VSB TECHNICAL UNIVERSITY OF OSTRAVA | FACULTY OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

HEPSCORE integration for WLCG accounting

Integrace HEPSCORE pro WLCG accounting

Bc. Leona Žůrková

Diploma Thesis

Supervisor: Ing. Lukáš Vojáček, Ph.D.

Ostrava, 2023

Diploma Thesis Assignment

Student: **Bc. Leona Žůrková**

Study Programme: N2647 Information and Communication Technology

Study Branch: 2612T025 Computer Science and Technology

Title: HEPSCORE Integration for WLCG Accounting
Integrace HEPSCORE pro WLCG Accounting

The thesis language: English

Description:

The thesis discusses work carried out in CERN Information Technology department, in a group that is responsible for the CERN compute infrastructure, including the accounting system which tracks the usage of compute resources for all the CERN communities, and reports to the World-wide LHC Computing Grid (WLCG) management.
The main topic of the thesis is about changing the workflow for the CERN compute accounting data processing and aggregation, as the new HEPSCORE hardware benchmark will replace the current HEPSPEC06 benchmark.
The main goal is to propose and implement the changes needed in the CERN compute accounting system to accommodate a different data format provided by the new benchmark.
The student will learn fundamentals of the existing CERN and WLCG accounting system, the HTCondor configuration, the batch share management tool and the way the benchmarking meta-data is propagated through the system. The student will gain a knowledge of how accounting workflow works, and how the hardware benchmarking tool is used. If successful, the thesis will be a significant contribution to CERN.

1. Analyze the current environment and technology used. Understand what benchmarking is, and how it is used in the WLCG accounting workflow.
2. Understand what the score returned by HEPSCORE represents. Learn about the structure of data provided by the new benchmark.
3. Suggest an approach to allow for sensible labeling of features in the benchmark data structure.
4. Update the Batch accounting system to accept the suggested data format.
5. Evaluate and compare the results.

References:

[1] C++: Step by step Beginners Guide in Mastering C++, Liam Damien, November 26, 2019, ASIN: B0822PN19S
[2] Modern Computer Architecture and Organization: Learn x86, ARM, and RISC-V architectures and the design of smartphones, PCs, and cloud servers, 2nd Edition 2nd ed. Edition; Jim Ledin, Dave Farley; Packt Publishing; 2nd ed. edition (May 4, 2022), ISBN-10:1803234512
[3] Linux for Beginners: A Practical and Comprehensive Guide to Learn Linux Operating System and Master Linux Command Line. Contains Self-Evaluation Tests to Verify Your Learning Level. Ethem Mining, Independently published (December 3, 2019), ISBN-10: 1671228081.

Extent and terms of a thesis are specified in directions for its elaboration that are opened to the public on the web sites of the faculty.

Supervisor: **Ing. Lukáš Vojáček, Ph.D.**

Consultant: Dr. Gavin McCance
Mgr. Jaroslava Schovancová, Ph.D.

Date of issue: 01.09.2022
Date of submission: 30.04.2023

Fields of study guarantor: prof. RNDr. Václav Snášel, CSc.
In IS EDISON assigned: 07.11.2022 11:59:22

Abstrakt

Tato diplomová práce se zabývá optimalizací procesu účtování výpočetních prostředků (CPU accounting) v rámci Worldwide LHC Computing Grid (WLCG) na CERNu. Studie se zaměřuje na analýzu stávajícího prostředí, porozumění metrikám benchmarkingu, jako je HEPSCORE, a navrhování integrovaného přístupu pro začleňování nových benchmarků. Výzkum zdůrazňuje nutnost aktualizace kódu Spark jobs pro zahrnutí hodnot HEPSCORE a efektivní plánování a reportování do externí služby APEL. Díky spolupráci s kolegy a využití námi vytvořeného testovacího prostředí je dosaženo úspěšné integrace. Výsledky práce poukazují na důležitost dokumentace a sdílení znalostí pro udržitelné zlepšování v komplexních technických prostředích.

Klíčová slova

CPU accounting, benchmarking, WLCG, alokace prostředků, Spark jobs, HEPSCORE, integrace, efektivita plánování, dokumentace

Abstract

This thesis investigates the enhancement of the CPU accounting workflow in the Worldwide LHC Computing Grid (WLCG) at CERN. The study focuses on analyzing the existing environment, understanding benchmarking metrics like HEPSCORE, and proposing an integrated approach for incorporating new benchmarks. The research highlights the need for updating Spark jobs code to include HEPSCORE values and streamline reporting to the third-party service, APEL. By collaborating with colleagues and utilizing a dedicated testing environment, successful integration is achieved. The findings emphasize the importance of documentation and knowledge sharing for sustained improvements in complex technical environments.

Keywords

CPU accounting, benchmarking, WLCG, resource allocation, Spark jobs, HEPSCORE, integration, efficiency, documentation

Acknowledgement

I would like to express my sincere gratitude to my colleagues at CERN, who provided valuable support and guidance throughout my journey. Their expertise and willingness to share their knowledge helped me understand the intricacies of the work environment and contributed significantly to the successful completion of this thesis. My biggest thanks belongs to Gavin McCance, who provided me the initial insights, and Maria Alandes Pradillo, who helped me in more hands-on tasks.

I would also like to extend my heartfelt thanks to my family in Czechia and flatmates in France for their unwavering support, encouragement, and understanding during this challenging phase. Their constant belief in me and their patience in accommodating my demanding schedule played a crucial role in my ability to focus and excel in my studies.

Additionally, I would like to acknowledge the support and guidance provided by my thesis advisor, Lukáš Vojáček, whose expertise and feedback were instrumental in shaping the direction of this research.

Finally, I am grateful to all the individuals who have contributed to my academic journey in various ways, whether through their insightful discussions, feedback on drafts, or general encouragement. Your contributions have enriched my understanding and have been invaluable to the completion of this thesis.

Thank you all for your indispensable support, both personally and professionally. Your belief in me and your willingness to help have made this endeavor possible.

Contents

List of symbols and abbreviations

IT	–	Information Technology
WWW	–	World Wide Web
YAML	–	Yet Another Markup Language
HTC	–	High Throughput Computing
CPU	–	Central Processing Unit
GPU	–	Graphics Processing Unit
HEP	–	High Energy Physics
CERN	–	Conseil Européen pour la Recherche Nucléaire
LHC	–	Large Hadron Collider
HL-LHC	–	High-Luminosity Large Hadron Collider
ALICE	–	A Large Ion Collider Experiment
ATLAS	–	A Toroidal LHC Apparatus
CMS	–	Compact Muon Solenoid
LHCb	–	LHC Beauty
WLCG	–	World LHC Computing Grid
VM	–	Virtual Machine
CVMFS	–	CernVM File System
SPEC	–	Standard Perfomance Evalutation Corporation
CINT	–	CPU INTeger
SI2K	–	SPEC CINT2000
HS06	–	HEP-SPEC 2006
HS23	–	HEPSCORE 2023
WL	–	workload

GEN	–	generation
SIM	–	simulation
DIGI	–	digitalisation
RECO	–	reconstruction

List of Figures

List of Tables

Listings

Chapter 1

Introduction

The pace of technological advancement has been accelerating at an unprecedented rate in recent years. The impact of new technologies on our daily lives and society as a whole is profound. Digital Darwinism is unkind to those who wait. It's crucial to understand to the importance of adaptability and collaboration in the face of change and uncertainty. This is the idea behind Darwin's book The Origin of Species[1], promoting the gravity of evolution, and the ideas can be also applied to technology.

This project focuses on the update of the CPU accounting workflow for WLCG. The way that computing resources are allocated and managed has a profound impact on the performance and efficiency of computer systems. As the demands placed on computer systems continue to grow, it is becoming increasingly important to optimize the allocation of resources. A concrete example of this is CERN's construction of a new datacentre at the Prevessin site, intended to accommodate the anticipated data loads from the planned High-Luminosity LHC and later Future Circular Collider, which is projected to be 4 times larger than the LHC to achieve better conditions for bigger discoveries. As the scale of resources expands, the imperative to ensure precise CPU accounting and effective resource handling becomes all the more crucial.

The goal of this project is to provide a comprehensive overview of the current state of the WLCG accounting workflow at CERN and describe integrating a new benchmarking unit to optimize the allocation of computing resources, and to make WLCG more efficient, effective, and adaptable.

Important part of this work is the accounting workflow analysis as well, as a part of all the information about it are fragmented in different information channels, and there is no central source of knowledge for it. A big part of the knowledge used to be in the heads of my colleagues, until I came to discover and capture it on this paper.

First in the Introduction and Background section you'll read about CERN, WLCG, Pledge, Accounting, Benchmarking, or general tools used in grid computing and CPU accounting to understand how this project fits into the real world. Later in Analysis chapter, you'll learn more about CERN-specific tools and environment. Then I'll introduce you to the decisions and changes made by higher

authorities, and I'll finish this thesis by writing about my decisions and the implementation.

1.1 Motivation

The HEP-SPEC2006 (HS06) benchmark has been a reliable estimate of CPU performance for many years, and is currently used by the WLCG for accounting and pledges. However, HS06 is based on the SPEC2006 benchmark that is no longer supported. Further, it uses applications that do not reflect those used by the HEP community.[1]. One specific case is that HS06 has bad scaling properties with the LHCb applications, in particular with the simulation (CPU-bound) application.[2] I'll share a few nice sentences with you from a book called The art of computer systems performance analysis[3] to summarize the reasons why do we aim for the accounting to be as precise as possible:

- "Performance is a key criterion in the design, procurement, and use of computer systems [...] to get the highest performance for a given cost."
- "The types of applications of computers are so numerous that it is not possible to have a standard measure of performance [...] for all cases."
- "The first step in performance evaluation is to select the right measures of performance, the right measurement environments, and the right techniques."

1.2 CERN

The European Organization for Nuclear Research, is one of the world's largest and most respected centers for scientific research. Located near Geneva, Switzerland, CERN is home to the Large Hadron Collider (LHC), the world's largest particle accelerator, and several other cutting-edge experiments in particle physics.
Scientists at CERN conduct research on the fundamental particles and forces that make up our universe, using particle accelerators to study the behavior of these particles under extreme conditions. They investigate questions such as how particles acquire mass, the nature of dark matter, and the possibility of extra dimensions.
CERN is also a center for international collaboration, with thousands of scientists from around the world working together on experiments and sharing knowledge and expertise. The organization is committed to promoting scientific education and outreach, with programs designed to inspire the next generation of scientists and increase public understanding of the role of science in our world.
The CERN main missions are to

- Conduct research in physics and search for new discoveries.
- Operate advanced particle accelerators and develop new technologies.

[1] That would take a long time to explain. If you're interested, check out the Mr. Giordano's great presentation on https://indico.in2p3.fr/event/20020/contributions/76823/attachments/56495/74936/LCG-France-12-12-2019-giordano.pdf

- Encourage international collaboration in science and knowledge exchange.
- Train students and researchers from around the world.
- Educate and inspire the public about science.
- Use science and technology for peaceful purposes and sustainable development.

1.2.1 The Large Hadron Collider

The Large Hadron Collider (LHC) is the largest and most powerful particle accelerator ever built. It is an engineering marvel, with a length of 27 kilometers (16.8 miles), making it the largest machine ever built by humankind.

Construction of the LHC began in 1998, and it took nearly a decade to complete. The first proton-proton collisions were observed in 2010, and since then, the LHC has been at the forefront of particle physics research. The LHC is used to accelerate beams of protons or heavy ions to nearly the speed of light before smashing them into each other at four points along the ring. These collisions produce a shower of particles that can be detected by massive detectors like ATLAS, CMS, LHCb, and ALICE (see Fig. 1.1 on page 17).

The LHC operates at an energy of 13 TeV (tera-electronvolts) for proton-proton collisions and up to 5.5 TeV per nucleon for heavy-ion collisions, making it the highest energy particle accelerator ever built. To achieve these energies, the LHC uses over 1,600 superconducting magnets, each weighing over 27 tonnes, to guide the particle beams around the ring. These magnets are cooled to a temperature of -271.3°C (-456.3°F, or 1.9K, which is colder than the 2.7K of outer space) using over 120 tonnes of liquid helium, making the LHC the largest cryogenic installation in the world.

In addition to its impressive size and energy, the LHC also generates an enormous amount of data. When two beams collide, the detectors at the collision points produce terabytes of data every second, requiring a massive computing infrastructure to store and analyze the data. The LHC Computing Grid, a worldwide network of computing centers, provides the computing power needed to process the data, with over 170 computing centers in 42 countries.

Since its inception, the LHC has made numerous groundbreaking discoveries, including the discovery of the Higgs boson in 2012, a particle that gives mass to other particles. The discovery confirmed the existence of the Higgs field, a fundamental component of the Standard Model of particle physics. The LHC has also been used to search for evidence of supersymmetry, extra dimensions, and other theories that go beyond the Standard Model.

In conclusion, the LHC is an incredible feat of engineering and a testament to the ingenuity and dedication of scientists and engineers around the world. Its impressive size, energy, and capabilities have revolutionized particle physics research and have led to groundbreaking discoveries that have expanded our understanding of the universe.

Alice

ALICE (A Large Ion Collider Experiment) studies the properties of quark-gluon plasma, which is a state of matter that is believed to have existed shortly after the Big Bang. By colliding lead ions together at high energies, ALICE can recreate the conditions that existed in the early universe and study the properties of this plasma.

Atlas

ATLAS (A Toroidal LHC ApparatuS) is a general-purpose detector designed to search for a wide range of new particles and phenomena beyond the Standard Model of particle physics. It has a complex and highly segmented design that enables it to detect and measure the properties of particles produced in high-energy collisions, including the Higgs boson, top quarks, and hypothetical particles like supersymmetric particles and extra dimensions.

CMS

CMS (Compact Muon Solenoid) is also a general-purpose detector designed to explore a wide range of physics phenomena. However, it has a more compact and streamlined design compared to ATLAS, which allows it to detect particles more efficiently in certain regions of the detector. CMS has made important contributions to the discovery of the Higgs boson and searches for new particles and forces, and it also studies the behavior of particles containing the beauty quark to investigate the dominance of matter over antimatter in the universe.

LHCb

LHCb (Large Hadron Collider beauty) is designed to study the properties of particles that contain the "beauty" or "bottom" quark. By studying the behavior of these particles, LHCb can search for evidence of new physics beyond the Standard Model, such as the existence of new particles or interactions.
Overall, each of these experiments is designed to complement the others and provide a more complete picture of the fundamental particles and forces that make up our universe.

Computing

During the design phase of the computing system for LHC data analysis in 1999, it became apparent that the necessary computing capacity surpassed the funding capacity available at CERN. However, several laboratories and universities that were collaborating on the LHC project had access to regional or national computing resources. This led to the question of whether these facilities could be combined to form a single LHC computing service. The advancements in wide-area networking,

increased capacity and bandwidth, and lower costs made this integration possible. Subsequently, this led to the development of the Worldwide LHC Computing Grid.

