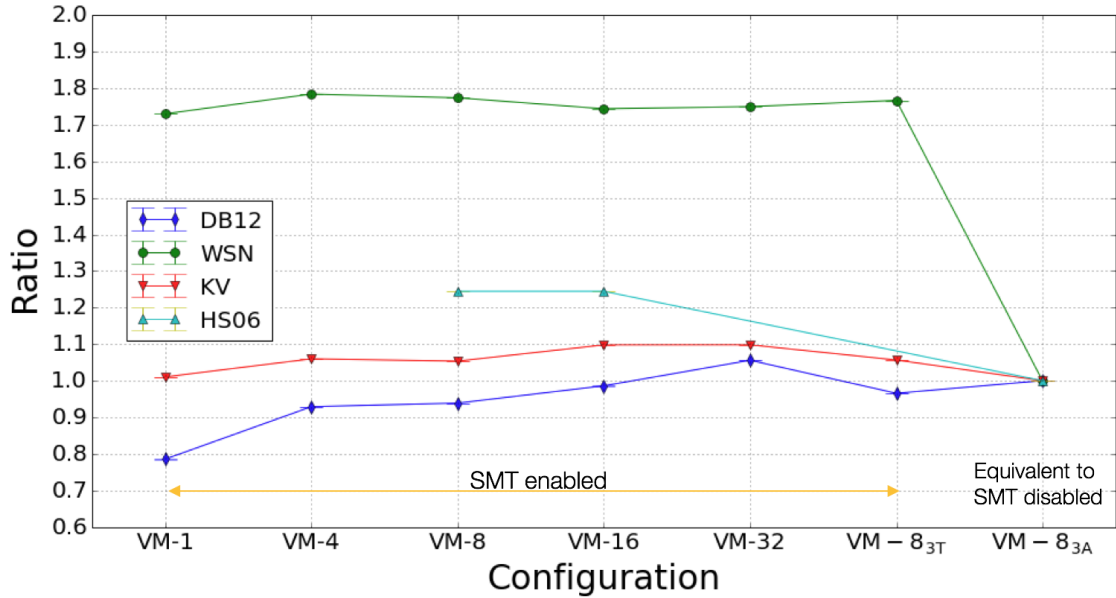


Figure 4.59: Relative performance between different configurations: DB12 (blue), WSN (green), KV (red) and HS06 (light-blue).

The relative performance (Fig. 4.59) highlights that when simultaneous multi-threading is enabled the HS06 performance has a 20% increase in the throughput, justifying the adoption of SMT enabled for this kind of workloads. WSN, in its simplicity, gains even more in throughput, reaching the 75% gain with respect to VM running on servers with SMT disabled. KV shows a limited gain of less than 10%, but still in the limit to affirm that the global throughput increases using SMT enabled. DB12 shows no benefit and even some penalty in several configurations. Further investigation of the reasons for that behaviour are reported in [69]. Table 4.6 summarises the means for each benchmark for each configuration used in the plot in Fig. 4.59.

	DB12 [Hz]	WSN [Hz]	KV [Hz]	HS06 [Hz]
VM-1	8.09	2770	0.698	
VM-4	8.99	2791	0.669	
VM-8	9.68	2798	0.724	4.05
VM-16	10.29	2808	0.755	2.04
VM-32	11.09	2804	0.755	
VM-8 _{3T}	10.15	2797	0.726	
VM-8 _{3A}	21.07	3221	1.385	

Table 4.6: Summary of the benchmarks mean for each studied configuration.

Chapter 5

Building a Model

Several techniques can be adopted to analyse and forecast the computing system's performance. The choice between them is driven by the specific purpose of the evaluation, by the knowledge of the system being modelled and by the level of desired accuracy. One approach consists of measuring the performance of a system under real conditions of workloads and configurations, as described in Chapter 4. This procedure provides reliable results at the expense of flexibility, since real system modification requires additional costs. In this regard, the simulation of computing resources can support the measurement-based approach since it investigates the system behaviour in a more flexible way than direct experimentation. The simulation approach uses techniques to realise models as an abstraction of the real system selecting the aspects relevant to the study. The greater flexibility can, in some case, increase the prediction power of a study. In order to provide a good level of accuracy, the performance measures of the real system are compared to the model outputs. Indeed, the measurements from the experimental approach are used to define realistic working points, to constrain parameters and to validate the simulation. Once the model is considered sufficiently reliable, it can be parametrised in order to enable evaluation under several hypothetical alternatives, such as different configurations or new features that cannot be easily included in the real system. The mentioned approach is an iterative process because of dependencies that exist among the definition of the model, among the techniques used to evaluate the prototype and among the procedures used to parametrize it.

This Chapter describes the foundation of a simulation based on models inspired by the results discussed in the previous chapters. The aim is to show, as an exemplification, how the measurements collected by the analysis of Chapter 4 are used to parametrise the model and how the model is validated.

5.1 Queuing Network Model

One of the most common techniques for the simulation approach is the queueing network technique [88], due to its balance between accuracy and flexibility. This technique is widely used not only in the computing field, but also in other domains. For example, the model can simulate queues at supermarket cashiers and its goal is to provide a balance between customer happiness and cashiers utilization costs. The

same worth for computing systems, in which applications compete for resources that are required to work repeatedly at full capacity. Queuing networks are stochastic models of resources sharing systems. The models consist of a network of interactive service centres, representing the real system resources, that provide service to collections of jobs representing the workloads of the real system. Therefore, a queueing network is defined by the service centres, the customers and network topology. Service center characteristics include the service time, the buffer space with its queueing scheduling and the number of servers. In particular, the queue scheduling is the algorithm used to decide which customer to serve next; for example, in the First Come First Served (FCFS) scheduling customers are served in the same sequence in which they arrive at the station. Customers are described by their number for closed models and by the arrival process to each service center for open models, the service demand of each service center and the types of customer. The various types of customer in the queueing network model defines different behaviours of the workload. The network topology models how the service centres are interconnected and how the customers move between them. When a job arrives at a service centre, it asks for the resource service and wait to be served; at completion time, the job exits the queue and moves to another service station. The interaction between the queues and the jobs simulates the performance of the system, which can be evaluated through a set of performance measures, such as resource utilization, throughput and customer response time.

5.2 Java Modelling Tool

The tool that has been used to implement the model is Java Modelling Tools (JMT) [89]. It is a project which started in 2002 in Politecnico di Milano to extend an existing set of tools for performance evaluation called Windows Modelling Tools. JMT is a free open source suite consisting of six tools for the performance evaluation, capacity planning and workload characterisation; moreover, the suite implements several algorithms for the exact, asymptotic and simulative analysis of queueing network models. The suite is composed of several components and for the purpose of this Thesis the component called *JSIM graph* has been used. It is a GUI that allows for an easy description of the network structure providing advanced features such as fork and join of customers, blocking mechanisms and regions with capacity constraints on population. Moreover, the simulation engine performs the statistical analysis of measured performance indices on-line, plots the collected values, discards the initial transient periods and computes the confidence intervals.

5.3 Baseline of the Model

In the following sections the models simulating the real system are described for the VM-16 and VM-8 configuration during the execution of the DB12, WSN and KV benchmarks. These scenarios are the most interesting to be modelled, since they are the most frequently used in the CERN batch system.

Considering that the real servers do not interfere one with the other in the

performance evaluation, the queuing networks simulate a single physical server in each of the aforementioned configurations. The workload characterisation relies on the distributions of the benchmarks' results collected in the measurement phase. All distributions of the real system are simplified to the Gaussian shape. Even if this is a simplification, it is supported by the fact that all experimental distributions were mainly gaussian, with outliers due to effects that have been ignored for the sake of model simplicity. Therefore the service time distribution of the queues are also modelled as Gaussian distributions. It should be noted that, whereas the measurements of Chapter 4 represent the throughput of the system, the simulation requires in input a time distribution and, therefore, the inverse is used. The implemented models are 'effective' models, where the CPU is represented as a 'black box' characterised by a single parameter, the service time distribution. Other components of the real server, such as caches, memory, disk and network are not directly included in the model, because they are not relevant for the proper compute performance (disk and network for instance) or because their interplay with the CPU is summarised by the measured time distribution (for instance L1 and L2 caches, and memory).

5.3.1 VM-16 and VM-8₄

Simulation of DB12 and WSN benchmarks

The goal of the first model is to simulate the real system executing the DB12 and WSN benchmarks in the VM-16 and VM-8₄ configurations. The schema of the model is represented in Fig. 5.1.

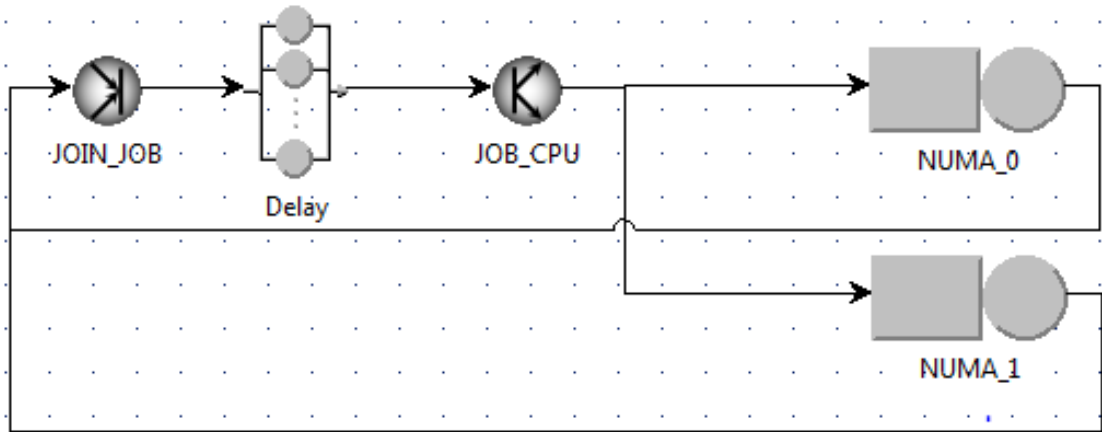


Figure 5.1: The model describes the VM-16 and VM-8₄ physical node in the DB12 and WSN simulation.