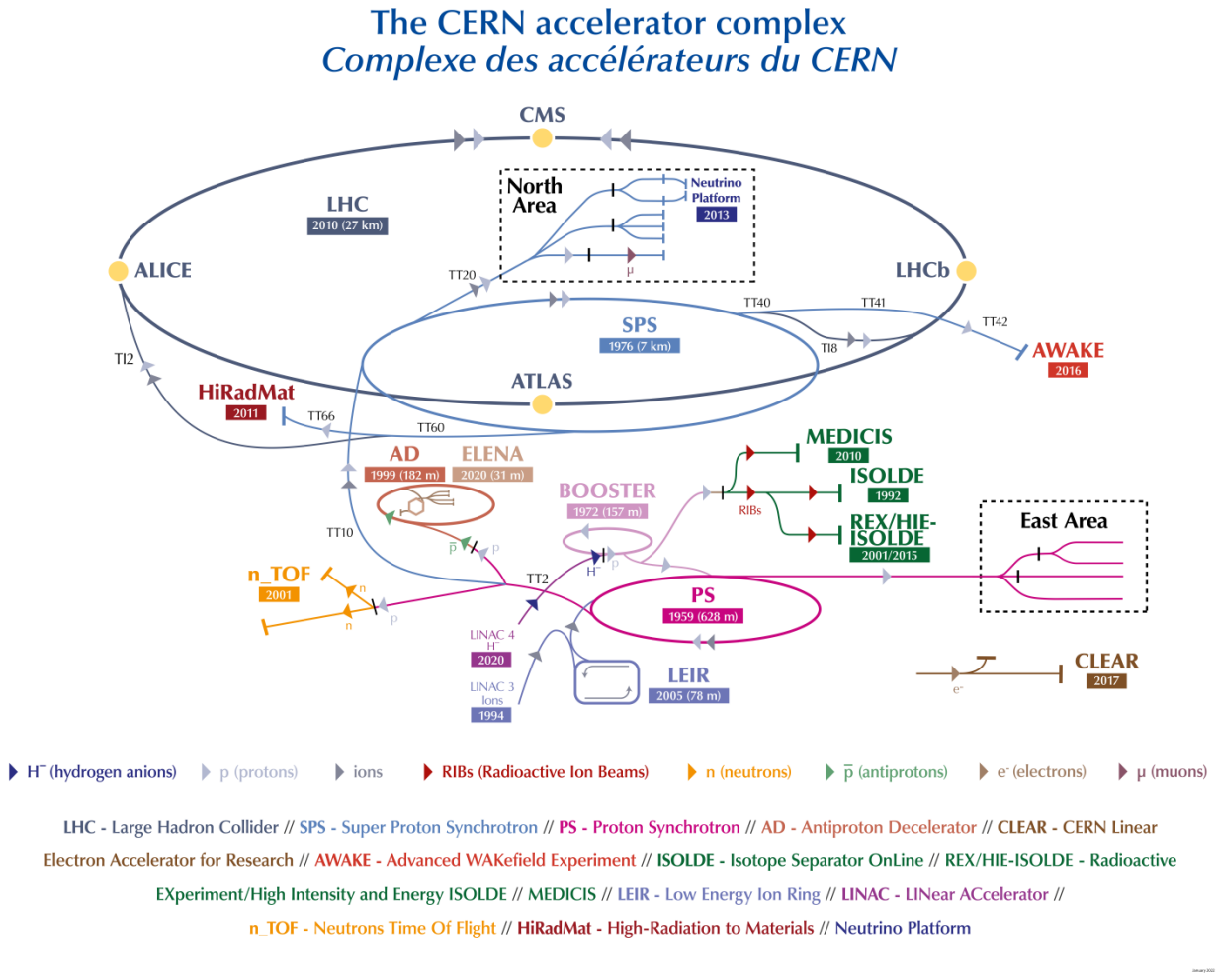

Figure 1.1: The CERN's accelerator complex [4]

1.3 Datacentre

The CERN Data Centre is the heart of CERN's entire scientific, administrative, and computing infrastructure. All services, including email, scientific data management and videoconferencing use equipment based here.

More than 450 000 processor cores and 10 000 servers run 24/7. Over 90% of the resources for computing in the Data Centre are provided through a private cloud based on OpenStack, an open-

source project to deliver a massively scalable cloud operating system.[2]
The CERN Data Centre currently holds over 300 petabytes (PB) of data, with plans to expand this capacity to several exabytes (EB) in the coming years. In 2023, CERN is planning to open newly built datacentre building in the Prevessin, France. The new datacentre is born to handle all the new data from planned High-Luminosity LHC (HL-LHC) which comes online in 2029.
It's computing and storage infrastructure is essential to the success of CERN's scientific program, enabling researchers from around the world to analyze and process the vast amounts of data generated by experiments at the LHC. With the continued growth of data volumes and the increasing complexity of analyses, the data centre's capabilities will become even more critical in the years to come.

1.4 Grid computing

Grid computing is a distributed computing model that allows multiple organizations to share computing resources over a network, such as the internet. This approach enables large-scale computing tasks to be performed more efficiently and cost-effectively than would be possible with a single centralized system. Grid computing is used in many fields, such as medical research, computer graphics, weather forecasting. But for the purpose of this thesis, we'll focus on its usage in the scientific research. Grid computing provides several advantages for scientific research, including

- scalability: Grid computing allows researchers to scale their computing resources to meet their needs, without having to invest in expensive hardware or infrastructure.
- collaboration: Grid computing enables researchers to collaborate across institutions, countries, and even continents, sharing data and computing resources to achieve common research goals.
- efficiency: By sharing resources and avoiding duplication of effort, grid computing can reduce costs and improve the efficiency of scientific research.
- accessibility: Grid computing makes advanced computational resources available to researchers who might not have access to such resources otherwise, democratizing scientific research and allowing more researchers to participate in cutting-edge projects.

To understand why a grid infrastructure was chosen, we must consider several factors that had an important impact on the design of the LHC computing environment. Primarily, the volume of data generated by the experiments is estimated at some 15 PB per year. These data are generated at a significant rate: some 300–400 megabyte (MB) per second by the two largest experiments, ATLAS and CMS. During the period when the LHC accelerates heavy ions, this data rate increases to around 2 gigabyte (GB) per second for ALICE, the dedicated heavy ion experiment. The fourth experiment, LHCb, generates data at the lower rate of <100 MB/s. Thus, during most of the

[2]CERN. The CERN Data Centre [online]. 2023. [visited on 2023-04-11]. Available from: https://home.web.cern.ch/science/computing/data-centre

LHC running, the total data rate is around 1 GB/s and this rises to close to 3 GB/s during heavy ion running. These data are archived at CERN and a second copy is distributed between 11 regional (Tier 1) centers in real time, together with an equal amount of data resulting from their initial processing. Thus, the distribution system must be capable of supporting these rates continuously.[5]

1.5 WLCG

The mission of the Worldwide LHC Computing Grid (WLCG) is to provide global computing resources for the storage, distribution and analysis of the data generated by the LHC. WLCG combines about 1.4 million computer cores and 1.5 exabytes of storage from over 170 sites in 42 countries. This massive distributed computing infrastructure provides more than 12 000 physicists around the world with near real-time access to LHC data, and the power to process it.
It runs over 2 million tasks per day and, at the end of the LHC's LS2 (Long Shutdown phase), global transfer rates exceeded 260 GB/s. These numbers will increase as time goes on and as computing resources and new technologies become ever more available across the world. CERN provides about 20% of the resources of WLCG.[3]
The WLCG operates using a hierarchical model (see Fig. A.1 on page 62), with Tier-0 sites at CERN receiving raw data from the LHC and processing it into "derived data," which is then distributed to Tier-1 sites located around the world. These Tier-1 sites store and distribute the data to Tier-2 sites, which provide computing resources for data processing and analysis by individual researchers and teams. The main responsibilities of the different tiers of the WLCG computing model are as follows[6]:

- Tier0 (CERN): safe keeping of RAW data (first copy); first pass reconstruction, distribution of RAW data and reconstruction output (Event Summary Data or ESD) to Tier1; reprocessing of data during LHC down-times;
- Tier1: safe keeping of a proportional share of RAW and reconstructed data; large scale reprocessing and safe keeping of corresponding output; distribution of data products to Tier2s and safe keeping of a share of simulated data produced at these Tier2s;
- Tier2: Handling analysis requirements and proportional share of simulated event production and reconstruction.

[3]CERN. The Worldwide LHC Computing Grid (WLCG) [online]. 2023. [visited on 2023-04-11]. Available from: https://home.cern/science/computing/grid

Chapter 2

Background

2.1 WLCG history

In 2002, the WLCG was established as a collaboration between CERN and dozens of other computer centers around the world. The goal of the WLCG was to provide a distributed computing infrastructure that could scale to meet the computing and data storage needs of the LHC experiments. Over the years, the WLCG has evolved to meet the changing needs of the LHC experiments. In the early years, the focus was on building a distributed computing infrastructure that could process and store the vast amounts of data produced by the LHC experiments. This required the development of new software tools and technologies to manage the distributed computing resources of the WLCG. As the LHC experiments continued and the data processing and storage needs grew, the WLCG evolved to incorporate new computing technologies, such as cloud computing and grid computing. The WLCG also expanded its membership to include new computer centers around the world, and developed new mechanisms for managing and allocating computing resources, such as the concept of pledges.
Today, the WLCG is a global collaboration of dozens of computer centers and thousands of researchers, working together to manage and analyze the data produced by the LHC experiments. Even you can help by donating you idle computer time, check out the LHC@home project. The WLCG continues to evolve and adapt to meet the ongoing challenges presented by the LHC experiments, and to support the scientific goals of the global particle physics community.

2.2 Pledge

One of the mechanisms used by the WLCG to manage and allocate computing resources is the concept of pledges. Pledges are commitments made by participating centers to provide a certain amount of computing resources to the WLCG for a specific period of time, usually a year. These commitments are based on the capabilities and capacity of the center, and are intended to ensure

that the WLCG has access to sufficient resources to support its data processing and analysis needs. Pledges are typically defined in terms of three main components: CPU, storage, and network. Pledging is an important aspect of resource management in the WLCG because it allows experiments to plan and allocate resources more effectively. By knowing in advance the amount of resources that will be available, experiments can better estimate their computing needs and adjust their workflows accordingly. Pledges are made by individual computing centers, such as universities or research institutions, and are typically based on the center's available resources and capacity. Once a pledge is made, the resource provider is expected to honor the commitment and provide the pledged resources to the grid.

Pledges are reviewed regularly to ensure that they continue to accurately reflect the capabilities and resources of each participating center. Overall, pledges are an important mechanism for coordinating and managing the distributed computing resources of the WLCG, and for ensuring that the collaboration has access to the resources it needs to support its scientific goals.

Overall, pledges are a key component of the WLCG's resource management system and help ensure that computing resources are available to support the analysis of data from the Large Hadron Collider experiments.

Pledges in 2022

To have some idea how does it work, let's see CPU pledges from CERN and Czech sites (see Tab. B.1), and then CPU pledges and requirements (see Tab. B.2) from the LHC experiments. The pledge is a promise from a site, how much computing power can they provide. It's defined as amount of CPUs in HS06 units. On the other side, requirements number is the amount of computing power being used by the site's users. As we can see in the table B.2, CMS experiment could in 2022 provide 540000 HS06, and it consumed a total of 540000 HS06. This means that CMS has fulfilled 100% of it's pledge when it comes to Tier-0 pledges. (Although it does not neccessarily mean that all the computations stayed on-site.) But in Tier-1 pledges, CMS pledged 17% more units than required, which means it's resources probably contributed to other sites which required more than pledged (for example LHCb).

2.3 Accounting

Grid accounting serves the purpose of monitoring the usage of computing resources by individual users, experiments, and sites within the WLCG ecosystem. To achieve this, the WLCG employs a sophisticated accounting system that systematically collects crucial information such as CPU time consumption, data processing, and storage utilization. By aggregating this information and processing it, reports are generated to provide insights into the pattern of resource usage and

utilization across the entire grid network.
In order to support the scientific experiments conducted at CERN, a specific amount of computing resources must be provided each year, sourced from funding agencies and various sites. To assess the efficiency of these resources, resource review boards analyze the actual resource utilization compared to the initial resource request. To maximize performance and cost-effectiveness, the procurement team purchases new hardware based on careful evaluations of efficiency and cost-effectiveness. These measures are essential in ensuring efficient resource management, avoiding unnecessary waste of both resources and funding.
Current role of WLCG accounting team at CERN

- collects data from batch and HPC services in order to provide information for resource usage by experiments, departments and services
- collects data for all resources known to OpenStack
- compares the usage of the resources versus the promised quota to the experiments
- creates daily reports for APEL
- provides API for mapping users to their charge group
- visualisation of all these information on dashboards (Grafana) [1]

As an example what accounting is good for, let's see Fig 2.1 on page 23. We can see, that all VO's CPU usage together (colorful columns) resulted in using more than they asked for (blue line with white dots). For the following years, each VO needs to reconsider asking for more resources and try to guess their CPU usage better accordingly to their planned changes.

2.4 Benchmarking

CPU benchmarking is a method used to evaluate the performance of a computer's CPU. This process involves running a series of standardized tests that simulate real-world workloads to determine the CPU's ability to handle different types of tasks.
There are several types of benchmarks, but generally we can divide them into 2 groups: Synthetic and real-world. When looking for a quick, general comparison between CPUs, synthetic benchmark should be a good choice. A range of tasks such as 3D rendering, file compression, web browsing, and floating-point calculations are simulated in synthetic tests. The performance of the CPU is measured for each task, and the results are then combined and evaluated to obtain a single score.
However, there are also real-world, so-called application-based benchmarks that focus on measuring the performance of CPUs in real-world scenarios such as video editing, gaming, or scientific simulations. These benchmarks can provide a more accurate representation of a CPU's performance under real-world conditions.
To benchmark a CPU in CERN, the HEPSPEC benchmark derived from synthetic SPEC CPU 2006

[1] CERN. Accounting Ops - Introduction [online]. 2023. [visited on 2023-04-24]. Available from: https://accountingops.web.cern.ch/

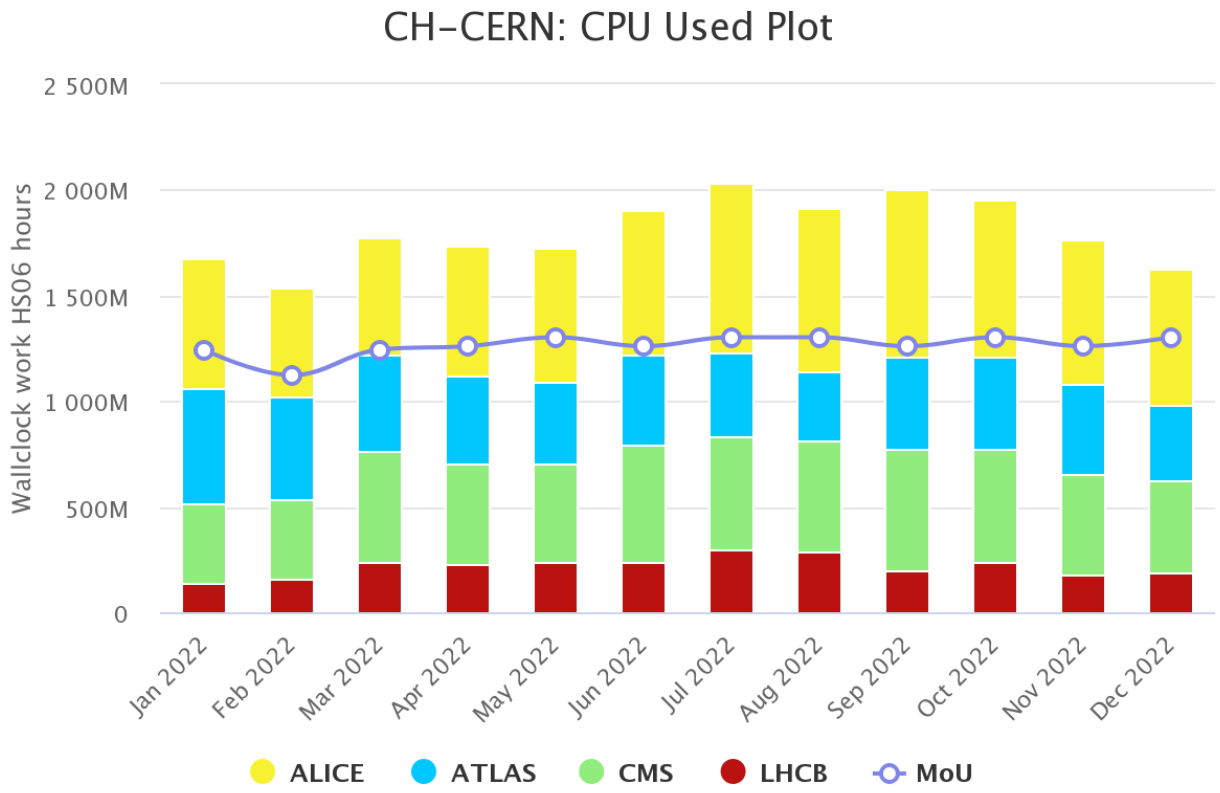

Figure 2.1: CPU used vs. requested in 2022, in wallclock HS06 hours [7]

benchmark is used in 2022. And I'm here to help with the transition to a new application-based benchmark called HEPScore 2023.

HEPSPEC 2006

HEP-SPEC2006 (HS06) is a suite of 7 C++ applications - subset of SPEC CPU 2006 benchmark. The Standard Performance Evaluation Corporation (SPEC), founded in 1988, is an American organization aims to establish, maintain and endorse a standardized set of performance benchmarks for computers. [...] The SPEC benchmarks are written in a platform neutral programming language, and the interested parties may compile the code using whatever compiler they prefer for their platform, but may not change the code. In order to use a benchmark, a license has to be purchased from SPEC. The costs vary from test to test with a typical range from several hundred to several thousand dollars. [8]

HEPscore 2023

HEPscore is the new HEP-specific CPU benchmark for WLCG. The workloads are created to suit CERN's needs. More about it in Analysis chapter, see 3.2.2 HEPscore (on page 30) and it's Workloads (section 3.2.1 on page 29).

2.5 Technologies

2.5.1 Containerization

A container can be considered a tiny and isolated virtual environment, which includes a set of specific dependencies needed to run a specific application. This makes it easier to run the software in same conditions on different environments to get appropriate results. The HEP workloads for CERN were containerized and fully validated in the spring of 2021. [9]

Docker and Apptainer (Singularity)

Several containerization platforms exist for the purpose of managing containers. In CERN we use Docker and Apptainer (previously known as Singularity), where Docker is designed for creating and deploying applications in a variety of environments, whereas Singularity is primarily intended for scientific computing and HPC workloads. That's why Singularity is more commonly used on WLCG.
When it comes to security, Docker is primarily intended to run untrusted third-party applications in a secure environment, so it includes features like isolated filesystems and network interfaces[10]. Singularity on the other hand is designed to run trusted applications in a shared environment, so it has less security overhead[11].

2.5.2 Batch systems

For any High-Performance Computing (HPC) system, it is essential to have a fair allocation of computational resources among users. This can be achieved through the use of batch queuing systems, which are responsible for scheduling and prioritizing jobs in a shared environment. Batch systems are equipped with a job scheduler that allocates resources such as CPUs to jobs and prioritizes them based on user-defined parameters. In addition, these systems have one or more queues to which jobs are submitted, and each queue can be configured to handle specific types of jobs, such as serial or parallel jobs, long or short jobs, or those with high memory requirements. Job schedulers normally operate only with dedicated machines. Often termed worker nodes, these dedicated machines are typically owned by one group and dedicated to the sole purpose of running compute jobs. [12] There are several batch systems that are used for WLCG purposes. For this project we'll work only with HTCondor, but I'll describe some others to compare.

Torque

Torque is a workload manager and job scheduler that allows users to submit and manage jobs on a cluster of computers. It is often used in HPC environments, where large-scale simulations or data processing tasks are performed. Torque provides a framework for submitting, tracking, and managing jobs across a distributed network of computers, and includes features such as job prioritization, scheduling, and resource allocation. It is often used in combination with other software, such as Maui or Moab, to provide HPC environment to manage resources and schedule jobs on a large scale. In contrast to other mentioned schedulers, Torque with Maui/Moab are more standardized and may be easier to set up and use, but may not provide the same level of customization.

SLURM

SLURM (Simple Linux Utility for Resource Management) is an open-source job scheduler and resource management system used in HPC environments. It provides a framework for scheduling and managing the allocation of computing resources, such as CPUs, memory, and GPUs, across a cluster of interconnected nodes. It is designed to be highly scalable, supporting large-scale systems with tens of thousands of nodes and millions of tasks. It offers a variety of scheduling policies and advanced features, including backfill scheduling[13], job prioritization, job arrays, job dependencies, and checkpoint/restart capabilities.

LSF

IBM Spectrum LSF is a powerful workload management platform for demanding, distributed HPC environments. It provides a comprehensive set of intelligent, policy-driven scheduling features that enables full utilization of your compute infrastructure resources and ensure optimal application performance.[14] This technology was used at Batch accounting before CERN migrated to HTCondor.

HTCondor

HTCondor (High-Throughput Computing Condor) is a specialized workload management system designed to manage large amounts of computational tasks across a pool of distributed computing resources such as clusters, grids, and clouds. It is a free and open-source software developed by the Center for High Throughput Computing at the University of Wisconsin-Madison.
It is another job scheduler and resource manager commonly used in scientific computing environments. HTCondor operates differently from Torque in that it uses a "matchmaking" system to match jobs to available resources based on various criteria such as job requirements, resource availability, and job priorities.
HTCondor can schedule jobs on dedicated machines, but unlike traditional batch systems, HTCondor is also designed to run jobs on machines shared and used by other systems or people. By

running on these shared resources, scheudler can effectively harness all machines throughout a campus. This is important because often an organization has more latent, idle computers than any single department or group otherwise has access to.[12]
You'll learn more about HTCondor in the CERN environment, in chapter 3.3.2 HTCondor cluster on page 37.

2.5.3 S3 Ceph storage

S3 is a storage API for various object storages. Amazon, Ceph, Google Cloud, Azure... are supporting these object storages, and the objects (files/data) within are stored in a bucket (directory/placeholder). To access S3 storage, an `access_key_id` and `secret_access_key` is needed, usually users can just generate them through the service's frontend application in their browser.
Red Hat Ceph is a kind of open-source software that has been developed to enable the easy storage of files, blocks, and other data types in a single, integrated system. This software uses its own file system, called CephFS, and can handle problems without any manual intervention, making it a very powerful storage solution. Moreover, it has the ability to detect and fix failures on its own, while also continually striving to minimize the cost of system administration.
Red Hat Ceph Object Gateway supports a RESTful API that is compatible with the basic data access model of the Amazon S3 API.[15] Thanks to this, we can install `s3cmd` tool on our Linux distro (in my case Ubuntu 22.04.2 LTS) and manage the storage easily with familiar arguments like "ls" or "get". `S3cmd` is written in Python, and it's an open source project available under GNU Public License v2 (GPLv2) and is free for both commercial and private use.[16]

2.5.4 BOINC

BOINC is a platform for distributed high throughput computing, i.e. large numbers of independent compute-intensive jobs, where there performance goal is high rate of job completion rather than low turnaround time of individual jobs. It also offers low-level mechanisms for distributed data storage. BOINC has a client/server architecture: the server distributes jobs, while the client runs on worker nodes, which execute jobs.
Here at CERN, we use BOINC in volunteer computing, where the worker nodes are consumer devices (desktop and laptop computers, tablets, smartphones) volunteered by their owners. BOINC addresses the various challenges inherent in this environment (heterogenity, host churn and unreliability, scale, security, and so on). Except CERN's LHC@home, there are other volunteer-computing projects using this platform, like Einstein@home or World Community Grid.
BOINC can run all existing HTC applications, including those that use GPUs and/or multiple CPU cores. It can use virtual machines to run existing Linux applications on Windows and Mac worker nodes. BOINC is distributed under the LGPL v3 open-source license. It can be used for

any purpose (academic, commercial, or private) and can be used with applications that are not open-source.[17]

2.5.5 Puppet

Puppet is a widely adopted configuration management tool in IT infrastructure automation. It enables system administrators to define and enforce the desired state of computer systems using a declarative language. By specifying configuration settings, packages, services, and files, Puppet automates the implementation and maintenance of these configurations across multiple systems. This centralized approach simplifies the management of complex environments and reduces manual effort by automating repetitive tasks. With its scalability and efficiency, Puppet ensures consistency and facilitates streamlined management of a large number of servers, making it a valuable asset in modern IT operations.
I particularly valued the utilization of Git for storing configurations. Examining previous commits and branches provided valuable insights into the evolution of managing internal tools, ultimately enhancing my understanding of these tools and their functionality.

2.5.6 APEL and EGI

APEL (Accounting Processor for Event Logs) is a software system used for collecting and processing accounting information in high-performance computing environments. It is specifically designed to handle accounting data generated by grid and cloud computing infrastructures. APEL collects data from various sources, including batch systems like HTCondor, and processes it to generate accurate and consistent usage records. Here at CERN, APEL plays a vital role in collecting accounting data related to CPU usage, wall time, and other resource consumption metrics from HTCondor-based computing clusters in WLCG.
EGI refers to the European Grid Infrastructure, which is a distributed computing infrastructure that connects and coordinates computing resources across different research institutions and countries. It ensures that accounting data from diverse sources, including APEL, are effectively gathered, combined, and accessible for analysis and reporting purposes. By serving as a framework, EGI promotes consistency in accounting practices, ensuring that resource usage is tracked and reported uniformly across the entire EGI infrastructure.

Chapter 3

Analysis

In this chapter, we'll see how the benchmarking data are created, used, and reported to third party services like EGI. First of all, let's make clear some terms when it comes to the WLCG accounting:

- job is a discrete unit of work, an executable with some optional arguments
- HEP workload is a single workload (job, executable) created by specialists from each experiment at CERN. It's purpose is to juice the CPU out so we can make some omelettes in the server room.
- `hep-workloads` is a repository, that runs a single HEP workload
- `hep-score` benchmark runs several HEP workloads using hep-workloads repostory, and averages/combines individual scores to give a single benchmark number for the machine
- `hep-benchmark-suite` can run several benchmarks (HS23 - HEPSCORE, HS06 - HEPSPEC, SI2K - SPEC INT 2000, ...) to collect their score, and is used by a procurement team when benchmarking a new piece of hardware
- I refer to a `hep-score` as a piece of code, while HEPSCORE, hepscore, hs23, HS23 stands for the new type of benchmark in general

3.1 Batch Accounting

Batch accounting is about collecting, processing and publishing accounting data from the HTCondor batch system. It's the main topic of this thesis - after understanding how does it work, the main task is to update of the batch accounting workflow by adding a new metric to compare CPUs. Batch Accounting is a part of IT-CD-CC section at CERN, and it's related to terms like computing centre, Tier-0, WLCG, Batch systems (HTCondor), benchmarking, resource coordination, and support.

3.2 The process of benchmarking a processor

Here at CERN, benchmarking is done by the Procurement team right before a new processor is going to be a new part of the production. The team uses an internal tool called Benchmark Suite, which can use several benchmarks to get a benchmark number for the CPU.
In this section, we'll see how to benchmark HEPScore23 manually, without the Benchmark Suite implementation, just to see how it's done, and what score will we get. First by using HEPSCORE locally, then on HTCondor to see how HTCondor works from the user's point of view. To start using hep-score, we need to select a workload(s), so let's have a look at them.

3.2.1 Workloads

In general, a workload is a measure of the amount of computing resources (such as CPU or memory) required to perform a specific task or set of tasks. In the context of benchmarking, workloads are typically used to evaluate the performance of computer systems or components.
For CERN experiments, the workloads can be quite complex and demanding. High Energy Physics (HEP) experiments generate vast amounts of data, and processing this data requires significant computing resources. The workloads for CERN experiments are typically characterized by large amounts of data that need to be processed with complex algorithms.
One example of a HEP workload is the simulation of particle collisions using Monte Carlo techniques. This involves generating large numbers of simulated events and analyzing them to test various theories and hypotheses. Another example is the reconstruction of data from the detectors, which involves processing enormous amounts of raw data to extract useful information about particle interactions.
To evaluate the performance of computing resources for these workloads, benchmarks such as SPEC and HEP-SPEC are used. These benchmarks simulate the types of workloads that are common and provide a way to compare the performance of different systems. By measuring the performance of these workloads, researchers can determine which computing resources are best suited for their needs and make more informed decisions about how to allocate computing resources for their experiments.
I had the privilege of participating in a few meetings where we looked into the complex matter of selecting workloads for the HEPScore benchmarking process. In 2021-2022, experts created ∼30 standalone containers of HEP workloads. With a total of 11 candidates submitted by various experiments, the goal was to choose a subset that would be feasible to run in a less time-consuming way (all 11 of them run for ∼15 hours). After much analysis, in February 2023 it was determined that a handful of workloads did not display a high correlation[1] with each other and thus were selected for inclusion. Although different weighting methods were considered, ultimately it was concluded that weighting the workloads equally was the simplest and most comparable option. As it

[1]See Mr. Sullivan's presentation from HEPscore Workshop 2022 on HEPscore candidates for more details: https://indico.cern.ch/event/1170924/

stands, we anticipate that workloads `LHCb_gen_sim`, `ATLAS_gen_sherpa`, `CMS_gen_sim`, `CMS_reco`, `ATLAS_reco`, and `Belle2_gen_sim_reco` (see Table B.3 on page 67 for run times) will be the chosen few to represent the HEPScore benchmark, for now defined in hepscore_beta.yaml, default configuration for running hep-score. To better understand some of the workload names, let's describe just two of them - to imagine why do the physicists need so much resources to run their jobs:

- SIM stands for simulation
  Simulation programs are complex code systems that simulate the collisions between elementary particles at the foreseen experimental conditions, and then "transport" the particles generated during the reaction through the matter of the detector. [...] Simulation is mostly a "CPU bound" activity, where the input data are limited to small configuration files and the output is similar to the output of the detector. [18]
- RECO stands for reconstruction
  The output of a detector is a set of electric signals generated by the passage of particles through matter. These are the so-called "raw data" or simply "raw". The "reconstruction" process aims at processing these signals to determine the features of the particles generating them, i.e. their mass, charge, energy and direction. [..] Depending on the complexity of the algorithms deployed, reconstruction can be an "I/O bound" or a "CPU bound" activity, where the input are the raw data and the output the ESD files. [18]

3.2.2 `hep-score`

`hep-score` is a benchmarking software, created by CERN engineers to benchmark machines on WLCG, able to run several workloads (created by CERN experts from each experiment). Usually it runs the workload 3 times to get the geometric mean of results, but all sort of things (including the number of runs) can be edited in .yaml file, where the configuration is defined.
The most important of `hep-score`, same as any other CPU benchmarking tool, is the resulting relative number to a reference machine. This number indicates, how much better/worse the benchmarked CPU is in running the workloads than the reference machine.