The model simulates a physical node having two CPUs, identified as NUMA_0 queue and NUMA_1. For both configurations, each service centre is characterised by a number of servers equal to 16 and by a FCFS queuing strategy, since in both cases all the threads are fully loaded. It should be noted that for the DB12 and the WSN, in both scenarios, no performance difference between threads has been highlighted. Therefore, the benchmark instances, for both DB12 and WSN, are simulated by jobs

of the same type. In particular, for VM-16 one job per queue is created, whereas for VM-8₄ two jobs per queue are created. In order to simulate each benchmark at thread level, jobs are split, through the JOB_CPU fork, into a number of tasks equal to the number of vCPUs of the equivalent VM. In each queue, for the VM-16 the job is split into 16 tasks, whereas for the VM-8₄ the two jobs are split into 8 tasks each. The complementary of a fork component is the join station. The latter is only used to recombine the tasks of a job split by the previous fork. When a task arrives at a join station, it waits until the arrival of all its sibling tasks. At this time, the original job is recomposed and routed to the next station. Therefore, the system response time of the model is evaluated after the JOIN_JOB component.

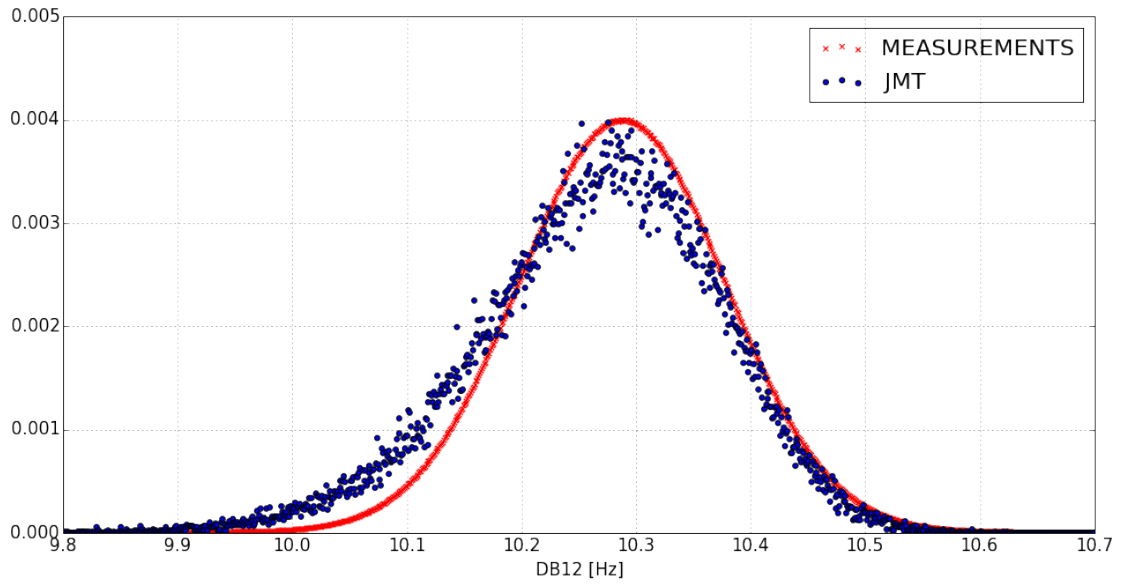
In order to validate the queuing network, the system response time of the model, μ_{out} , is imposed to be equivalent to the response time of the real system. This latter is computed as the inverse of the mean of the Gaussian fit in Fig. 4.10 (Fig. 4.11) for DB12 (WSN) in VM-16 and in Fig. 4.25 (Fig. 4.26) in VM-8₄. Remembering that a join station waits the arrival of all the sibling tasks, μ_{out} of the model is calculated as the maximum over the tasks' service time (Equation 5.1). The tasks service time, μ_{in} , is computed with an algorithm that takes as input the $\mathcal{N}(\mu_{out}, \sigma_{out})$ and the number of tasks (n). Table 5.1 summarises the expected response time, μ_{out} , and the service time, μ_{in} .

$$\mu_{out} = E[\max_n \mathcal{N}(\mu_{in}, \sigma_{in})] \quad (5.1)$$

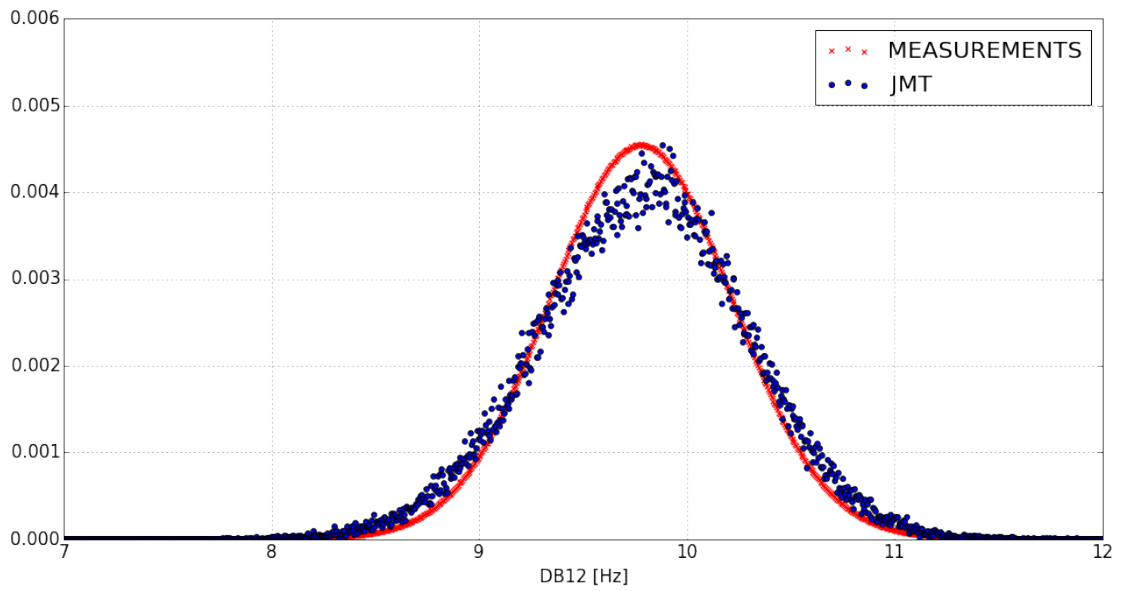
VM-16			VM-8 ₄	
	DB12 [s]	WSN [s]	DB12 [s]	WSN [s]
μ_{out}	0.0972	$3,56 \times 10^{-2}$	0.1033	$3,57 \times 10^{-2}$
μ_{in}	0.0968	$3,55 \times 10^{-2}$	0.0912	$3,48 \times 10^{-2}$

Table 5.1: Summary of the expected response times, μ_{out} , and service time, μ_{in} for DB12 and WSN in the VM-16 and VM-8₄ configurations.

To verify the accuracy of the simulation, the model results are compared to the measurements of the real system. In order to do so, the model results are converted into throughput. The Gaussian fit of the measurement-based approach are plotted on the same frame of the distributions of the model-based approach. In particular, the dot markers identify the JMT model throughput, whereas the cross markers identifies the measurement-based distribution. For VM-16 and VM-8₄ respectively, Fig. 5.2 defines the DB12 distributions and Fig. 5.3 defines the WSN distributions. For DB12 benchmark, in both configurations, the model approximates the real system profile with a satisfactory level of accuracy. The same considerations apply for the WSN benchmark. Therefore, the model in Fig. 5.1 has been validated for the VM-16 and VM-8₄ scenarios executing the DB12 and WSN benchmarks.



(a)



(b)

Figure 5.2: DB12 results distribution, for the VM-16 scenario (a) and VM-8₄ scenario (b). The red curve identifies the Gaussian fit of analysis' data, whereas the blue curve defines the distribution of results of the model simulation.