Configuration

The whole yaml file is a "hepscore_benchmark" dictionary, and contains two other dictionaries: "benchmarks" and "settings", both required. In the "benchmarks" section we define all the workloads we want to run, in our case it's just `atlas-gen-bmk`, because it's the shortest workload I could find that time, but more of them can be defined (see 3.2.1 Workloads section on page 29). Referential scores, or `ref_scores` is a list of sub-scores to collect from the benchmark container output JSON, with reference scores from the specified "reference_machine". Each sub-score is divided by its reference score, and the geometric mean is taken from the results to compute a final score for the benchmark container.

After we define the name of results file and `ref_scores`, we may update the weight, but it has a very small effect on the result, and does not make sense updating it on just one workload. Specific workloads may have arguments, in our case the arguments are needed to define a number of events and threads. (This workload can be ran in parallel, so number of threads would decrease the time run significantly.)
In the "settings" the most important thing to define is "reference_machine". In our case the reference machine is Intel Xeon E5-2630, so the resulting number is relative to this machine with 8 cores and 16 threads, in 64 bit version. Another things to define are the method (agreed to be geometric mean) and repetitions (that's what we're doing the geometric mean from). The HEPscore23 benchmarking software can run both Docker or Singularity containers, but most of workloads are prepared only in Singularity. (The new name for Singularity is officially Apptainer, and I may refer to it by both names depending on the context.)

```yaml
hepscore_benchmark:
  benchmarks:
    atlas-gen-bmk:
      results_file: atlas-gen_summary.json
      ref_scores:
        gen: 384
      weight: 1.0
      version: v2.1
      args:
        threads: 1
        events: 5
  settings:
    name: TestBenchmark
    reference_machine: "CPU Intel(R) Xeon(R) CPU E5-2630 v3 @ 2.40GHz"
    registry: docker://gitlab-registry.cern.ch/hep-benchmarks/hep-workloads
    method: geometric_mean
    repetitions: 3
    scaling: 355
    container_exec: singularity
```

Listing 3.1: hep.yaml - My example of configuration file to run hep-score

Running locally

After defining the benchmark(s) in the configuration yaml, I can test our CPU with it. I chose the example configuration with `atlas-gen-bmk`, because it takes ∼10 minutes, so for 3 runs it takes

about half an hour. To get the hep-score program, we just need to pip install it from a git repository into a python virtual environment. And to get the desired result, we simply need to run it. In our case, we'll use `-f` option to define our own configuration file (as in Fig. 3.1 on page 31), and `-m` to specify our container platform as Singularity (even though Singularity is default, it's nice to be aware of that we're using it). This can be summarized as script 3.2 on page 32.

```
#!/bin/bash
source /eos/user/l/lzurkova/myvenv/bin/activate
mkdir testdir
pip install --user git+https://gitlab.cern.ch/hep-benchmarks/hep-score.git
hep-score -m singularity -f hep.yaml testdir/
```

Listing 3.2: Bash script to run hep-score

For the next half an hour we can see how the jobs are executed. Check Listing 3.3 on page 32 to see the output of hep-score run. After the run we get the geometric mean of the 3 runs, which is in this case `46.221`. And except the console output, we got more detailed logs and results in the `testdir` directory, which we created for this purpose. A new directory called `HEPscore_29Mar2023_130241` is created inside, containing subdirectory for each benchmark (only `atlas-gen-bmk` in our case), and two files sharing the name of the benchmark, as defined in the configuration file (in our case (`TestBenchmark.log` and `TestBenchmark.json`). The benchmark directory contains more detailed information about each run with severeal logs and jsons, but for our purpose they're unnecessarily detailed. The `TestBenchmark.log` file contains more detailed information about run, and the `TestBenchmark.json` file contains the results of each run, summary, and environment configuration - see an example in Listing C.1 on page 69). This file was used for an analysis of different configurations by the CERN engineers to select carefully the right set of workloads to get an effective score. After we've benchmarked CPU on my laptop and saw how it works, we're ready to benchmark a worker node from the Datacentre.

```
2023-03-29 13:02:41 hepscore [INFO] TestBenchmark Benchmark
2023-03-29 13:02:41 hepscore [INFO] Config Hash:
    cc648ef17e4bed772c3db48ddc86e3783a57342f4fba59eb672c358256fedd2d
2023-03-29 13:02:41 hepscore [INFO] HEPscore version: 1.5.0
2023-03-29 13:02:41 hepscore [INFO] System:         Linux lzurkova-x1 5.19.0-38-
    generic #39~22.04.1-Ubuntu SMP PREEMPT_DYNAMIC Fri Mar 17 21:16:15 UTC 2
    x86_64
2023-03-29 13:02:41 hepscore [INFO] Container Execution: singularity
2023-03-29 13:02:41 hepscore [INFO] Implementation: singularity
2023-03-29 13:02:41 hepscore [INFO] Registry:       docker://gitlab-registry.cern.
    ch/hep-benchmarks/hep-workloads
```

```
2023-03-29 13:02:41 hepscore [INFO] Output:        /home/lzurkova/Code/hep-test/
    testdir/HEPscore_29Mar2023_130241
2023-03-29 13:02:41 hepscore [INFO] Date:          Wed Mar 29 13:02:41 2023

2023-03-29 13:02:41 hepscore [INFO] Executing 3 runs of atlas-gen-bmk [v2.1]
2023-03-29 13:02:41 hepscore [INFO] Starting run0
2023-03-29 13:15:34 hepscore [INFO] Starting run1
2023-03-29 13:26:02 hepscore [INFO] Starting run2
2023-03-29 13:36:30 hepscore [INFO]
2023-03-29 13:36:30 hepscore [INFO] Final result: 46.221
```

Listing 3.3: Console output after running hep-score locally

Running on HTCondor

The output was just the same, except the scores. Where my Thinkpad with Intel(R) Core(TM) i5-10310U CPU @ 1.70GHz got score 46.221 for the atlas event generations (see the listing C.1), the CERN's node Intel(R) Xeon(R) CPU E5-2650 v4 @ 2.20GHz got 206.5529. I created a table B.4 on page 68 to compare those CPUs and the reference machine on the atlas-gen-bmk workload, and the set of 7 chosen workloads mentioned in the 3.2.1 Workloads section on page 29. The table is not occupied by scores everywhere, because to run the whole set of workloads on my laptop, it would take quite amount of time. And reference machine has just the score for hepscore-beta, because I do not posses this type of CPU to test it for only atlas-gen-bmk workload. But what can we read from the table is that my Thinkpad has circa 4 times worse CPU than the WLCG node, and the node is $\sim$ 2 times better than the reference machine.

To get the number of the CPU in the WLCG, I needed to submit the hep-score run as a job, inside the Singularity container. To do that, I need to know something about environment (see following section 3.3 Environment on page 35), but to keep "running hepscore" parts together in this text, I'll describe it here.

First I define submission file for the batch system, where I set the details necessary for HTCondor to schedule the job's run, you can see an example in the listing 3.4 on page 35. My job is an easy one, I don't need any in-run arguments, or run several jobs in series, or any other kind of special treatment. That's why I define my universe in which the job runs as `vanilla` - just a fancier way to say *default*. Then I need to tell what do I want to run, usually by mentioning the bash script which actually runs the job. After this there are log files defined, and we can even specify `requirements` to set the node specifications we're expecting. I defined the machine name, but others requirements can be specified like operating system or how many cores are we looking for, so HTCondor can match it as we desire. Then we need to import our own hep-score configuration file, which I named my_hep.yaml, and define that we want to tranfer the output file TestBenchmark.json

back to LXPLUS (where I'm submitting the job from) on both successful or unsuccesful exit.
The `+JobFlavour` defines the maximum runtime of the job in a creative way, where espresso stands for 20 minutes, microcentury is 1 hour, longlunch is 2 hours, and several others up to "nextweek" which stands for 7 days. We could define just the same thing with `+MaxRuntime` by defining the number of seconds, but "microcentury" sounds better. Note that this submission file is for running the atlas-gen-bmk workload only, because for the 3 repetitions of 7 workloads, a microcentury would be enough and the job would be killed.
The last line in the file is for queuing the job. To run it once, the key word `queue` is just enough, but we can define the number of runs too. It's possible to use variables in this file, for example instead of defining the `executable` file, we can set it to `$(filename)`, and then on the last line we can write something line `queue filename matching files *.sh` to run all the bash files in our directory. But it's not a good idea for benchmarking a node, because we want the CPU to use 100% of work for our benchmark to get the right score.
Now let's have a look on the bash script in the listing I created to run the hep-score job in HTCondor. First we need to make the my-hep.sh executable, so we need to run `chmod u+x my-hep.sh` so HTCondor can run it. As you can see in the file (Listing 3.5 on page 35), I'm using existing virtual environment I already created. First I made it in a wrong place, to be more precise I used AFS storage, because that's where you end up when you log to LXPLUS. Then I had hard time figuring out why does it not work. I was examining the HTCondor logs and trying to find a reason why is my job on-hold and not running, but the logs did not mention anything related to the cause. After some reading about LXPLUS storages (see section 3.3.1 on page 36) I found out that AFS is not supposed to be used like that, so the permissions didn't allow me to run the virtual environment in AFS from my EOS directory.
Back to the script, when I created the venv in the right place. Second line mentions creating "testdir" where we will expect the output of the hepscore, and that's where the HTCondor will find the TestBenchmark.json file which we want to transfer back to the LXPLUS to see the results. Then I pip install the hep-score from CERN's GitLab repository, and finally I run the hep-score itself, defining singularity as a type of container, hep.yaml as my configuration file (which I needed to define in the hep.sub to transfer there), and the testdir where I expect the HTCondor to find the output after a successful run.
After creating the hep.sub and my-hep.sh files and setting the right permissions, we can submit the job to HTCondor by running a command `condor_submit hep.sub` on LXPLUS. Once it's running, we can check the state of job(s) by typing `condor_q`, and see something like Listing C.2 on page 72. The first `condor_q` command shows the state of my job when I tried to use the virtual environment from the AFS (wrong) storage. Then I could get more details about held job(s) `condor_q -hold` (or with `condor_q -better-analyze 1234567.0`. After I found out the reason and fixed it, I removed the old job using `condor_q rm 1234567.0` command and submitted a new one.

```
universe              = vanilla
executable            = my-hep.sh
arguments             = $(ClusterId)$(ProcId)

output                = output_htc/hep.$(ClusterId).$(ProcId).out
error                 = output_htc/hep.$(ClusterId).$(ProcId).err
log                   = output_htc/hep.$(ClusterId).log

requirements          = (machine=?="node123.cern.ch")

transfer_input_files  = my_hep.yaml
should_transfer_files = YES
when_to_transfer_output = ON_EXIT_OR_EVICT
transfer_output_file  = TestBenchmark.json
+JobFlavour = "microcentury"

queue
```

Listing 3.4: *hep.sub* - Submission file for HTCondor

```
#!/bin/bash
source /eos/user/l/lzurkova/myvenv/bin/activate
mkdir testdir
pip install --user git+https://gitlab.cern.ch/hep-benchmarks/hep-score.git
hep-score -m singularity -f hep.yaml testdir/
```

Listing 3.5: *my-hep.sh* - Bash script to run hep-score as a job in HTCondor

3.3 Environment

In case you did not read the 2 Background section yet, I recommend to go back and get familiar with the general tools first. In this section, I'll introduce and describe tools and software used in CERN-specific cases and configurations. I'm trying to keep it short, so except basic info (like dates or abbrevations explanation) I'm trying to explain only the parts I needed to understand to do my job here.

3.3.1 LXPLUS

Lxplus stands for "Linux Public Login User Service" and it is a Linux cluster at CERN, which provides a centralized computing service for the High Energy Physics community. Users can log in remotely to lxplus from anywhere with an internet connection and can submit jobs to the HTCondor batch system for distributed computing.
The PLUS service (Public Logon User Service) is the interactive logon service to Linux for all CERN users. The cluster LXPLUS consists of public machines provided by the IT Department for interactive work[2], including batch submission like HTCondor or SLURM. LXPLUS also provides access to distributed storage resources like EOS and CERNBox.

Operating systems support

Scientific Linux CERN 6 was released in 2010 and was until at least Decembrer 2020. In 2015, CERN began migrating away from the Scientific Linux collaboration to provide the next version (RHEL 7 rebuild).
CERN CentOS 7 was released in 2014 and will be supported until July 2024. CentOS 8 at CERN was initially introduced 12.12.2019. The operating system was supported by CERN IT as the next production linux distribution of CERN, as of April 2020.

Filesystems and storage

When you log in to LXPLUS, you can access the AFS and EOS file systems and view the CVMFS read-only filesystem for experiment software. In the batch system, worker nodes have similar privileges, enabling them to access input files from and write output files to your user directories.
The AFS (Andrew File System) Service provides networked file storage for CERN users, in particular home directories, work spaces and project spaces. The AFS Service is based on OpenAFS, an open-source distributed filesystem which provides a client-server architecture for location-independent, scalable, and secure file sharing. AFS uses Kerberos for authentication and provides access control lists (ACLs) to control permissions.
EOS is a disk-based, low-latency storage service, developed at CERN, and in principle the successor of AFS. EOS is primarily used for storing and accessing large-scale data, while AFS is used for smaller files and personal user areas.
CernVM File System (CVMFS) provides a scalable, reliable and low-maintenance software distribution service. Files and directories are hosted on standard web servers and mounted in the universal namespace `/cvmfs`. It was developed to assist High Energy Physics (HEP) collaborations to deploy

[2]CERN. LXPLUS (IT-CD-CC) [online]. 2023. [visited on 2023-04-19]. Available from: https://cern.service-now.com/service-portal?id=functional_element&name=LXPLUS

software on the worldwide-distributed computing infrastructure used to run data processing applications.
CERNBox is the "cloud storage solution from CERN", providing a service similar to Google Drive, Dropbox, etc. It is built on top of EOS and the open-source ownCloud file-sync software.

3.3.2 HTCondor cluster

In the CERN's HTCondor cluster, there are several components that work together to manage the resources of an HTCondor system and execute jobs in an efficient and coordinated manner.

- CONDOR SCHEDD - The scheduler is responsible for accepting jobs and assigning them to worker nodes based on available resources, priorities, and other criteria.
- CONDOR CE - The compute element is an extension that communicates with the scheduler and manages the worker nodes. It provides information about available resources and can also submit jobs to the scheduler. It allows jobs to be executed on local resources, while providing a uniform interface to the grid-based systems.
- CONDOR CM - The central manager is the main control point for the HTCondor system, managing the scheduler and compute elements. It is responsible for keeping track of the state of the system, monitoring job progress, and enforcing policies.
- CONDOR WN - Worker nodes are the machines that actually execute the jobs. They communicate with the scheduler and central manager to receive jobs and report their status. They may also communicate with the compute element to provide information about available resources.