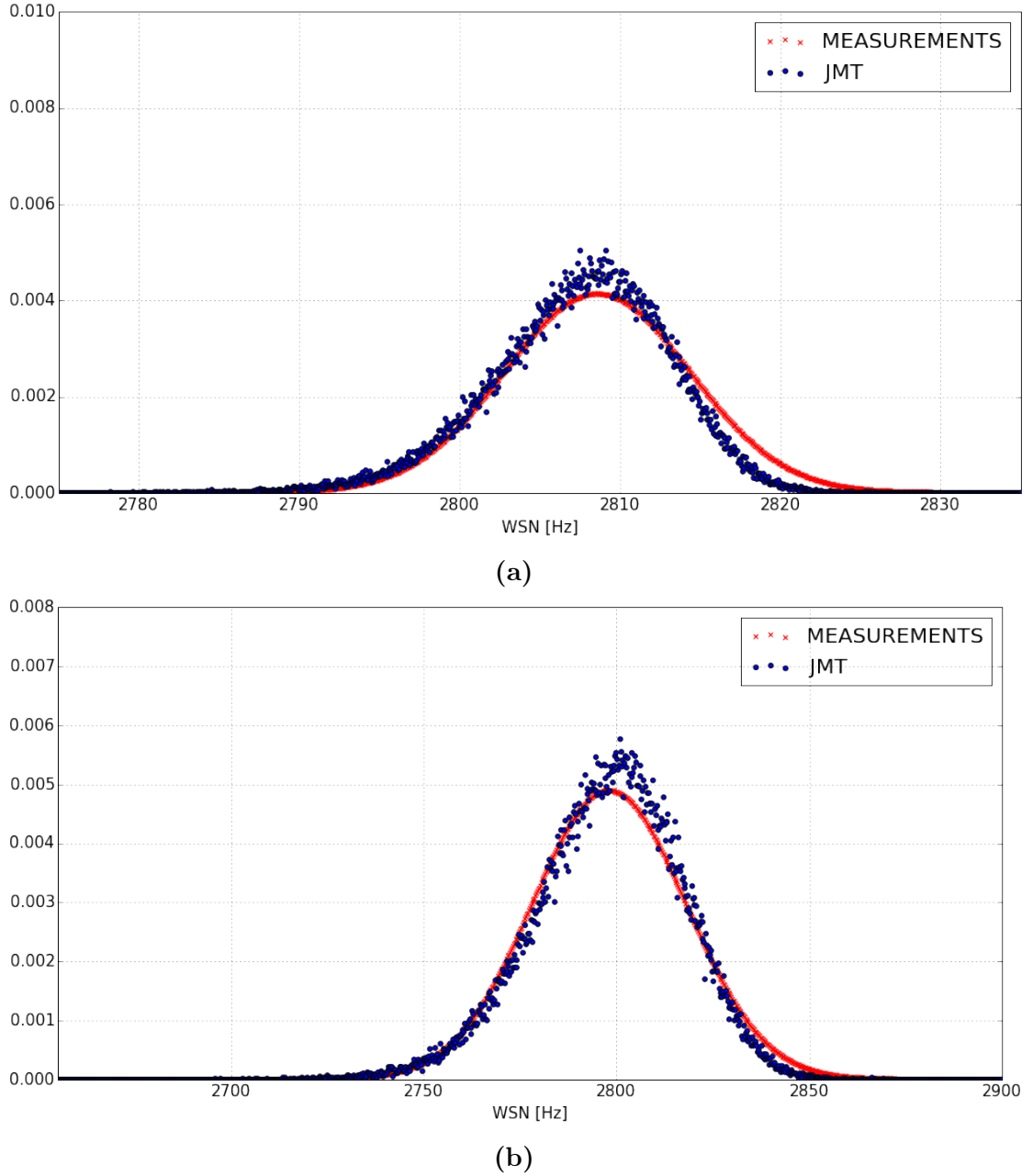


Figure 5.3: WSN results distribution, for the VM-16 scenario (a) and VM-8₄ scenario (b). The red curve identifies the Gaussian fit of analysis' data, whereas the blue curve defines the distribution of results of the model simulation.

Simulation of KV benchmark

The KV benchmark simulation requires a different model structure able to reproduce the performance difference between CPU's threads highlighted in Section 4.2.2. Figure 5.4 shows the queue network used to model the system. This last is a thread level generalisation of the model for VM-16 and VM-8₄ (Fig. 5.1). There are several commonalities shared between the two queuing networks. Firstly, the two CPUs of the physical node are modelled with the NUMA_0 and NUMA_1 queues. Moreover, for both VM-16 and VM-8₄ scenarios, each service centre is characterised by a

number of servers equals to 16 and by a FCFS queuing strategy, since all threads are fully loaded. Finally, through the `ASSIGN_VM`, one job per queue is created for VM-16 scenario, whereas two jobs per queue are created for VM-8₄ scenario.

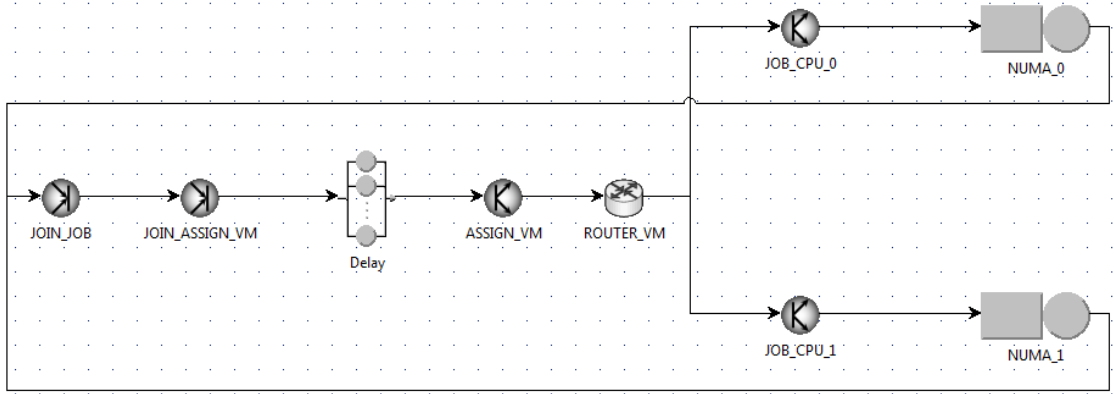


Figure 5.4: The model describes VM-16 and VM-8₄ physical node in the KV simulation.

As for DB12 and WSN, the model simulates the benchmark at thread level and, therefore, jobs are split into a number of tasks equal to the number of vCPUs of the equivalent VM. However, for the KV benchmark, due to the performance difference between vCPUs, the tasks are not of the same type and, therefore, two classes are defined. The `fast` class identifies the threads performing faster, whereas the `slow` class identifies the threads performing slower. It should be remembered that the slow threads are the two vCPUs belonging to the first physical core of the second CPU. For the VM-16 scenario, the job running on the `NUMA_0` queue is split into 16 fast tasks by the `JOB_CPU_0` fork, whereas the job running on the `NUMA_1` queue is split into 14 fast and 2 slow tasks by the `JOB_CPU_1` fork. For the VM-8₄ scenario, the two jobs running on the `NUMA_0` queue are split into 8 fast tasks each by the `JOB_CPU_0` fork, whereas the two jobs running on the `NUMA_1` queue are split into 7 fast and 1 slow tasks each by the `JOB_CPU_1` fork. This division is a simplification of the real system behaviour, since the scheduler does not pin the threads in a fixed way; therefore, the two slow tasks might belong to the same VM. As for the DB12 and WSN benchmarks simulations, the expected response time, μ_{out} , is the inverse of the mean of the Gaussian fit. For the VM-16, μ_{out} is extracted from Fig. 4.15 (Fig. 4.16) for the fast (slow) task and for the VM-8₄, from Fig. 4.31 (Fig. 4.32). Instead, the service time, μ_{in} , is computed through the Equation 5.1. Table 5.2 summarises the service time, μ_{in} , and the expected response time, μ_{out} .

VM-16			VM-8 ₄	
	fast [s]	slow [s]	fast [s]	slow [s]
μ_{out}	1.3091	1.3380	1.3659	1.3966
μ_{in}	1.2740	1.3050	1.3321	1.3556

Table 5.2: Summary of the expected response times, μ_{out} , and service time, μ_{in} for KV in the VM-16 and VM-8₄ configurations.

To verify the accuracy of the simulation, the model results are compared to the measurements of the real system. In order to do so, the model results are converted into throughput. The Gaussian fit of the measurement-based approach are plotted on the same frame of the distributions of the model-based approach. In particular, the dot markers identify the JMT model throughput, whereas the cross markers identifies the measurement-based distribution. Figure 5.5 show the KV distributions for VM-16 and VM-8₄ configurations respectively. In both configurations the model approximates the real system profile with a satisfactory level of accuracy. The two peaks are reproduced. In particular the left peak of the model's curve, identifying the VM performing of the second CPU, accurately follows the respective Gaussian fit whereas the right peak, identifying the VM performing of the first CPU, follows with a discrepancy of the 0.01% the relative Gaussian fit. The model in Fig. 5.4 has been validated for the VM-16 and VM-8₄ scenarios executing the KV benchmark.

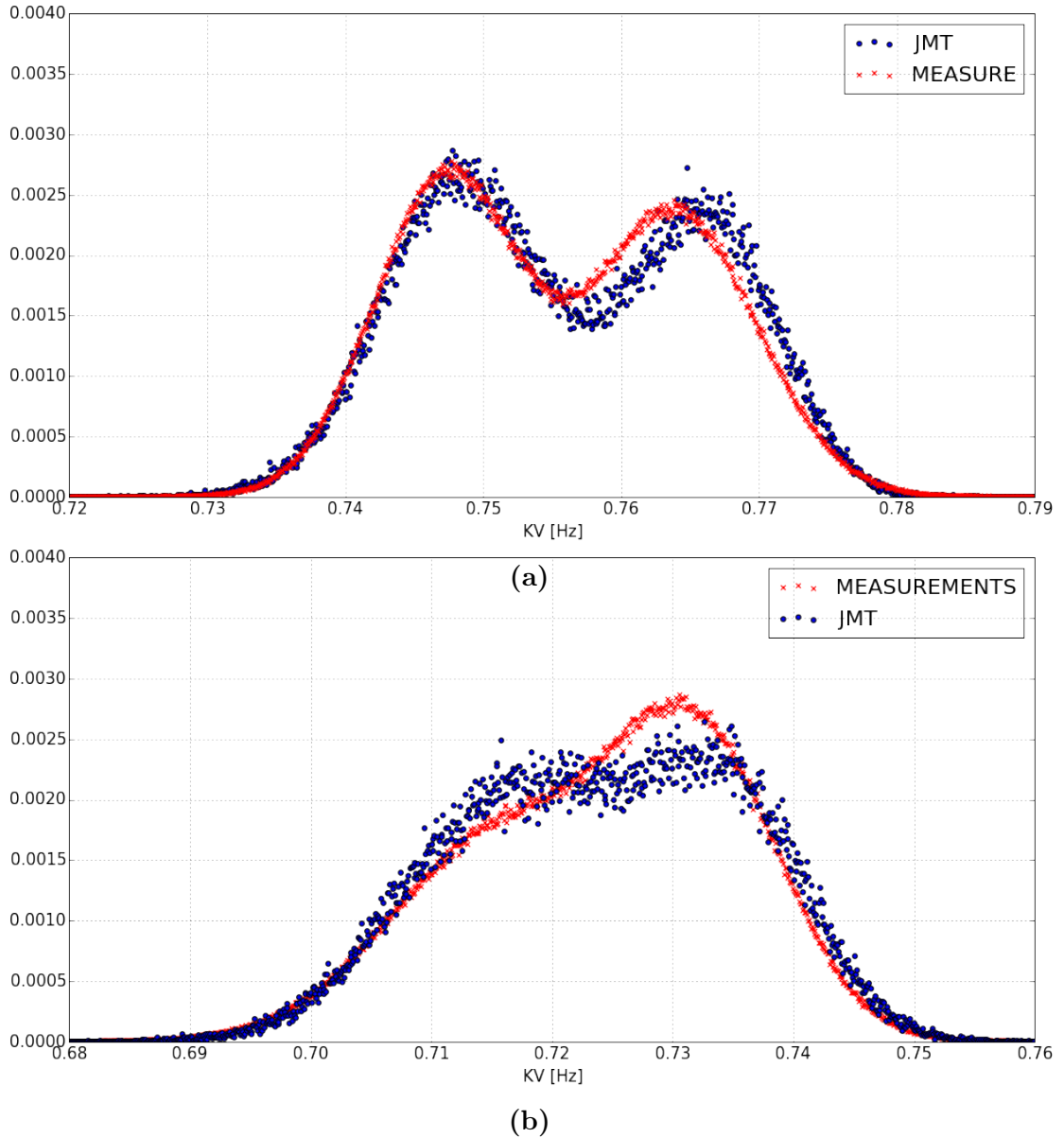


Figure 5.5: KV results distribution, for the VM-16 scenario (a) and VM-8₄ scenario (b). The red curve identifies the Gaussian fit of analysis' data, whereas the blue curve defines the distribution of results of the model simulation.

5.3.2 Simulation of the VM-8₃ scenario

The configuration with 3 VMs, each consisting of 8 vCPUs each, needs to introduce a higher level of detail with respect to VM-16 and VM-8₄, since the number of VMs is not symmetrically distributed among NUMA nodes. Indeed, in the VM-8₃ configuration there are two scenarios: one VM on the first CPU and two VMs on the second CPU, or two VMs on the first CPU and one VM on the second CPU. About 50% of the time the VM runs alone on the first CPU, whereas the remaining percentage of times the VM execute alone on the second CPU. The VM-8₃ model is shown in Fig. 5.6 and represents a particular case of the model in Fig. 5.4.

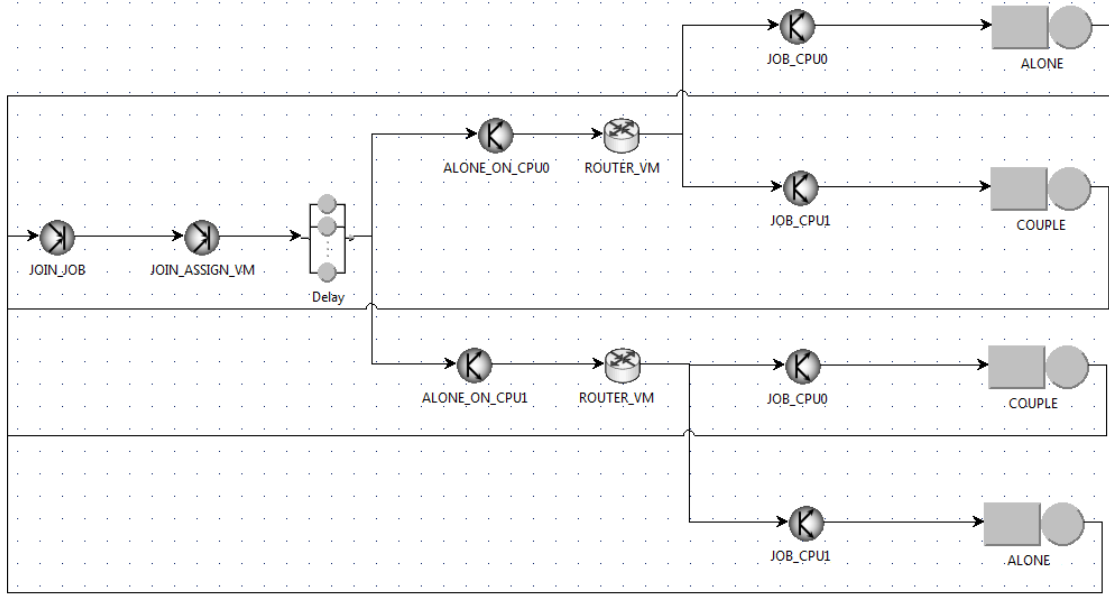


Figure 5.6: The model describes the VM-8₃ physical node in the simulation of DB12, WSN and KV benchmarks.

The model visualises a symmetric structure, in which the above branch, after `ALONE_ON_CPU0` fork, designs the server in which one VM runs on the first CPU and two VMs run on the second CPU (`NUMA_ALONE0` branch); whereas, the lower branch, after `ALONE_ON_CPU1` fork, designs the server in which two VMs run on the first CPU and one VM runs on the second CPU (`NUMA_ALONE1` branch). This replica is due to constraints of the JMT tool, that does not allow, for this simulation, a sharper scheduling of jobs in the system. Due to the symmetry of the model, following describes the `NUMA_ALONE0` branch only, but the same considerations can also be made for the `NUMA_ALONE1` branch. The role of the `ALONE_ON_CPU0` fork is to create jobs that simulate the response time of a benchmark run. The jobs are distinguished in two classes, the `cpu0` class defines the benchmark instance to be dispatched on the above branch, whereas the `cpu1` class defines the benchmark instance to be dispatched on the below branch. Therefore, in the `NUMA_ALONE0` branch, one `cpu0` enters the `JOB_CPU0` fork, two `cpu1` enters the `JOB_CPU1` fork. As for the previous simulations, in order to model the benchmarks at thread level, jobs are split, with `JOB_CPU0` and `JOB_CPU1` forks, into a number of tasks

equal to the number of vCPUs of the equivalent VM. Depending on the benchmark being simulated, the type of task can vary. For DB12 and WSN, since there is no difference between vCPUs, only the `fast` class is defined. Therefore each job is split into 8 `fast` tasks each. In the `NUMA_ALONE0` branch, 8 tasks are executed in `ALONE` queue, 16 on `COUPLE` queue. For the KV benchmark, due to the difference between vCPUs, the `fast` class is identified, as the threads performing faster, and the `slow` class, as the threads performing slower. Therefore, on the higher branch each job is split into 8 `fast` tasks, whereas on the lower branch each job is split into 7 `fast` tasks and 1 `slow` task. This division is a simplification of the real system behaviour, since the scheduler does not pin the threads in a fixed way. The tasks service time, μ_{in} , is computed through the Equation 5.1. Instead, the expected response time, μ_{out} , is the inverse of the mean of the Gaussian fit. For the DB12 benchmark, the service time, μ_{out} , is extracted from Table 4.6 (Fig. 4.41) for the `ALONE` (`COUPLE`) queue. For the WSN benchmark, the service time, μ_{out} , is extracted from Table 4.6 (Fig. 4.42) for the `ALONE` (`COUPLE`) queue. For the KV benchmark, the service time in the `ALONE` queue, μ_{out} , is extracted from Fig. 4.39 (Fig. 4.40) for the `fast` (`slow`) task; whereas the service time in the `COUPLE` queue, μ_{out} , is extracted from Fig. 4.46 (Fig. 4.47) for the `fast` (`slow`) task. Table 5.3 summarises, for DB12 and WSN, the service time, μ_{in} , and the expected response time, μ_{out} . Table 5.4 summarises, for the KV benchmark, the service time, μ_{in} , and the expected response time, μ_{out} .

DB12			WSN	
	ALONE [s]	COUPLE [s]	ALONE [s]	COUPLE [s]
μ_{out}	0.0475	0.0985	$3,10 \times 10^{-2}$	$3,57 \times 10^{-2}$
μ_{in}	0.0434	0.0971	$3,01 \times 10^{-2}$	$3,45 \times 10^{-2}$

Table 5.3: Summary of the expected response times, μ_{out} , and service time, μ_{in} for DB12 and WSN in the VM-8₃ configuration.

ALONE			COUPLE	
	fast [s]	slow [s]	fast [s]	slow [s]
μ_{out}	0.7134	0.7275	1.3602	1.3938
μ_{in}	0.7033	0.7174	1.3228	1.3385

Table 5.4: Summary of the expected response times, μ_{out} , and service time, μ_{in} for KV benchmark in the VM-8₃ configuration.

To verify the accuracy of the simulation, the model results are compared to the measurements of the real system. In order to do so, the model results are converted into throughput. The Gaussian fit of the measurement-based approach are plotted on the same frame of the distributions of the model-based approach. In particular, the dot markers identify the JMT model throughput, whereas the cross markers

identify the measurement-based distribution. For the DB12 benchmark in Fig. 5.7, the model provides a good approximation of the curve that describes the performance of the VM executing alone. The right curve, that models the VM-8₃ - couple, is not exact since it suffers from an inexact evaluation of the variance. For the WSN benchmark simulation in Fig. 5.8, the model well approximates the Gaussian fit of the measurements. For the KV simulation in Fig. 5.9, the model well approximates the real system profile. In particular, the peaks of the VM8₃ - couple and VM8₃ - single executions are accurately reproduced. Therefore, the model in Fig. 5.6 has been validated for the VM-8₃ scenario executing the DB12, WSN and KV benchmarks.

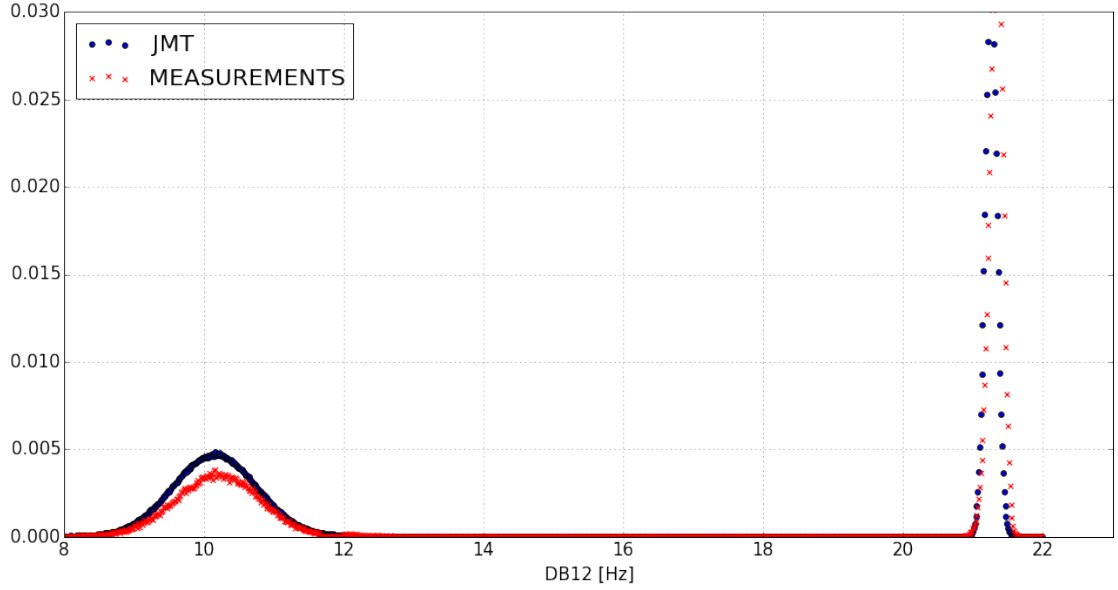


Figure 5.7: Distribution DB12 results in the VM-8₃. The red curve identifies the Gaussian fit of analysis' data, whereas the blue curve defines the distribution of results of the model simulation.