To visualize the environment around HTCondor cluster (service), see Fig. 3.3.2 on page 37, where LXPLUS and HTCondor acts as an intermediary between the user and worker node executing the job. The part I want to hightlight for the purpose of this work, is the fact that after a job is submitted through lxplus, condor scheduler queues it, and after the job is done, the scheduler logs details (see section 3.3.2 ClassAds) to a history file. Once a day, the current history file is flipped to a history file for that day, and later `rsync`ed to another dedicated (Thooki) node.
An option `-spool` can be used when submitting a job, which causes the output files not be generated right after the job finishes, but only when requested by user after the job is over (user has 10 days before the finished job removal). This causes the history files to not contain all the jobs in the right order, but sometimes a job ends up in the different day's history file. For this, an internal CERN Accounting tool called Thooki exists. (You can read more about 3.3.3 Thooki on page 39.)

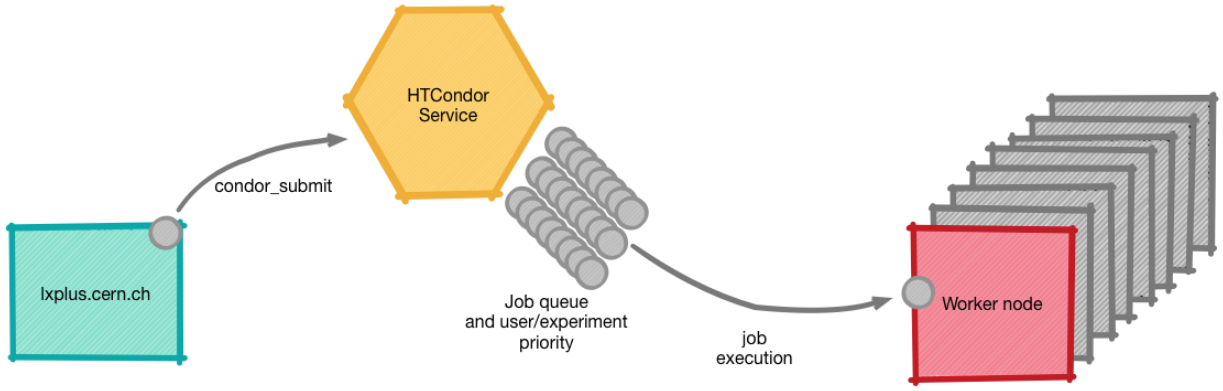

Figure 3.1: Job pattern used at Batch Service from [19]

ClassAds

In the context of HTCondor, ClassAds play a crucial role in the matchmaking and management of resources. They essentially consist of attribute-value pairs that describe the characteristics of resources and job requirements. These ClassAds are used to match jobs with suitable resources based on their attributes, allowing for efficient resource allocation and dynamic matchmaking. By leveraging ClassAds, HTCondor allows us an effective resource management within the WLCG.
Part of it is propagating the machines' benchmarking data, as we can add our own ClassAds for specific jobs or machines. Cloud team as already passing the machine's HEP-SPEC score to us, so we'll use the same way to propagate our new HEP-SCORE benchmarking results. Now let's look at some specific ClassAds:
Items with prefix `match_` are used for matching the job's requirements and machine's resources. They contain information about the available resources and their characteristics. HTCondor uses these ClassAds to find the best machines that meet a job's requirements. By comparing the job's needs with the attributes of available machines, HTCondor decides which ones are a good fit. For example if a machine has an attribute `match_cpu = 8`, and we submit a job which requires 8 CPUs, HTCondor would identify this machine as a potential match.
`RemoteWallClockTime` refers to the elapsed time, in seconds, that a job has spent running on a remote worker node. It represents the actual time taken by the job to complete its execution, regardless of any suspensions or pauses that may have occurred during its execution. RemoteWall-ClockTime is a useful metric for understanding the total time a job has utilized on the remote resources and can be used for monitoring and analyzing job performance and resource utilization. For example, if a job has this ClassAd with value 92015, it means that the job occupied the machine for $\sim$15.5 hours.

3.3.3 Thooki cluster

The Thooki cluster consist of a "master" and 3 "minion" nodes. Master node mounts the Ceph volume /accounting/incoming, allows the schedulers to rsync the history files every hour, and cleans history files older than 4 weeks. Minions run the Thooki service (internal CERN tool), and check on new unsync history files from all schedulers. The history files are divided between the 3 minions. Several processes are going on:

- copying unsynced history files from /accounting/incoming/scheduler to /acounting/thooki/raw/scheduler/[...]/history.gz
- encoding `condor_history -file; history_file -json` to /accounting/thooki/processed/scheduler/[...]/history.json - let's consider these as "working-on" files, which stays there for $\sim$ 10 days for updating the right days (because of the `-spool` option).
- parse history file and create a json for each new date
- merge files in scheduler.json and mark files as history[...].json.sync, (puts right jobs in right /accounting/thooki/processed/scheduler/[...].json file for that date)
- zips all merged scheduler files for affected dates to /accounting/thooki/zip/[...]/scheduler.json.gz and deletes all merged files
- after that, Thooki updates a database which keeps track about clear/dirty day reports and copies the .gz file to s3://DATABUCKET

3.3.4 Ceph S3 storage

For basic description of S3 and Ceph S3 see 2.5.3 on page 26. For this project, we'll be working with an instance of Ceph S3 stroage, and we'll be using two different bucket, to which I'll be referring as s3://DATABUCKET and s3://REPORTBUCKET. As mentioned in the paragraph above, s3://DATABUCKET is for storing raw data from HTCondor's hisotry files in JSON format. s3://REPORTBUCKET is a place filled by Spark jobs. To read more about these buckets and files inside them, see 5 Integrating on page 46.

3.3.5 Accounting (Spark) jobs

Accounting jobs is a piece of code which takes care of creating reports, updating data in Ceph S3, and pushing data to APEL Accounting. On a dedicated machine in CERN network, crontab is taking care of repeatedly summarizing data from previous day, month, or year in different reports. The application can do several things, depending on one of following argumets

- `cloud-summaries -start 'DD-MM-YYYY' -end 'DD-MM-YYYY'`
- `condor-summaries -start 'DD-MM-YYYY' -end 'DD-MM-YYYY'`
- `services-summaries -start 'DD-MM-YYYY' -end 'DD-MM-YYYY'`
- `monthly-summary -day D -month M -year YYYY`

- `website-table -month M -year YYYY`
- `apel -month M -year YYYY`
- `hardware -date 'DD-MM-YYYY'`
- `email -date 'DD-MM-YYYY'`

where DD, MM and YYYY should be replaced by numeric values indicating day, month and year. For my project, we'll dive more into functionality of accounting-jobs with `condor-summaries`, `monthly-summaries`, and `apel` arguments later in section 5 Integrating on page 46. The code of the jobs is structured into directories by type of work. For our purpose, we'll take a closer look at the *accounting-jobs/bin*, *accounting-jobs/daily* and *accounting-jobs/monthly* directories. Root directory contains files like *requiremets.txt, setup.py, accounting-jobs.spec, Dockerfile, Makefile*, some *README...* but we don't need to know the details of those right now.
Our journey starts with *accounting-jobs* directory, where in *accountingjob.py* a function `process_jobs()` is called. This function is using parsed arguments to append imported jobs into an array and launch them accordingly with `pyspark` and its `SparkContext()`.

3.3.6 APEL

As was already mentioned on page 27 in the 2.5.6 section, APEL is a place to public the WLCG resources. To send the data there, CERN has it's own tool called accounting-jobs, which sends SSM messages to the APEL Publisher. SSM is a text file containing a header, and records in key:value form. To see an example of SSM message, you can take a look at Listings C.5-C.4 on pages 74-73. The APEL Publisher now accepts new messages because of this project, see 4.2 on page 45.

3.4 Analysis summary

To get the "big picture" of things, you can see 3.2 on page 3.2.The WLCG, together with OSG and few other world wide computing grids are publishing it's data to APEL, which provides the gateway for data to (not only) visualize on EGI. (The WLCG is specifically designed to support the computing needs of the LHC experiments, while Open Science Grid is a broader distributed computing infrastructure that supports a wide range of scientific research projects across various disciplines.) From EGI, CERN is taking data back to CRIC, where it mixes with internal usage of grid from experiments, and uses it in MONIT Grafana to create nice plots for understanding how effectively CERN manages the resources. Now, we can "zoom in" on the part, which I'm calling "WABB", or WLCG Accounting Black Box, because it really was. You can find more details about WABB in the Fig. 3.3 on page 42. Upon exploring the workflow, I realized that the initial phase is outside the scope of my role in the IT-CD-CC section. First, the procurement team benchmarks a new CPU with a dedicated software tool called Benchmarking Suite, which already can run benchmarks like HEPSCORE, HEPSPEC, or SI2K. After that, the Cloud team propagate these

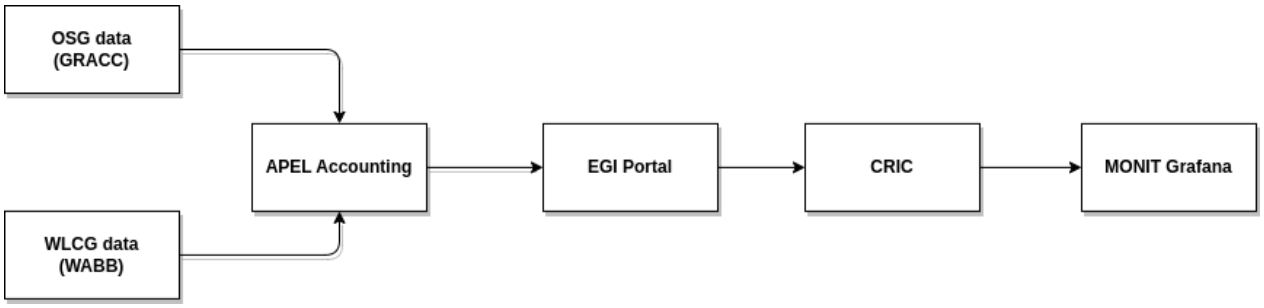

Figure 3.2: Benchmarking data workflow for WLCG Batch

values (through more layers) to OpenStack project metadata, from where my colleague can get the data and let it flow to HTCondor logs with his Puppet configuration mastery.

From the HTCondor I can get the data and send it to APEL after it goes through Thooki, S3 buckets and Spark Jobs (see Fig. 3.4 on page 43). The nice thing is that Thooki is written just to reorganize the data from logs, so all the data (including our new hepscore values) goes into the DATABUCKET in S3 Ceph. From there my task is to upgrade Spark Jobs code to send the values to APEL in a new message version (refer to section 4.2 APEL messaging update on page 45 for further details, and see Fig. 3.5 on page 43 to enjoy the visualization).

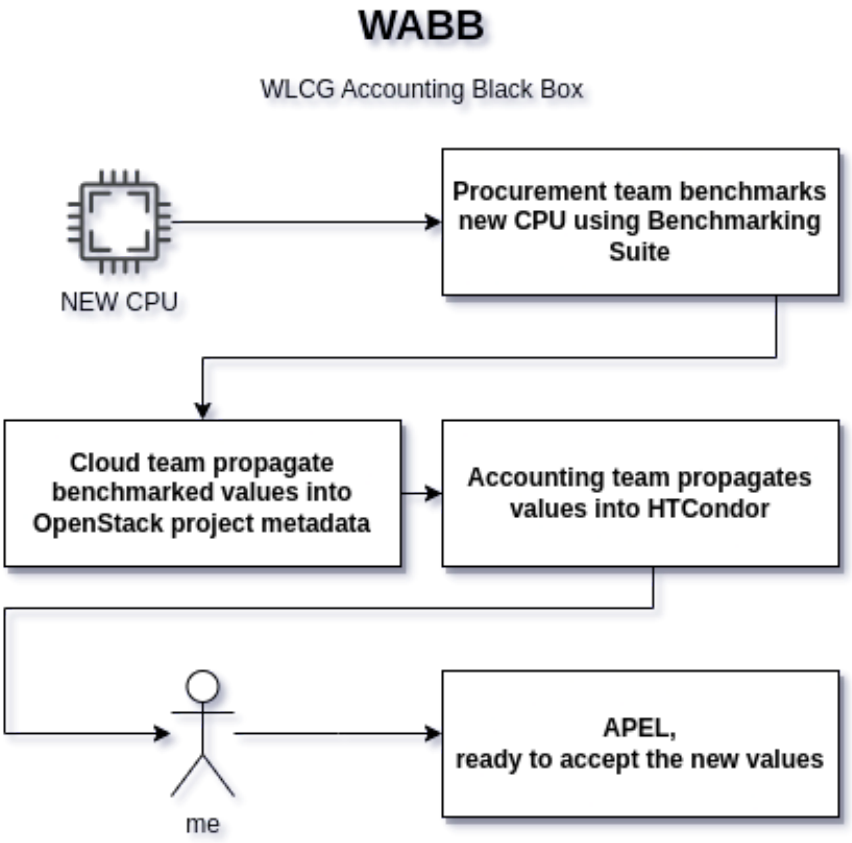

Figure 3.3: My position in WLCG Accounting Black Box usecase of benchmarking a new CPU