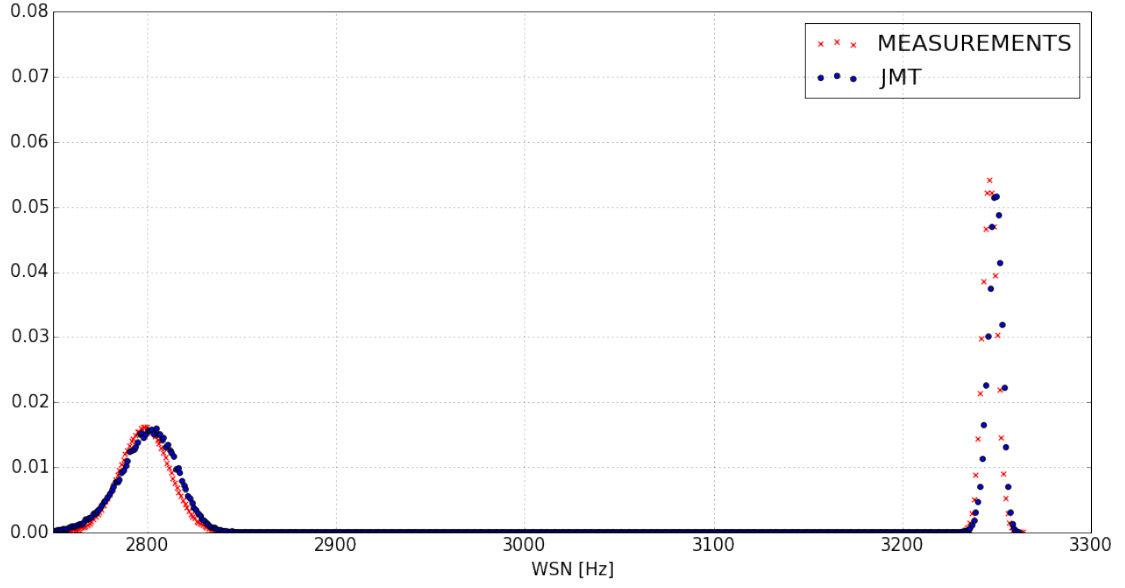


Figure 5.8: Distribution of WSN results in the VM-83. The red curve identifies the Gaussian fit of analysis' data, whereas the blue curve defines the distribution of results of the model simulation.

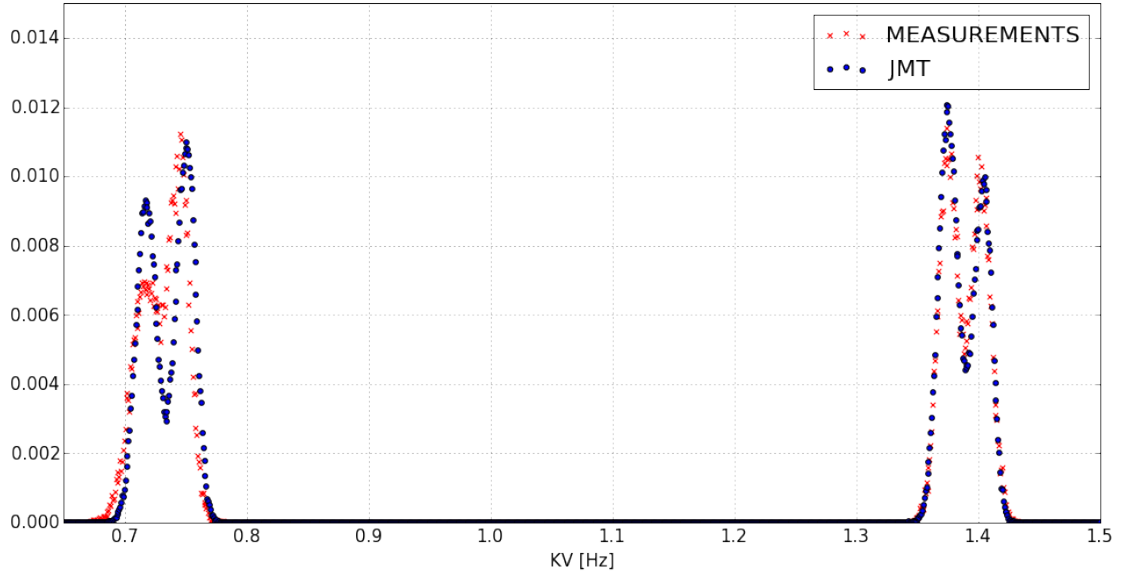


Figure 5.9: Distribution of KV results in the VM-83. The red curve identifies the Gaussian fit of analysis' data, whereas the blue curve defines the distribution of results of the model simulation.

The three models simulating the VM-16 and VM-8, in Fig. 5.1, Fig. 5.4 and Fig. 5.6, have been validated with the simulation approach explained in the above sections. Since the models correctly simulate the benchmarks executions, they can be exploited, in future studies, for the evaluation of different parametrization of the system. For example, in order to verify the performance of the server correlating a different configuration.

Conclusions

This work presents the performance analysis of a set of servers installed in the CERN’s data centre and used for the LHC computing activity. The study has been pursued with two distinct approaches: the performance measurement and the model based approach. Both approaches analyse the performance of various VMs’ configurations – ranging from a single virtual CPU to 32 virtual CPUs per VM – when running four different benchmark applications, namely HS06, DB12, WSN and KV. The measurement-based and the model-based approaches have unveiled complementary aspects of the system.

From the measurements point of view, the analysis provides, for each benchmark application, the relative performance gap among VM configurations. This result highlights discrepancies between the performance of the benchmark workloads, in particular among the fast benchmarks — DB12, WSN and KV — and the long running HS06. In addition, the KV measurements revealed a difference in performance between threads on the same CPUs. Further studies conducted at CERN [69] have verified this behaviour not only in the virtualised environment but also in the bare-metal servers. The aforementioned feature has not been observed in the VM-1 and VM-32 scenarios, due to the structure of their configurations that causes extreme variability of results, and averaged measurements respectively. Moreover, the study uncovered an unusual profile for the DB12 benchmark in the VM-32 configuration, reason being most likely due to an additional process running on the physical node.

The model-based approach, parametrised with the measurements results, has been used to simulate the real system in executing the DB12, WSN and KV benchmark in the VM-16 and VM-8 configurations. In order to exploit the potential of the models, they can be used as a starting point to study the system response to different workloads or server configuration.

Appendix A

Script to launch benchmarks

The set of virtual machines in the Nova cell have been provisioned through a Python script able to interface with Openstack through the Nova API. The script makes use of a configuration file, slightly different for each use case, which takes care of setting up different parameters such as the VM identifier, the flavor type, the image type and a path to a user-data file. The latter is a cloud-init user-data file used to contextualized the virtual machines and thus preparing the environment to run the benchmarks. It includes the sequence of authorized users' SSH keys and the commands to install dependencies and configurations needed to actually run the benchmarks as well as to clone the cern-benchmark GitLab repo, to test the branches and to install the suite into the VMs.

The user-data's final goal is to schedule the cron jobs to run the benchmarks. It should be remembered that the scheduling of the benchmarks depends on each considered scenario. VM-1, VM-4, VM-8 and VM-16 have benchmark' run every half an hour, whereas VM-32 has benchmark run every hour. The python script discovers which hypervisor the virtual machine belongs to. The physical node value is then passed to the script which is started by the cron jobs. In the following snippet of code [A.1](#) the cron jobs used to run the DB12, WSN and KV every half hour are displayed.

```
1 - echo '0,30 * * * * root /root/run\_bmk whetstone \&> /dev/null' > /  
  etc/cron.d/cern--benchmark--DB12  
2 - echo '5,35 * * * * root /root/run\_bmk DB12 \&> /dev/null' > /etc/  
  cron.d/cern--benchmark--whetstone  
3 - echo '15,45 * * * * root /root/run\_bmk kv \&> /dev/null' > /etc/  
  cron.d/cern--benchmark--kv
```

Listing A.1: Examples of cron jobs used to run the benchmarks.

Appendix B

The Elastic Family Tools

Figure B.1 shows the chain composed of the three open source tools, belonging to the Elastic family, that support the monitoring process of the virtual environment. Benchmark results are organised into JSON files which are collected by the message broker and parsed by Logstash to be injected into an Elasticsearch instance to index and store the relevant data which can be finally queried and visualised through the Kibana web interface.



Figure B.1: Monitoring tools flow.

B.1 Transportation Layer: Logstash

Logstash [90] is a tool dealing with data logging and event processing, used to ingest and parse logs in order to extract the relevant data from a wide variety of sources; it also has the capability to filter, manipulate and shape the data so that it is easier to work with. This data can be then serialized into a standard JSON (JavaScript Object Notation) format to be sent into the Elasticsearch layer via HTTP.