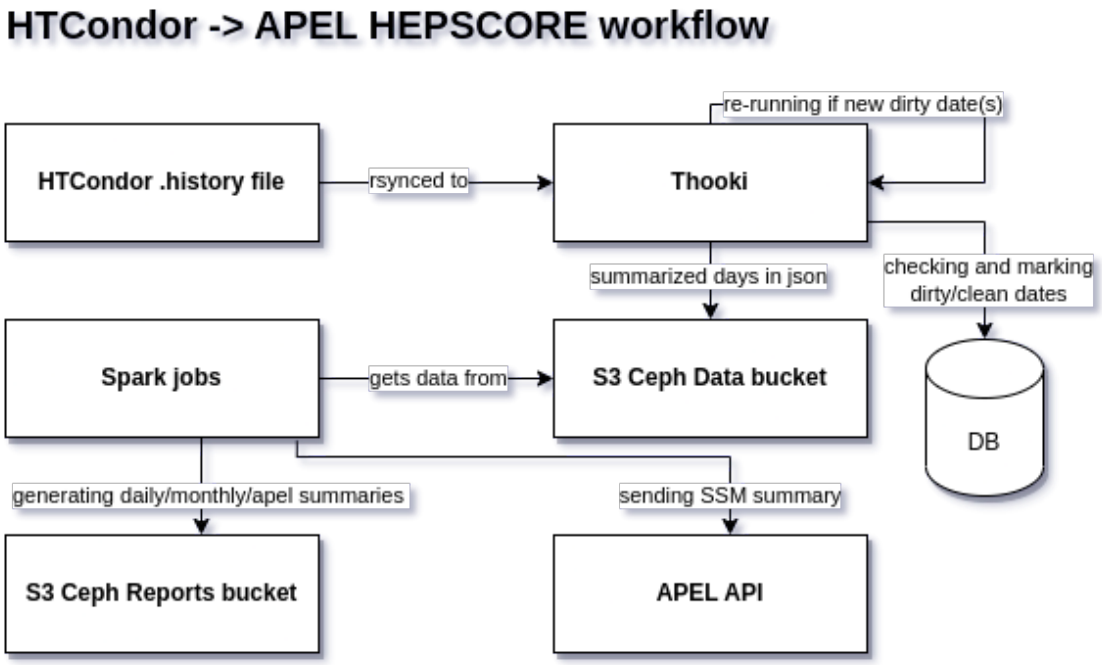

Figure 3.4: Accounting data workflow from HTCondor cluster to APEL

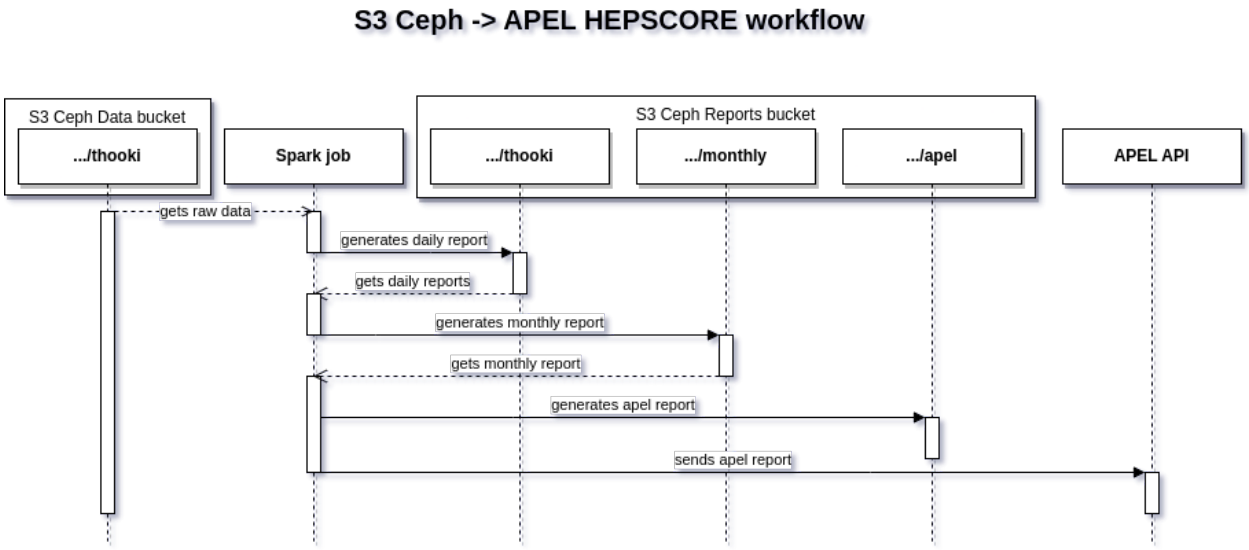

Figure 3.5: Visualization of Spark job chronological data exchange with S3 storage

Chapter 4

Changes

In my initial research project for my diploma thesis, I was tasked with proposing an alternative method to assign meaningful labels to features in the benchmark data structure. However, the decision was subsequently made by higher-level decision-makers to maintain the usage of only the score component. This decision was driven by the consideration of workload and the optimal utilization of potential additional labels. As a result, I will not be suggesting any new labeling for benchmark features, and we will rely solely on a single numerical value. Nonetheless, there are still numerous modifications to be implemented for the new benchmark, particularly in the accounting workflow of the WLCG system.

4.1 ATF decisions

A group called Accounting Task Force has been created in order to manage this transition. I personally was not part of it, but I need to be updated in case they make any more decisions. This group did meet several times before the practical integration started, and they made following decisions[1]:

- Sites won't have to re-benchmark existing resources with HEPscore
- New resource purchased by the sites will be benchmarked with HEPscore
- HEPscore will be normalized to HS06 with factor 1

These decisions leads to smoother transition and less workarounds on both places CERN and APEL. The fact that it's not necessary to re-benchmark old CPUs with new benchmark is very practical, but we still want to do that for some CPUs from newest batches, so we can compare the old and new way of accounting those nodes.

[1]Decided on the WLCG Workshop 2022 in Lancester, you can check the presentation materials on https://indico.cern.ch/event/1162261/

4.2 APEL messaging update

As a crucial component of my project, APEL has introduced a revised message format through their API. Their project specifically focuses on working with the Secure STOMP (Simple Text Oriented Messaging Protocol) Messenger, known as SSM messaging format. In the previous versions `v0.2` or `v0.3` (which were updated to v0.4 in April 2023), two key attributes were included for benchmarking results: `ServiceLevelType` to denote the benchmark name, and `ServiceLevel` to indicate the corresponding score. This meant that a single message could only publish one benchmark type and it's associated score at a time.
However, considering the introduction of the new hepscore benchmark and the need for comprehensive analytics, we aim to incorporate accounting details in both hepspec (old) and hepscore (new) metrics. This requires adapting the existing message format to accommodate multiple benchmark types and their respective scores, enabling us to gather and analyze a wider range of performance data.
There are several types of message, which APEL gathers: individual job records, summary records, and normalised messages. As SSM format is human-readable in form of "key: value" lines, you can compare the changes yourself (see Listings C.3-C.7 on pages 73-75), but let's summarize most important changes:

- key `ServiceLevelType` indicating a name of the benchmark is deprecated in v0.4
- key `Servicelevel`, `NormalisedWallDuration` and `NormalisedCpuDuration` is a float number in v0.3 indicating just a value for HS06, but for v0.4 it's associative array (dictionary) with names of benchmarks and their scores / wall durations / cpu durations

Individual job message

In addition to this change, APEL admitted there was a conflict between naming and versioning in previous versions, so instead of summary-job-message v0.3 (Listing C.5) containing keys "NormalisedWallDuration" and "NormalisedCpuDuration" we'll use a normalised-summary-message v0.4 (Listing C.6), and not summary-job-message v0.4 (Listing C.7).

4.3 My decision

Since APEL can accept a dictionary of benchmarks, I made a decision to make our Spark jobs component dynamic as well. This decision ensures that if there is a new benchmark added to the Benchmarking Suite and subsequently propagated to the HTCondor logs, it will be simpler to extend its propagation to APEL or any other reporting/archiving systems. By implementing this dynamic approach, we enhance the flexibility and adaptability of our system to accommodate future benchmarks seamlessly.

Chapter 5

Integrating

This chapter talks about more specific technical analysis of internal tool(s), to be able to implement the proposed changes. I need to check how the buckets are used, create a testing environment, implement code changes, test them in our environment, and finally deploy to production and wait for the first new batch of CPU to see if it works in production.
Unfortunately, by the time of this thesis submission, I won't get past the testing environment, because of two things: the IT department restructuring during 2022, where not all the information details were transferred about projects and internal tools. In other words, during implementation, me and my colleague Maria were discovering the technical aspects of this projects on-the-fly. Second reason is the lengthy delays of a big multinational company included in the procurement of a new batch of servers.

5.1 Buckets and reports

First, let's take a look at the S3 Ceph buckets I'll be working with and describe the data flow between them. The main data resource for Spark jobs are buckets in S3 Ceph, which I can manually manage through `s3cmd` tool on my Ubuntu, or use `boto` Python interface for Azure web services to manage buckets from the Spark jobs code. For our task, we'll need 2 buckets:

- `DATABUCKET` is a bucket, which is filled by Thooki. The structure of this bucket includes directory for each year, month, day, and contains compressed json for each machine. The json file itself contains data from HTCondor history files, where one line is one json containing ClassAds from one job. For example a path to get a file can look like "s3://DATABUCKET/[...]/condor_thooki/2023/05/01/node123.cern.ch.json.gz", and the first line formatted can look something like in listing C.8 on page 76 (the file is missing most lines, and I replaced sensitive data).

- `REPORTBUCKET` is filled by Spark jobs. We'll have a look at several directories: `condor_thooki`, `monthly`, and `apel`.
  - `condor_thooki` directory has similar structure as in DATABUCKET, but directory for each day contains 2 files named "results". One as JSON, second as CSV. The path to a json file can look like "s3://REPORTBUCKET/[...]/condor_thooki/2023/05/01/results.json". The files itself contains summarized data for each day, including `cpu, qfan, hepspec_cpu, hepspec_wall, infrastructure, numcores, numjobs, vo` and `wall` (see listing 5.1 on page 48). (These data are not secret anymore, you can find them by applying the right filter on the EGI web application for 1st April 2023. I'll leave this exercise on a reader.)
  - `monthly` contains data of all the machines in results.json and results.csv files in directories of years and months. Example of reaching a csv file can be "s3://REPORTBUCKET/[...]/monthly/2023/05/results.csv", and the file can look like Lst. 5.2 on page 48.
  - `apel` directory contains a summary file in SSM (Secure STOMP Messenger) format for each day. This file looks like a text file with a header and several messages separated by `%%`. Example of filepath concerning this file can be "s3://REPORTBUCKET/[...]/apel/2023/05/01/summary.ssm".

First, the Thooki tool fills bucket called DATABUCKET with the data from HTCondor .history files. Then Spark jobs are taking this data to create condor summaries, then monthly summaries, and finally APEL messages.

Let's see the format of data in the DATABUCKET: The file described in listing C.8 on page 76 contains the data from .history files straight from HTCondor. Featuring mosthly key-value pairs containing details about configuration and machine-matching done by batch system. (Notice how keys with prefixes "match_exp" and "match" matches? That's an example of successfull matching by the HTCondor.)

Now, when accounting-jobs are called with argument condor-summaries, it creates the content of thooki directory in REPORTBUCKET as in the listing 5.1 on page 48. Featured numbers are not secret, because those are Atlas's usage of grid which we include in reports. But other can be local which we do not report to a third-party services. This day report is summarizing all the jobs submitted by Atlas to the HTCondor batch system, which were finished on 1st April 2023.

After this report is created for every day in a month, the monthly summary can be generated by calling Spark jobs with argument monthly-summary, and REPORTBUCKET's directory monthly is filled with a "results" file (see listing 5.2 on page 48. This file contains monthly records for each VO, including normalised CPU and wall durations. After monthly reports are done, we can send data to APEL. Again, Spark jobs grabs previously generated report (the monthly one this time) and generates the SSM message for APEL as in APEL-summary-job-message: v0.3 (see Listing C.5 on page 74), including the normalised wall/cpu duration again.

```
[
    {
        "cpu": 663537996,
        "fqan": "atlas, grid pilot",
        "hepspec_cpu": 8196551140,
        "hepspec_wall": 8451537072,
        "infrastructure": "grid",
        "numcores": 1,
        "numjobs": 26437,
        "vo": "atlas",
        "wall": 683610865
    },
    #[...]
]
```

Listing 5.1: Example of a shortened file in the REPORTBUCKET in thooki directory for a specific day (1st April 2023)

```
   {
   "facts" : {
      "hepspec_factor" : 30.8629694951544,
      "numcores" : 1.96600017200688
   },
   "records" : [
      #[...]
      {
         "AverageDailyWallDuration" : "6411.00",
         "Chargeback" : true,
         "CpuDuration (d)" : "163775.30",
         "Infrastructure" : "grid",
         "Month" : 4,
         "NormalisedCpuDuration (hs06d)" : "2106232.17",
         "NormalisedWallDuration (hs06d)" : "2482476.22",
         "Notes" : "",
         "NumberOfCores" : 1,
         "NumberOfJobs" : 796746,
         "Resource" : "condor",
         "Site" : "CERN-PROD",
```

```
        "VO" : "alice",
        "VORole" : "public, grid sgm",
        "WallDuration (d)" : "192329.94",
        "Year" : 2023
      },
      #[...]
   ]
]
```

Listing 5.2: Example of a shortened file in the REPORTBUCKET in monthly directory for a specific month (April 2023)

Now you know all the buckets and file formats we'll be working with in "my part" of changing the workflow to add HEPSCORE to reports. But to get the initial data from DATABUCKET filled by Thooki, which is luckily dynamic and adds everything from the HTCondor history files, we need to get the hepscore to the history file. Even though it's not "my part", it's still very interesting how many people and processes must be involved to get it done. First, the procurement team needs to dedicate a machine to run our new benchmark, and write a new script for their machine-adding procedure, which includes "cleaning" the new CPU. This new script will now include instructions to run our new benchmark on the dedicated machine, and propagate the score to IRONIC (which is a tool similar to OpenStack, but for physical machines). The score is then propagated to OpenStack project metadata by the Cloud team, and from there, my colleague Ben will propagate it to HTCondor logs.
From the .history files I can grab the data and update our workflow to propagate it to APEL. But before I can do it, I need to ask Luca and Nikos (procurement team), José (cloud team) and Ben (batch team) to get the data there.

5.1.1 Mocking data

Before a new machine will be purchased for our batch system, and a new workflow applied (including hepscore benchmarking), it takes some time. For June 2023, no new machines are planned to join the production in the near future. The testing buckets are ready, prepared in the same structure as production, but empty now. That's why I want to mock the data in our new TESTING-DATABUCKET, because I would need to wait too long before testing my code. For this purpose, I created a script (see Listing C.9 on page 77) that adds the new hepscore ClassAds into a file from DATABUCKET. I didn't want to waste much time on that script, because it'll be only run few times, and only for my purpose to mock the data before the new production data will be available, so please, pardon my "French" code. There are things which could have been written in a better way, use more arguments instead of variables etc, but for my purpose I decided to invest my energy in other parts of this project. To think about memory usage a little bit, I created the script to run

it day by day, so we won't need several GBs of free space on my computer, no matter date interval do we want to mock.
In reality, I created 3 scripts: C.9 mock.py, C.10 mock-day.sh, and C.11 mock-month.sh (the mock-month.py can mock any length, but the initial idea was to mock a month to be able to create the monthly and APEL summary), see pages 77 - 79.
The first python script takes care of locally mocking a directory in a specific format. I will not include the arguments parsing parts, because it's not important for this thesis. So when you look at listing C.9 on page 77, remember the variables day, month and year represents a day, a month and a year. And variable `dir` represent absolute path of a directory we're working in. You can run this script by calling `python3 mock.py -date 2023-05-01`, and it'll inform you about the progress by printing a line after a directory was mocked. But for the script to work, you already need to have your directory(ies) `original-XX` filled with original data.
And because I didn't want to spend more time on getting the data, and then syncing it to S3, I created the mock-day.sh script (see listing C.10 on page 78). To run this script, you need to have s3 configuration files for both production and testing environment. You can run the script calling `./mock-day.sh -y 2023 -m 05 -d 01` after allowing the script to be ran with `chmod` tool. For this thesis, I edited just the bucket paths in this script.
Now, I don't want to spend time by running this script once every hour or so, so I created another script, which allows me to enter an interval to mock. (I do not enjoy long scripts, that's why I rather created a new one to call my mock-day, instead of editing the old one.) Check out the listing C.11 on page 79 to get the idea.