B.2 Storage Layer: Elasticsearch

Elasticsearch [91] is a real-time search index engine that relies on Lucene, an open-source information retrieval software library. This storage layer does not consist of tables and schemas such as relational databases. Referring to Fig. B.2, Elasticsearch stores data in the form of JSON documents inside an index, which is a collection of documents that have similar characteristics. The index can be split into multiple shards to achieve two goals: the first one is to improve the

performance throughput by parallelizing the search operations, the second one is to guarantee the service availability and the load balancing thanks to the data redundancy obtained by the shards replicas. The indexes are stored in one or more servers, which are called nodes. A node is a single Elasticsearch server, whose purpose is to archive and manage the data in order to participate in the indexing and search capabilities. Multiple sets of nodes can be organized in unique clusters, which allows the infrastructure to scale horizontally.

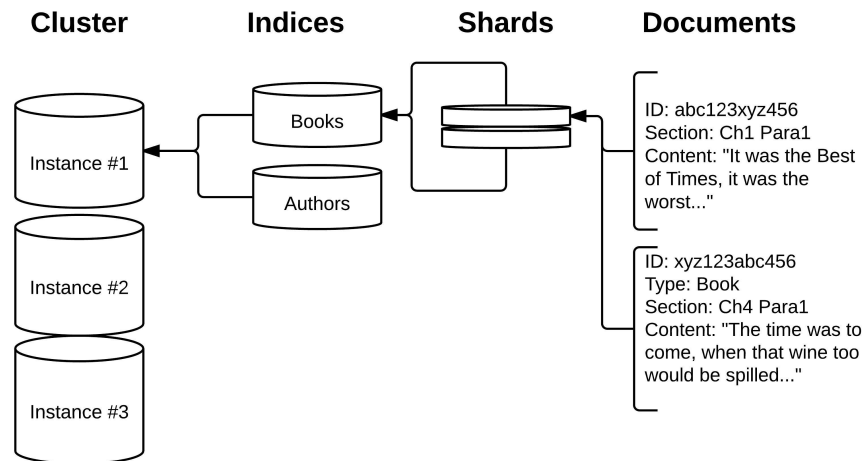


Figure B.2: Elasticsearch structure example [92].

B.3 Visualisation Layer: Kibana

One of the last steps in every evaluation study is the graphical presentation of final results which presents the information in a concise way and helps to clarify the point, to reinforce a conclusion, to summarise the results and to see trends and patterns. Kibana [93] is a data visualisation engine that allows for the building of custom and dynamic dashboards to show the data retrieved from Elasticsearch. Data can be deeply analysed and visualised (Fig. B.3) in terms of point-and-click pie charts, bar graphs, trend lines, maps and scatter plots, in order to extract useful trends and patterns on top of large volumes of data. The data consultation inside Kibana relies on the Elasticsearch's index concept.

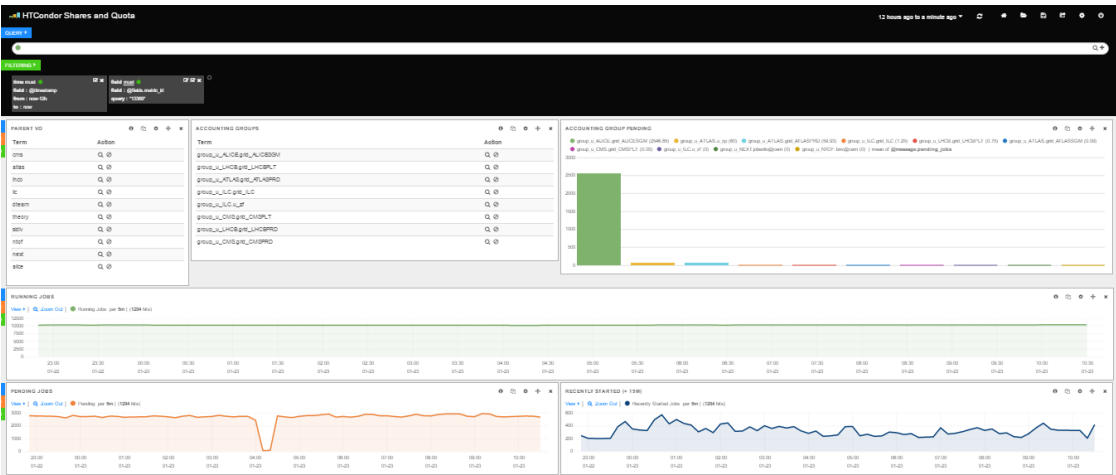


Figure B.3: Kibana visualization tool.

Appendix C

Interaction with Elasticsearch and Kibana

C.1 Query to Extract Data from Elasticsearch

In order to retrieve JSON files from Elasticsearch, a query to the system is performed as shown in the following code fragments. Before being able to connect to the Elasticsearch service is necessary to authenticate through the CERN Web Single Sign On (SSO) services. The Elasticsearch index from which JSONs are gathered is *sdccloud-int*. It should be noted that JSONs are retrieved in bunches defined by a starting and an ending date. For example, this script `script.SH "2016-06-20T00:00:00Z" "2016-06-27T23:59:59Z"` gathers all the benchmarks that run in the interval going from midnight of 20/06/2016, to one minute before midnight of 27/06/2016; which consists of 7 days running. The search API executes a search query that can either be provided using a simple query string as parameter, or using a request body. The result of the query is a JSON file, identified by the starting date and the ending date. The [C.1](#) snippet of code defines the parameter of the query submitted to Elasticsearch to obtain the query answer. In particular, the `range` parameter, retrieves the JSON file whose `message._timestamp` is inside a specific time interval. The `message._timestamp`, represents the interval when the JSONs is gathered. The `term` parameter searches for the documents that contain the exact terms specified. For the purpose of this thesis the aim was to analyse data coming from the cloud identified by `CERN-wig_project_011`. The `size` parameter configures the maximum amount of hits to be returned. It has been chosen to extract all data allocated in 24 hours. The `include` parameter identifies the values that must be included in the final JSON, on the contrary the `exclude`, identifies the values that must be excluded from the results since deemed useless for the thesis purpose.

```
1  "filtered": {
2    "query": {
3      "bool": {}
4    },
5    "filter": {
6      "bool": {
7        "must": [
```

```

8      {
9          "range": {
10             "message._timestamp": {
11                 "from": "' '$1'",
12                 "to": "' '$2'"
13             }
14         },
15     },
16     {
17         "terms": {
18             "message.metadata.cloud": [
19                 "CERN-wig_project_011"
20             ]
21         }
22     }
23 ]
24 }
25 }
26 }
27 },
28 "size": 10000000,
29 "partial_fields" : {
30     "bmks" : {
31         "include" : ["message._timestamp", "message.metadata
32                     .*", "message.profiles.rkv.*", "message.profiles.
33                     phoronix.*", "message.profiles.fastBmk.*", "
34                     message.profiles.whetstone.*"],
35         "exclude" : ["*.unit", "*.units", "*rkv*entries", "*speed*"]
36     }
37 }

```

Listing C.1: Query to JSON from Elastic.

The following snippet [C.2](#) of JSON code represents an example of the query result.

```

1  {
2      "_index": "sdcccloud-int",
3      "_type": "vmspec",
4      "_id": "the240s-901bfaac-d358-405a-b943-0a4accf56d24_7b9cc47e-6b51
5             -48cd-94c2-e3943856d941_2016-07-21T15:05:02Z",
6      "_score": null,
7      "_source": {
8          "message": {
9              "_timestamp": "2016-07-21T15:05:02Z",
10             "_id": "the240s-901bfaac-d358-405a-b943-0a4accf56d24_7b9cc47e-6
11                  b51-48cd-94c2-e3943856d941_2016-07-21T15:05:02Z",
12             "profiles": {
13                 "fastBmk": {
14                     "value": "8.01781737194"
15                 }
16             },
17             "metadata": {
18                 "benchmark_target": "machine",
19                 "meminfo": 1876284,
20                 "UID": "the240s-901bfaac-d358-405a-b943-0a4accf56d24_7b9cc47e
21                     -6b51-48cd-94c2-e3943856d941",

```

```

19         "classification": "i6_1_f6m63s2_mhz2394.452",
20         "freetext": "VM-1",
21         "cpuname": "Intel(R) Xeon(R) CPU E5-2630 v3 @ 2.40GHz",
22         "ip": "188.185.184.233",
23         "osdist": "Scientific Linux CERN SLC release 6.8 (Carbon)",
24         "bogomips": 4788.9,
25         "VO": null,
26         "cpunum": 1,
27         "pnode": "P06253971G07182",
28         "mp_num": "1",
29         "pyver": "2.6.6",
30         "cloud": "CERN-wig_project_011"
31     }
32 }
33 "@timestamp": "2016-07-21T13:06:55.539Z"
34 },
35 "fields": {
36     "message._timestamp": [
37         1469113502000
38     ]
39 }
40 }

```

Listing C.2: Example of JSON from Elastic.

C.2 Benchmark Details in the Kibana Dashboard

Kibana can show the details of the benchmarks completed in each virtual machine as in Fig. C.1. From this page it is possible to extract information about both the benchmarking score and the VM's metadata. This data can be retrieved in Elasticsearch through a JSON file.