5.2 Testing environment

As we need to implement everything carefully to not destroy the accounting data, first we need to plan the changes needed to do the upgrade. In practice that means to prepare a testing environment to try the code changes and to discuss it further with colleagues. In my analysis I found out that the only piece of software which needs changes is the Spark jobs (accounting-jobs) tool, ran by a crontab on a dedicated node. So together with my colleague Maria we created and configured a testing node.

5.2.1 APEL testing environment

There are two ways to send an SSM message to APEL: using AMS (Argo Messaging System) token, or using a DN (Distinguished Name). Spark jobs are using the DN to send a message in production, so to test our new message format on the testing node, we'll use it as well. This means adding our new testing DN to CERN's GOCDB gLite-sql service, and then reaching people from APEL to let them know about our new DN. The other option is to ask for an AMS token, which is the included

in the request's metadata.
A test repository from APEL has been set up. To send records, we have chosen the DN that has a corresponding gLite-APEL service in GOCDB and informed the APEL team of the DN, and configured the SSM with AMS protocol, defined the right `host`, `ams_project` and `destination`.

5.2.2 Testing node

Of course we encountered some challenges when trying to make the testing node work. First we made our testing node using QA branch configuration of the Puppet for Spark Jobs. After deploying my new version of code (see section 5.3 Spark job changes on page 51), the APEL sending part still did not work. We started getting pip error, which was traced back to invalid syntax in certifi package core for Python 2.7. After checking logs and searching on the internet, I found out it's a versioning problem for a requests package. After some more digging in the code, I discovered that requirements (packages to install) for pip package manager were defined, but without versions, and together with QA's puppet configuration we were using another different-version in-system tools like ssmsend, which is used to send the SSM to APEL. This is something I - as a person with some web applications developing experience - considered as a bad practice.
Maria was so kind to reinstall our testing node with a Puppet configuration for production, changing just the details concerning the production environment, for example keys and secrets to access S3 storage. And guess what, it worked like a charm.

5.3 Spark jobs changes

In this section, I'll describe the workflow and changes of the `accounting-jobs` (Spark jobs) tool I worked on. The order of steps is not chronological to the way I was figuring it out, but it follows the order of executing the commands, because they need to be ran one after another. First, to get to know the code I tried to modify parts of it in Jupyter notebook on SWAN using Spark cluster. I mocked the input instead of reaching S3 buckets from this environment, and just printed outputs instead of saving/sending reports. This approach worked for `monthly-summary` and `apel` parts, but for `condor-summaries` there is a lot of filtering and grabbing data from different directories, which would probably take me longer to mock the input data then wait for my testing node working with S3 storage.

5.3.1 Creating `condor-summaries` report

Basically the changes were only in two files. In the config.py file, consisting of defined variables, I just added new items into a variable with "hepscore" prefix. The new `CSV_FIELDS` array is now defined as `['vo', 'fqan', 'infrastructure', 'numcores', 'wall', 'cpu', 'hepspec_wall', 'hepspec_cpu', 'hepscore_wall', 'hepscore_cpu', 'numjobs']`. That was easy, right?

The second part in the condor.py file needs a little bit more understanding of the code. I will not describe the whole code of accounting-jobs project, but I'll try to summarize all the parts we need to do this change. Let's start with the function `spark_job(self, start, end, fields=None)` . This method is the entrypoint for the CondorTK class - it's responsible for the acquisition of data from S3, and generating correlated reports. Parameters `start` and `end` are dates to be used when reading data from S3.
After `spark_job()` filters out incomplete data, it calls `_upload_report(self, data_frame, fields, upload)`, which calls `_resolve_daily_reports(self, df_aux, fields)`, which calls `_generate_daily_report(self, df_aux, fields)`. Here in the generating method we encounter the first "hepspec" mentions.
The function generating daily report is responsible for aggregation of the data frame data into the desired columns, and accepts two parameters: `df_aux` is the Spark dataFrame to extract the data frame, and `fields` are the fields to be used to output the data frame columns (in our case, the function is called with the fields defined in the config.py, which I mentioned in the paragraph above. First, the data in the variable gets filtered out according to pre-defined blacklist, so we only get data which we want in the condor summaries. After that, a new dataframe is created using the filtered `df_aux` variable using the pyspark's `withColumn` dataframe method. See Listing C.12 on page 80 to see what does it look like. For this part, I just duplicated the parts with "hepspec" and renamed it to "hepscore", and the new code produces very similar output as the original one on listing 5.1 on page 48, just with new keys "hepscore_cpu" and "hepscore_wall". Easy one, right?

5.3.2 Creating `monthly-summary` report

First thing I did was to rename a variable defining keys for normalised CPU and wall durations, which had a "hs06d" unit in the name to just "d" as days, because the value of this key-value pair will now be a dictionary with all the benchmark names and their score.
Original monthly summary record output features two items in the json file: one under the key "facts" which gives us a short monthly summary, and "records" which gives us mothly summaries for each VO in the context if their role (for example VO Alice can have several roles, but only the public grid is published to APEL, so the "Chargeback" is true). To make this dynamic, I had to analyze the code and fill the normalised durations with all the benchmarks.
First, I changed the "calculate_hs06" method which is used for the monthly report to generate the factors (the first item in the report, see 5.2 on page 48). Old method just took sum of all the "hepspec_wall" values, and divided it by the sum of "wall" values. But my new method will need to return scores for all the benchmarks. So first I prepared a dictionary variable "result_factors" to fill and return, and then I got benchmark names from the whole month (I created a method which goes through the whole month and returns unique benchmark names for the whole batch). Then for each benchmark I summed their wall values and divided it by the summed "wall" value as in

the original version. A problem I had to handle here, is missing benchmarks - not all records from the batch have been tested with all the benchmarks. So for every record I checked again, if it has a value for the specific benchmark, and only then I added the values to the sums.
Then I did several very similar changes in other parts of the code, for example in the method which adds "Notes" to records, one of the use-cases is when CPU usage exceeds wall duration - this I did dynamic for each benchmark name, as it handles one-by-one report in the batch and adds notes.
Few other finalising touches were done, like dividing the durations by seconds of the day, or formatting it to 2 decimal places, and the report is good to go. You can check the new version of monthly report in listing 5.3 on page 53 and compare it with the old one (see Listing 5.2 on page 48).

```
{
   "facts" : {
      "hepscore" : 39.5656119750508,
      "hepspec" : 32.972658617077,
      "numcores" : 1.83018047022626
   },
   "records" : [
      #[...]
      {
         "AverageDailyWallDuration" : "6070.24",
         "Chargeback" : true,
         "CpuDuration (d)" : "153204.47",
         "Infrastructure" : "grid",
         "Month" : 5,
         "NormalisedCpuDuration (d)" : {
            "hepscore23" : "2337104.01",
            "hepspec" : "1947586.23"
         },
         "NormalisedWallDuration (d)" : {
            "hepscore23" : "2856599.13",
            "hepspec" : "2380499.23"
         },
         "Notes" : "",
         "NumberOfCores" : 1,
         "NumberOfJobs" : 1124548,
         "Resource" : "condor",
         "Site" : "CERN-PROD",
         "VO" : "alice",
         "VORole" : "public, grid sgm",
```

```
        "WallDuration (d)" : "188177.50",
        "Year" : 2023
      },
      #[...]
    ]
}
```

Listing 5.3: Example of a shortened file in the TESTING-REPORTBUCKET in monthly directory for a specific month (May 2023)

5.3.3 Creating `apel` report

As mentioned in section 4.2 Apel messaging update on page 45, there was a conflict between naming and versioning, so my first code change was to change the header of the message from "APEL-summary-job-message: v0.3" to "APEL-normalised-summary-message: v0.4". Then, in a part where the metrics are normalised (divided by the number of cores), I again made a change to make it dynamic - the method accepts the record and returs a modified record, so instead of getting the normalised durations as a number, the method gets there a dictionary. I just made a loop through the dictionary and returned it the whole record modified with values divided by the number of CPUs. Another method connected with the new dictinary was needed, and I called it "convert_dict_units" to call it on the dictionary just before creating the SSM message. It just takes Normalised durations (for both CPU and wall), and divide it by the number of cores.

5.4 Summary

When I now compare the master branch and my branch in the GitLab project, the statistics says that 13 files changed with 289 additions and 268 deletions, from which 46 additions and 189 deletions are related to a configuration of gitlab pipeline and makefile for testing and fixing the proccess of creating an RPM package to install on the node. As a result, I added or replaced 164 lines of code. That does not seem much, as I like to keep my code expressive and easy for understanding, but in the context that this small change was made possible by an analysis of the entire system and workflow, it has a great value.

During the project, I conducted an analysis of the existing infrastructure and tools utilized in the WLCG accounting workflow. This examination allowed me to identify the strengths and weaknesses in the current system. Additionally, I acquired a comprehensive understanding of running the HEPSCORE benchmark both locally and within the batch system, gaining insights into the meaning behind the provided numerical values.

To integrate the updates proposed by a third-party service, I implemented the necessary changes in the code of an internal tool responsible for a part of accounting workflow. However, due to

the lengthy processes involved in adding CPUs within a multinational company, I was unable to compare the results of the old (HEPSPEC) and new (HEPSCORE) benchmarks to assess the effectiveness of the accounting system on production data, so I worked with mocked data during my analysis. Nonetheless, given the nature of accounting and planning, it will be possible to evaluate the effectiveness of these benchmarks in the coming years.

Chapter 6

Conclusion

In this final chapter, I will provide a recap of the project objectives, highlighting the contributions made by myself and my team. I will discuss the challenges we faced and the solutions we developed. I will present the results and findings obtained from our work and share the lessons we learned during my 10-month internship at CERN. Finally, I will conclude by summarizing the project's outcomes and proposing some future recommendations.

6.1 Overview

In this thesis, my main objectives were to:

- Take a close look at the existing environment and technology being utilized at CERN. I aimed to gain a comprehensive understanding of the concept of benchmarking and its role in the accounting workflow of the WLCG.
- Familiarize myself with the significance of the score generated by HEPSCORE. I delved into the details of the data structure associated with the new benchmark, seeking to grasp its underlying components.
- Propose a practical approach to effectively label the features within the benchmark data structure, ensuring that they are meaningful and relevant. This proposition was made by higher authority, and I followed their lead.
- Implement necessary updates to the Batch accounting system, enabling it to seamlessly accommodate the suggested data format.
- Conduct a thorough evaluation and comparison of the obtained results, thereby assessing the effectiveness and impact of the implemented changes.

6.2 Challenges and solutions

Starting a new project and living abroad for the first time can be tough, but it's a great learning experience. Both involve getting to know the new environment, understanding how things work, and fitting in. In the beginning of my internship, end of 2022, I focused on analyzing my surroundings, meeting colleagues, and exploring internal environment and tools like hepscore code, lxplus, HTCondor, and S3 Ceph storage. Prior to this, my background was in web application development, so learning about batch systems and grid computing was an enlightening experience.
Afterwards, I investigated how benchmarking data flows through the WLCG Accounting systems, which took a long time as there's no centralized source of knowledge available. I needed to assemble the broader picture together, just like solving a puzzle. Whenever I encountered a puzzle piece, I would search the internet or CERN's network to determine where it fits. In most cases, I was successful, but when faced with bigger challenges, I sought assistance from my colleagues. Initially, this was a challenging aspect for me because everyone had their own work to do. However, I realized the importance of seeking information from them, documenting the knowledge and consolidating it into a paper or document. It's essential to ensure that valuable information isn't stored in people's minds but is shared and accessible to others.
Once I understood the data flow, I realized that the only changes needed were in Spark jobs. I delved deeper into CPU benchmarking to involve the right people and make the necessary adjustments to include HEPSCORE values in WLCG Accounting. Since I needed data for testing before it was available in production, I created a test environment and updated the Spark jobs code.
APEL provided a testing environment for submitting new message versions, but our Spark jobs needed to handle different configurations. To ensure we didn't affect production data, we set up a test machine to send messages to the correct endpoint. I used the Swan tool to connect to CERN's Spark cluster and made changes to the Spark jobs code. By the time the test machine was ready, my code was already updated in git, and it successfully generated reports from the test data, which were sent to APEL.

6.3 Contributions and results

My main contribution to the WLCG Accounting project at CERN involved enhancing the Spark jobs code to incorporate the reporting of new HEPSCORE values to a third-party service. Working alongside Maria, we established a fully functional testing environment for Spark jobs, which includes a dedicated node configured for testing and a separate S3 Ceph instance with testing buckets. This environment allows us to work with test data before new batches of CPUs are measured using HEPSCORE. By implementing a dynamic approach to code changes, I ensured that adding additional benchmarks in the future will be much simpler, without the need for extensive modifications to the Spark jobs code.

The dynamic change and testing node ensures that any potential new benchmark or feature can be easily tested and evaluated.

In terms of recommendations for future improvements, I suggest prioritizing proper code documentation and commenting within the internal tools. It is crucial to ensure that knowledge is not dependent on the presence of specific individuals who possess it. In an environment where knowledgeable individuals come and go more swiftly than expected, valuable insights can be lost if adequate documentation practices are not implemented.

Bibliography

1. DARWIN, Charles. *The Origin of Species*. Ed. by APPLEMAN, Philip. New York: Norton, 1859.
2. CHARPENTIER, P. Benchmarking worker nodes using LHCb productions and comparing with HEPSpec06. In: *Journal of Physics Conference Series*. 2017-10, vol. 898, p. 082011. Journal of Physics Conference Series. Available from DOI: 10.1088/1742-6596/898/8/082011.
3. JAIN, R. *The art of computer systems performance analysis: techniques for experimental design, measurement, simulation, and modeling*. Wiley New York, 1991.
4. CERN. *The CERN accelerator complex, layout in 2022* [online]. 2023. [visited on 2023-06-29]. Available from: https://cds.cern.ch/record/2800984.
5. CRITCHLOW, T.; DAM, K.K. van. *Data-Intensive Science*. Taylor & Francis, 2013. Chapman & Hall/CRC Computational Science. ISBN 9781439881392. Available also from: https://learning.oreilly.com/library/view/data-intensive-science/9781439881415/.
6. SHIERS, Jamie. The Worldwide LHC Computing Grid (worldwide LCG). *Computer Physics Communications*. 2007, vol. 177, no. 1, pp. 219–223. ISSN 0010-4655. Available from DOI: https://doi.org/10.1016/j.cpc.2007.02.021. Proceedings of the Conference on Computational Physics 2006.
7. CERN. *WLCG Accounting Reporting Portal CRIC* [online]. 2023. [visited on 2023-06-29]. Available from: https://wlcg-cric.cern.ch/wlcg/reporting/tier1/report.
8. XIE, Gang; XIAO, Yong-Hao. How to Benchmark Supercomputers. In: *2015 14th International Symposium on Distributed Computing and Applications for Business Engineering and Science (DCABES)*. 2015, pp. 364–367. Available from DOI: 10.1109/DCABES.2015.98.
9. GIORDANO, Domenico. *HEPiX Benchmarking WG update* [online]. 2021. [visited on 2023-04-13]. Available from: https://indico.cern.ch/event/1078853/contributions/4576275/attachments/2334610/3979470/HEPiX-Workshop-Autumn-26-10-2021-giordano.pdf.
10. DOCKER INC. *Networking overview* [online]. 2023. [visited on 2023-04-17]. Available from: https://docs.docker.com/network/.