@timestamp	Q Q July 21st 2016, 15:06:55.539
@version	Q Q 1
_id	Q Q the240s-901bfaac-d358-405a-b943-0afaccf56d24_7b9cc47e-6b51-48cd-94c2-e3943856d941_2016-07-21T15:05:02z
_index	Q Q sdcccloud-int
_source	Q Q {"message":{"_timestamp":"2016-07-21T15:05:02z","_id":"the240s-901bfaac-d358-405a-b943-0afaccf56d24_7b9cc47e-6b51-48cd-94c2-e3943856d941_2016-07-21T15:05:02z","profiles":{"fastBmk":{"unit":"test_H506","value":"8.017817377194"},"metadata":{"benchmark_target":"machine","meminfo":{"the240s-901bfaac-d358-405a-b943-0afaccf56d24_7b9cc47e-6b51-48cd-94c2-e3943856d941"},"classification":"i6_1_f6m63s2_mhz2394.452","freetext":"VM-1","cpuname":"Intel(R) Xeon(R) CPU E5-2630 v3 @ 2.40GHz","ip":"188.185.184.233","osdist":"Scientific Linux CERN SLC release 6.8 (Carbon)","bogomips":4788.9,"VO":null,"cpunum":1,"pnode":"P06253971G07182","mp_num":"1","pyver":"2.6.6","cloud":"CERN-wig_project_011"},"@version":"1","@timestamp":"2016-07-21T13:06:55.539z"}}
_type	Q Q vmspec
message._id	Q Q the240s-901bfaac-d358-405a-b943-0afaccf56d24_7b9cc47e-6b51-48cd-94c2-e3943856d941_2016-07-21T15:05:02z
message._timestamp	Q Q July 21st 2016, 17:05:02.000
message.metadata.UUID	Q Q the240s-901bfaac-d358-405a-b943-0afaccf56d24_7b9cc47e-6b51-48cd-94c2-e3943856d941
message.metadata.VO	Q Q
message.metadata.benchmark_target	Q Q machine
message.metadata.bogomips	Q Q 4788.9
message.metadata.classification	Q Q i6_1_f6m63s2_mhz2394.452
message.metadata.cloud	Q Q CERN-wig_project_011
message.metadata.cpuname	Q Q Intel(R) Xeon(R) CPU E5-2630 v3 @ 2.40GHz
message.metadata.cpunum	Q Q 1
message.metadata.freetext	Q Q VM-1
message.metadata.ip	Q Q 188.185.184.233
message.metadata.meminfo	Q Q 1876284
message.metadata.mp_num	Q Q 1
message.metadata.osdist	Q Q Scientific Linux CERN SLC release 6.8 (Carbon)
message.metadata.pnode	Q Q P06253971G07182
message.metadata.pyver	Q Q 2.6.6
message.profiles.fastBmk.unit	Q Q test_H506
message.profiles.fastBmk.value	Q Q 8.017817377194

Figure C.1: Detail of a virtual machine benchmarking.

Table C.1 describes the meaning of the most useful and interesting metadata taken from the previous example in Fig. C.1.

Metadata	Description
msg._id	Name of the VM on which the benchmark is running with date and hour
msg._timestamp	Benchmark start time
msg.metadata.UID	Name of the VM on which the benchmark is running
msg.metadata.cloud	Cloud in which the VM is
msg.metadata.cpuname	Name of the CPU
msg.metadata.cpunum	Number of virtual cores
msg.metadata.freetext	Name to identify the use case
msg.metadata.ip	IP address of the VM
msg.metadata.osdist	Operating system
msg.metadata.pnode	Name of the physical host
msg.profiles.fastBmk.value	Value of the benchmark

Table C.1: Meaning of Kibana metadata.

Appendix D

Boxplot

A boxplot [94] is a graphical method to display a dataset, since it gives information regarding data central tendency, median, spread and outliers. The boxplot splits the data set into quartiles (see Fig. D.1), which are cutpoints that divide the range of a probability distribution into contiguous intervals with equal probabilities. The latter concept is strictly correlated to the percentile, that is a measure used to indicate a value below which a given percentage of observations in a group of observations fall. The 50-percentile, or median, identifies the mid-point of the dataset so that half of the values are greater than or equal to this value and half are less. The same definition can be used for the upper and lower quartiles: below the upper quartile fall the 75% of the values, whereas above the lower quartile fall the 25% of values. The distance between lower and higher quartile is named inter quartile range (IQR). Other useful information of the boxplot are the whiskers and the outliers. The whiskers identify the values inside the range $[(25\% - 1.5 * IQR), (75\% + 1.5 * IQR)]$, whereas beyond the whiskers, data are considered outliers and plotted as individual points.

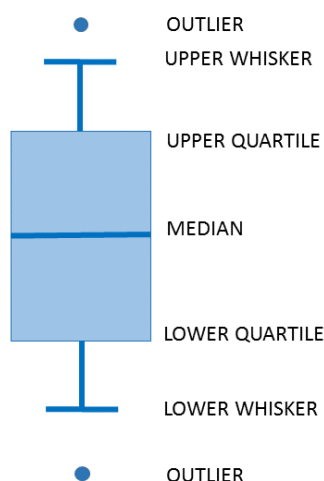


Figure D.1: Main components of a boxplot.

A boxplot is a way of graphically examining one or more sets of data since it compares distributions verifying the central value and spread of the data with respect to the median value. Moreover, the distance between the smallest and the

largest value, including any outliers, identifies the spread of the data. If the box is comparatively short, the overall dataset has a low level of variability, on the contrary, if the box is comparatively long, the dataset has a high level of variability. Therefore, Fig. D.2 relates a boxplot with its statistical distribution summarising the different cases.

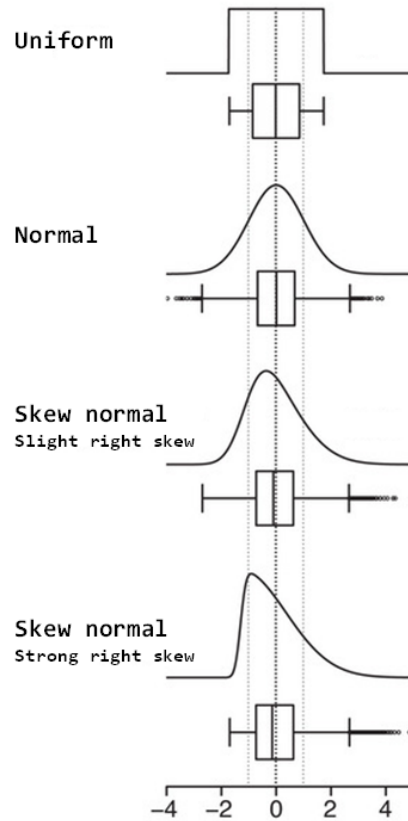


Figure D.2: Relation between the boxplot and the distribution. Figure (a) shows a normal gaussian, Figure (b) shows a thin gaussian whereas Figure (c) shows a large gaussian. Figure (d) describes an asymmetric distribution.

Bibliography

- [1] *The Digital Universe of Opportunities: Rich Data and the Increasing Value of the Internet of Things*. emc.com/leadership/digital-universe/2014iiview/executive-summary.htm. EMC Digital Universe, 2014 (cit. on p. 4).
- [2] D. Laney. “3-D Data Management: Controlling Data Volume, Velocity and Variety”. In: (2001) (cit. on p. 4).
- [3] Matthew Finnegan. *Boeing 787s to create half a terabyte of data per flight*. computerworlduk.com/data/boeing-787s-create-half-terabyte-of-data-per-flight-says-virgin-atlantic-3433595/. 2013 (cit. on p. 5).
- [4] J. Pearlstein. *Information Revolution: Big Data Has Arrived at an Almost Unimaginable Scale*. wired.com/2013/04/bigdata/. Wierd Magazine, 2013 (cit. on p. 5).
- [5] *International Network of Crisis Mappers*. crisismappers.net/. INCM (cit. on p. 5).
- [6] *MapReduce Tutorial*. https://hadoop.apache.org/docs/r1.2.1/mapred_tutorial.html. Apache software foundation, 2013 (cit. on p. 6).
- [7] C. Kesselman and I. Foster. *The Grid: Blueprint for a New Computing Infrastructure*. San Francisco, CA: Morgan and Kaufmann, 1999 (cit. on p. 7).
- [8] B. Nasser et al. “Access Control Model for Inter-organizational Grid Virtual Organizations”. In: *On the Move to Meaningful Internet Systems 2005: OTM 2005 Workshops*. 2005, pp. 537–551 (cit. on p. 7).
- [9] Eng Anwar J Alzaid and Eng Jassim M Albazzaz. “Cloud Computing: An Overview”. In: *International Journal of Advanced Research in Computer and Communication Engineering* 2.9 (2013), pp. 3522–3525 (cit. on p. 8).
- [10] *App Engine*. cloud.google.com/appengine/. Microsoft, 2010 (cit. on p. 9).
- [11] *Categorizing the Cloud...* blogs.msdn.microsoft.com/johnalioto/2010/08/16/categorizing-the-cloud/. Google Cloud Platform, 2016 (cit. on p. 10).
- [12] *Microsoft’s Office Web Apps*. <https://office.com/>. Microsoft, 2016 (cit. on p. 10).
- [13] I. T. Foster et al. “Cloud Computing and Grid Computing 360-Degree Compared”. In: *CoRR* abs/0901.0131 (2009) (cit. on p. 11).

- [14] *Conseil Européen pour la Recherche Nucléaire*. home.cern/. CERN (cit. on p. 12).
- [15] *CERN LHC: the guide. faq. frequently asked questions*. Geneva: CERN, 2006 (cit. on p. 13).
- [16] *Die Opensource Grid*. www.linux-magazin.de/Ausgaben/2004/06/Alien-im-Wunderland. Linux Magazin (cit. on p. 13).
- [17] *ALICE experiment*. aliceinfo.cern.ch/Public/Welcome.html. CERN (cit. on p. 14).
- [18] *ATLAS experiment*. atlas.cern/. CERN (cit. on p. 14).
- [19] *CMS experiment*. cms.web.cern.ch/. CERN (cit. on p. 14).
- [20] *LHCb experiment*. lhcb.web.cern.ch/lhcb/. CERN (cit. on p. 14).
- [21] *"Black Hole" event superimposed over a classic image of the ATLAS detector*. www.atlasexperiment.org/photos/full-detector-photos.html. CERN (cit. on p. 14).
- [22] *CMS detector in a cavern 100 m underground at CERN's Large Hadron Collider*. cms.web.cern.ch/news/what-cms. CERN (cit. on p. 14).
- [23] *ALICE's "front absorber", in the centre of the detector, after installation and alignment*. www.theguardian.com/science/gallery/0007/nov/20/cern. CERN (cit. on p. 14).
- [24] *General view of the LHCb detector component*. <http://cds.cern.ch/record/1090809>. CERN (cit. on p. 14).
- [25] I. G. Bird. "LHC computing (WLCG): Past, present, and future". In: *Grid and Cloud Computing: Concepts and Practical Applications* 192 (2016), pp. 1–29 (cit. on pp. 15, 21).
- [26] *CERN Announces New LHC Discovery*. www.redorbit.com/news/science/1920830/cern_announces_new_lhc_discovery/. CERN (cit. on p. 15).
- [27] I. G. Bird. "Computing for the Large Hadron Collider". In: *Annual Review of Nuclear and Particle Science* (cit. on p. 16).
- [28] *Diagram showing the tier system of WLCG*. sciencenode.org/feature/large-hadron-colliders-worldwide-computer.php. CERN (cit. on p. 16).
- [29] *The Worldwide LHC Computing Grid (WLCG)*. indico.cern.ch/event/563741/contributions/2278074/attachments/1337610/2012822/Cyprus-150916.pdf. CERN (cit. on p. 17).
- [30] *GEANT*. www.geant.org/. GEANT (cit. on p. 17).
- [31] *Energy Science Network*. www.es.net/. ESNET (cit. on p. 17).
- [32] A. Bartels. *Data Center Evolution: 1960 to 2000*. <http://blog.rackspace.com/datacenter-evolution-1960-to-2000>. Rackspace, 2011 (cit. on p. 19).

- [33] *History of computing in 1960*. www-03.ibm.com/ibm/history/history/decade_1960.html. ibm (cit. on p. 19).
- [34] *History of computing in 1980*. www-03.ibm.com/ibm/history/history/decade_1980.html. ibm (cit. on p. 19).
- [35] *Computing at CERN history*. timeline.web.cern.ch/timelines/Computing-at-CERN. cern (cit. on p. 20).
- [36] *UNIX*. en.wikipedia.org/wiki/UNIX. Wikipedia (cit. on p. 20).
- [37] *Centre du calcul du CERN*. home.cern/about/computing. Wikipedia (cit. on p. 20).
- [38] *CERN Data Center*. <http://information-technology.web.cern.ch/about/computer-centre>. CERN, 2013 (cit. on p. 20).
- [39] A. J. Peters and L. Janyst. “Exabyte scale storage at CERN”. In: *Journal of Physics: Conference Series*. Vol. 331. 5. IOP Publishing. 2011, p. 052015 (cit. on p. 21).
- [40] *Monitoring dashboard of the CERN Data Center*. <https://meter.cern.ch>. CERN, 2016 (cit. on p. 21).
- [41] *Hardware virtualization*. en.wikipedia.org/wiki/Hardware_virtualization. Wikipedia (cit. on p. 22).
- [42] *Hyper-V*. [technet.microsoft.com/en-us/library/hh831531\(v=ws.11\).aspx](http://technet.microsoft.com/en-us/library/hh831531(v=ws.11).aspx). Microsoft (cit. on p. 23).
- [43] *Xen Server*. www.citrix.it/products/xenserver/. citrix (cit. on p. 23).
- [44] *VM Server for Sparc*. www.oracle.com/virtualization/oracle-vm-server-for-sparc/index.html. Oracle (cit. on p. 23).
- [45] *VM Server for x86*. www.oracle.com/virtualization/vm-server-for-x86/index.html. Oracle (cit. on p. 23).
- [46] *VMware ESX/ESXi*. www.vmware.com/it/products/esxi-and-esx.html. vmware (cit. on p. 23).
- [47] *VMware Workstation*. en.wikipedia.org/wiki/VMware_Workstation. Wikipedia (cit. on p. 23).
- [48] *VMware Player*. en.wikipedia.org/wiki/VMware_Workstation_Player. Wikipedia (cit. on p. 23).
- [49] *Virtualbox*. www.virtualbox.org/. Oracle (cit. on p. 23).
- [50] *QEMU documentation*. wiki.qemu.org/Main_Page. GNU (cit. on p. 23).
- [51] *KVM documentation*. help.ubuntu.com/community/KVM. ubuntu (cit. on p. 23).
- [52] *Scientific research runs on OpenStack*. <https://openstack.org/software/>. OpenStack, 2016 (cit. on p. 24).
- [53] T. Bell et al. “Scaling the CERN OpenStack cloud”. In: *J. Phys.: Conf. Ser.* 664.2 (2015), 022003. 9 p (cit. on p. 24).

- [54] *Swift documentation*. docs.openstack.org/developer/swift/. OpenStack (cit. on p. 24).
- [55] *Neutron documentation*. docs.openstack.org/developer/neutron/. OpenStack (cit. on p. 24).
- [56] *Ceilometer documentation*. docs.openstack.org/developer/ceilometer/. OpenStack (cit. on p. 24).
- [57] *Heat documentation*. docs.openstack.org/developer/heat/. OpenStack (cit. on p. 24).
- [58] *Trove documentation*. docs.openstack.org/developer/trove/. OpenStack (cit. on p. 24).
- [59] *Sahara documentation*. docs.openstack.org/developer/sahara/. OpenStack (cit. on p. 24).
- [60] *Ironic documentation*. docs.openstack.org/developer/ironic/deploy/user-guide.html. OpenStack (cit. on p. 24).
- [61] *Nova documentation*. docs.openstack.org/developer/nova/. OpenStack (cit. on p. 24).
- [62] A. Wiebalck. “Compute Resource Provisioning for the Large Hadron Collider” (cit. on p. 28).
- [63] *Glance documentation*. docs.openstack.org/developer/glance/. OpenStack (cit. on p. 28).
- [64] *Cinder documentation*. docs.openstack.org/developer/cinder/. OpenStack (cit. on p. 28).
- [65] *Horizon documentation*. docs.openstack.org/developer/horizon/. OpenStack (cit. on p. 28).
- [66] *Keystone documentation*. docs.openstack.org/developer/keystone/. OpenStack (cit. on p. 29).
- [67] *Load Sharing Facility Platform*. help.unc.edu/help/lsf-load-sharing-facility/. University of North Carolina (cit. on p. 29).
- [68] *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling*. Wiley Interscience, 1992 (cit. on p. 31).
- [69] *Benchmarking Cloud Resources for HEP*. linux.web.cern.ch/linux/scientific6/. CERN, 2017 (cit. on pp. 31, 55, 80, 94).
- [70] *Intel Xeon Processor*. ark.intel.com/products/83356/Intel-Xeon-Processor-E5-2630-v3-20M-Cache-2_40-GHz. Intel (cit. on p. 32).
- [71] *Non Uniform Memory Access*. en.wikipedia.org/wiki/Non-uniform_memory_access. Wikipedia (cit. on p. 33).
- [72] *Samsung Enterprise SSD*. www.samsung.com/semiconductor/products/flash-storage/enterprise-ssd/MZ7KM960HAHP?ia=832. Samsung (cit. on pp. 34, 35).

- [73] *Samsung's 5th Generation Green Memory Solution* (cit. on p. 34).
- [74] *Linux at CERN*. linux.web.cern.ch/linux/scientific6/. CERN, 2015 (cit. on p. 35).
- [75] *Apache ActiveMQ*. activemq.apache.org/. Apache (cit. on p. 38).
- [76] This unpublished picture was provided by courtesy of Domenico Giordano. "CERN Cloud Benchmarking Suite" (cit. on p. 38).
- [77] *Dirac Benchmark 2012*. gitlab.cern.ch/mcnab/dirac-benchmark/tree/master. CERN (cit. on p. 40).
- [78] Philippe Charpentier. "Benchmarking worker nodes using LHCb simulation productions and comparing with HEP-Spec06". In: (2016). URL: <https://cds.cern.ch/record/2229747> (cit. on p. 40).
- [79] R. Longbottom. *Whetstone Benchmark History and Results*. roylongbottom.org.uk/whetstone.htm. HEPiX Benchmarking Working Group, 2014 (cit. on p. 40).
- [80] *HEPiX Benchmarking Working Group*. www.hepux.org/e10227/e10327/e10325. HEPiX (cit. on p. 40).
- [81] C. D. Spradling. "SPEC CPU2006 Benchmark Tools". In: *SIGARCH Comput. Archit. News* 35 (Mar. 2007), pp. 130–134 (cit. on p. 41).
- [82] *How to run the HEP-SPEC06 benchmark*. w3.hepux.org/benchmarks/doku.php?id=bench:howto. HEPiX (cit. on pp. 41, 42).
- [83] *iPython notebooks*. ipython.org/notebook.html. iPython (cit. on p. 43).
- [84] *Python Data Analysis Library*. pandas.pydata.org/. NUMFocus (cit. on p. 43).
- [85] *Pandas DataFrame*. pandas.pydata.org/pandas-docs/stable/dsintro.html#dataframe. Pandas (cit. on p. 44).
- [86] *Tutorial on Linux kernel profiling*. perf.wiki.kernel.org/index.php/Tutorial. Google, 2015 (cit. on p. 55).
- [87] Marco Guerri, Domenico Giordano, and Cordeiro Cristovao. *Profiling CPU-bound workloads on Intel Haswell-EP platforms*. Tech. rep. CERN-IT-Note-2017-002. Geneva: CERN, 2017. URL: <https://cds.cern.ch/record/2257973> (cit. on p. 55).
- [88] E.D. Lazowska. *Quantitative System Performance: Computer System Analysis Using Queueing Network Models*. Prentice Hall, 1984. URL: <https://books.google.it/books?id=9QwfvvAACAkJ> (cit. on p. 82).
- [89] *Java Modelling Tools*. jmt.sourceforge.net/. Politecnico di Milano, Imperial College of London, 2017 (cit. on p. 83).
- [90] *Logstash*. elastic.co/products/logstash. elastic (cit. on p. 96).
- [91] Elastic. *Elasticsearch Reference*. elastic.co/guide/en/elasticsearch/reference/current/index.html (cit. on p. 96).

- [92] Stephen Puiszis. *Getting Started with Elasticsearch*. stephen.fm/getting-started-with-elasticsearch/ (cit. on p. 97).
- [93] Elastic. *Kibana User Guide*. <https://elastic.co/guide/en/kibana/current/index.html> (cit. on p. 97).
- [94] *Boxplot*. en.wikipedia.org/wiki/Box_plot. Wikipedia (cit. on p. 103).