11. SYLABS INC. *Security in Apptainer* [online]. 2022. [visited on 2023-04-17]. Available from: https://apptainer.org/docs/user/main/security.html.

12. THAIN, Gregory. *Welcome and Introduction to HTCondor* [online]. 2023. [visited on 2023-04-17]. Available from: https://htcondor.readthedocs.io/en/latest/users-manual/welcome-to-htcondor.html.

13. SLURM TEAM. *Scheduling Configuration Guide* [online]. 2023. [visited on 2023-04-17]. Available from: https://slurm.schedmd.com/sched_config.html.

14. IBM. *About IBM Spectrum LSF* [online]. 2023. [visited on 2023-04-18]. Available from: https://www.ibm.com/docs/en/spectrum-lsf/10.1.0?topic=started-quick-start-guide.

15. REDHAT. *Chapter 7. Object Gateway S3 API* [online]. 2023. [visited on 2023-04-13]. Available from: https://access.redhat.com/documentation/en-us/red_hat_ceph_storage/1.3/html/object_gateway_guide_for_red_hat_enterprise_linux/object_gateway_s3_api.

16. S3TOOLS.ORG. *Amazon S3 Tools: Command Line S3 Client Software and S3 Backup* [online]. 2023. [visited on 2023-04-13]. Available from: https://s3tools.org/s3cmd.

17. UNIVERSITY OF CALIFORNIA. *Boinc Overview* [online]. 2023. [visited on 2023-04-13]. Available from: https://boinc.berkeley.edu/trac/wiki/BoincOverview.

18. BRUN, Rene; CARMINATI, Frederico; CARMINATI, Giuliana Galli (eds.). *From the web to the grid and beyond*. 2012th ed. Berlin, Germany: Springer, 2012-01. The Frontiers Collection.

19. CERN. *Overview - Batch Docs* [online]. 2023. [visited on 2023-06-29]. Available from: https://batchdocs.web.cern.ch/concepts/index.html.

20. CERN. *Tiers - Woldwide LHC Computing Grid* [online]. 2023. [visited on 2023-06-29]. Available from: https://wlcg-public.web.cern.ch/tiers.

21. CERN. *Regional Centers and pledges provided by VOs* [online]. 2023. [visited on 2023-06-29]. Available from: https://wlcg-cric.cern.ch/core/pledge/list/.

22. CERN. *VO Requiremet list* [online]. 2023. [visited on 2023-06-29]. Available from: https://wlcg-rebus.cern.ch/core/vopledgereq/listcomp/.

Appendix A

Figures

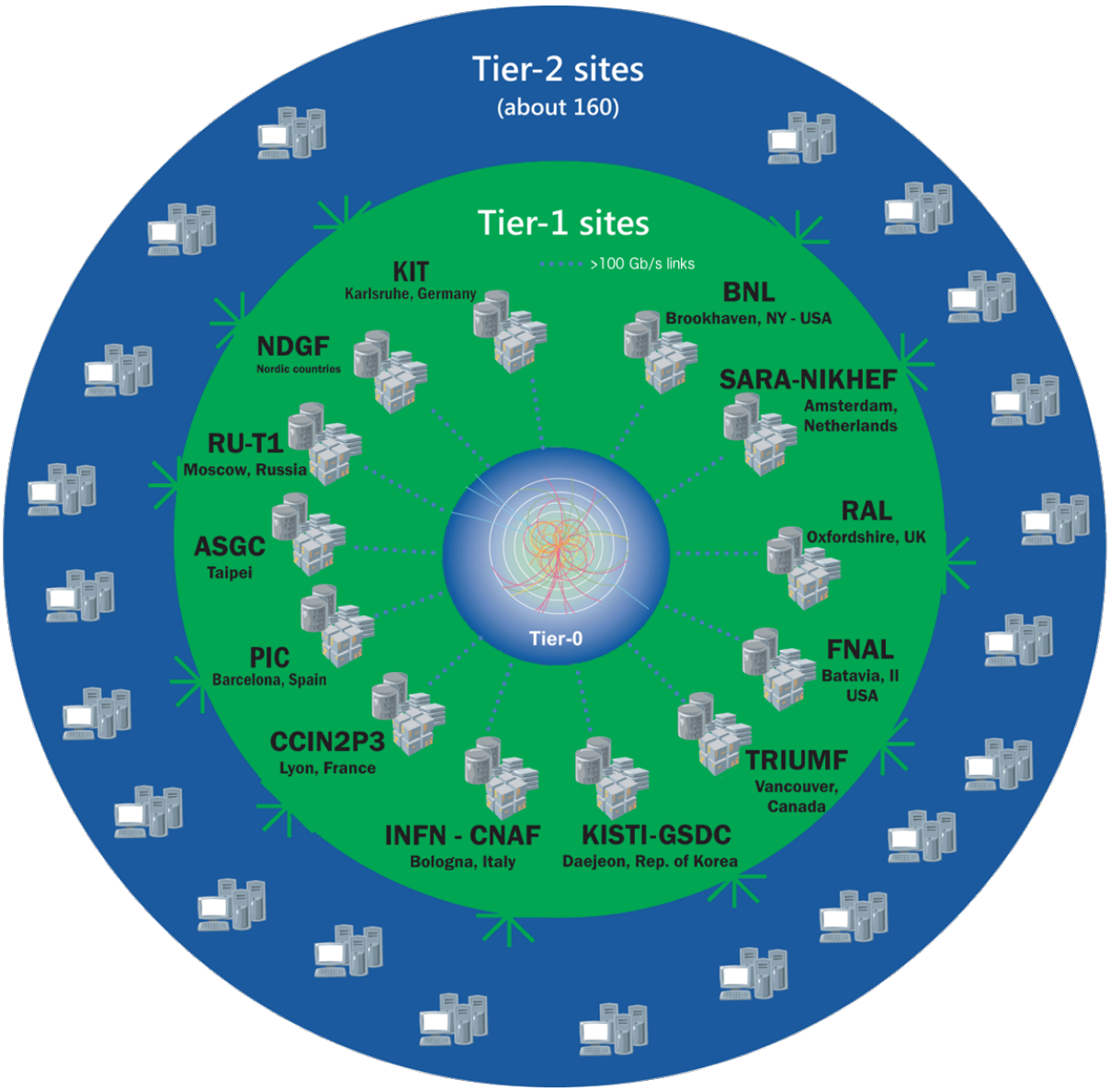

Figure A.1: The tiers of WLCG [20]

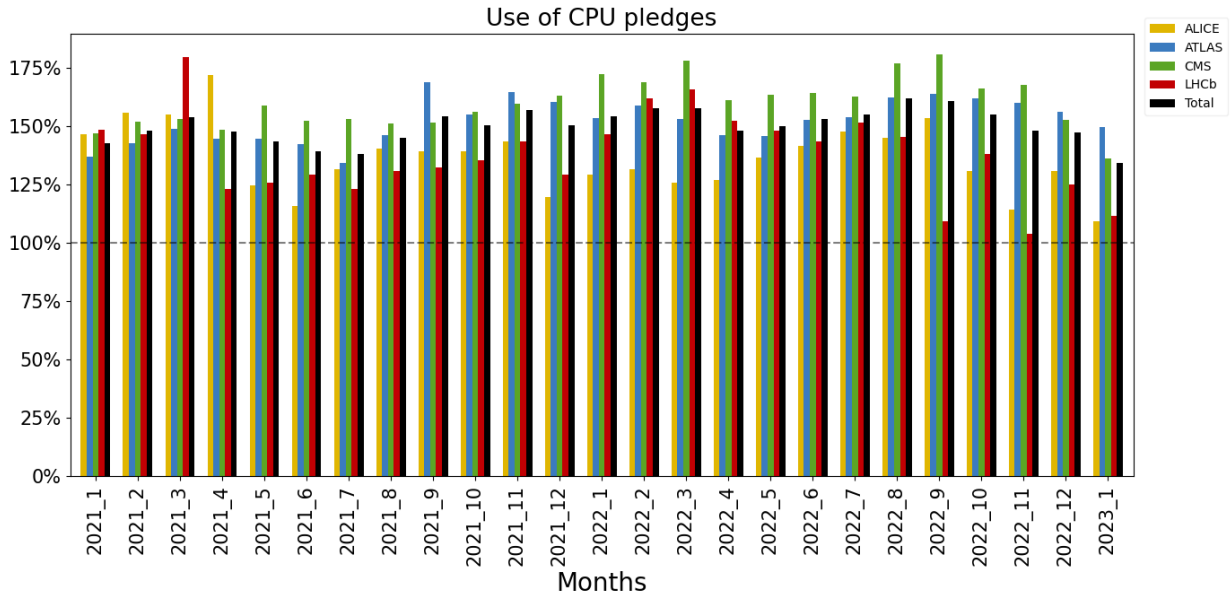

Figure A.2: Pledge usage for 2022

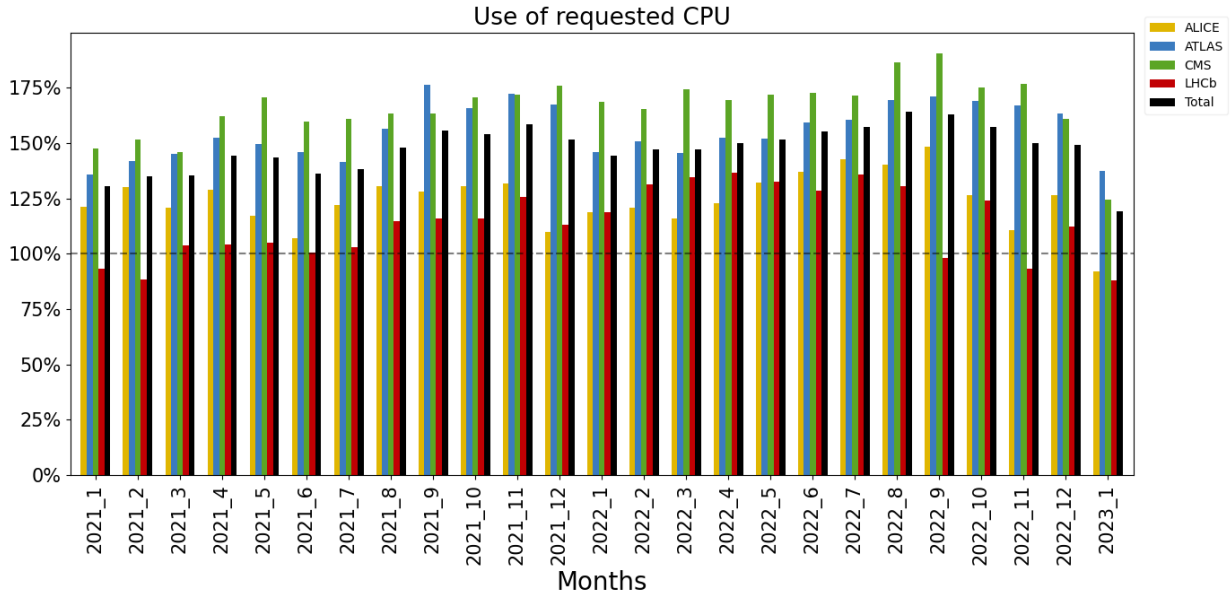

Figure A.3: Requested usage for 2022

Appendix B

Tables

Federation	Tier	VO	Country	Pledge for 2022 (HS06)	Pledge for 2023 (HS23)
CH-CERN	0	ALICE	Switzerland	471 000	541 000
CH-CERN	0	ATLAS	Switzerland	550 000	740 000
CH-CERN	0	CMS	Switzerland	540 000	720 000
CH-CERN	0	LHCb	Switzerland	189 000	215 000
CZ-Prague-T2	2	ALICE	Czechia	20 000	24 400
CZ-Prague-T2	2	ATLAS	Czechia	30 000	35 000

Table B.1: CERN and Czech sites CPU pledges for 2022 and 2023 [21]

VO	Tier	Required	Pledged	Difference
ALICE	0	471 000	471 000	0 %
	1	498 000	448 440	-10 %
	2	515 000	516 611	0 %
ATLAS	0	550 000	550 000	0 %
	1	1 300 000	1 383 650	+6 %
	2	1 588 000	1 655 983	+4 %
CMS	0	540 000	540 000	0 %
	1	730 000	851 910	+17 %
	2	1 200 000	1 209 534	+1 %
LHCb	0	189 000	189 000	0 %
	1	622 000	514 531	-17 %
	2	345 000	332 640	-4 %
Summary	0	1 750 000	1 750 000	0 %
	1	3 150 000	3 198 531	+2 %
	2	3 648 000	3 714 768	+2 %

Table B.2: VO CPU Pledge and Requirement list for 2022 [22]

Workload	Running Time (m)	# of events	# of threads	beta-selected
`atlas_gen_sherpa`	31	200	1	yes
`atlas_reco_mt`	69	100	4	yes
`atlas_sim_mt`	156	5	4	
`cms_gen_sim`	42	20	4	yes
`cms_digi`	31	50	4	
`cms_reco`	51	50	4	yes
`belle2_gen_sim_reco`	25	50	1	yes
`alice_gen_sim_reco`$*$	194	3	4	
`lhcb_gen_sim`	104	5	1	yes
`juno_gen_sim_reco`	67	50	1	
Gravitational Wave	138	1	4	
Total	908 (15+ hours)			
Total beta-selected	322 (5+ hours)			

$*$ - Alice time is high due to technical problems with reco workload. Reco is $\sim$50% of running time.

Table B.3: HEP workloads running time in minutes. Times are geometrical mean of three runs on reference machine.

Machine	Note	Score for `atlas-gen-bmk`	Score for hepscore-beta set
Core(TM) i5-10310U CPU @ 1.70GHz	my laptop	46.221	-
Xeon(R) CPU E5-2650 v4 @ 2.20GHz	my WLCG node	206.5529	150.1573
Xeon(R) CPU E5-2630 v3 @ 2.40GHz	reference machine	-	80.0000

Table B.4: Intel(R) processors performace comparison in units of the new HEPSCORE benchmark

Appendix C

Listings


```
{
   "app_info" : {
      "config_hash" : "cc648ef17e4bed772c3db48ddc86e3783a57342f4fba59eb672c358256fedd2d",
      "hepscore_ver" : "1.5.0"
   },
   "benchmarks" : {
      "atlas-gen-bmk" : {
         "app" : {
            "bmk_checksum" : "48598ad84ee79d858700ecc86b4e7c9a",
            "bmkdata_checksum" : "809f5a89604aab45e53d3f1f69d6c6f9",
            "containment" : "singularity",
            "cvmfs_checksum" : "1940e3491d437c6325aff2a2d5ab8fdd",
            "description" : "ATLAS Event Generation based on athena version 19.2.5.5",
            "version" : "v2.1"
         },
         "args" : {
            "events" : 5,
            "threads" : 1
         },
         "results_file" : "atlas-gen_summary.json",
         "run0" : {
            "duration" : 773,
            "end_at" : "Wed Mar 29 13:15:34 2023",
            "report" : {
               "log" : "ok",
               "wl-scores" : {
                  "gen" : 48.4736
               },
               "wl-stats" : {
```

```
        "avg" : 6.0592,
        "count" : 8,
        "max" : 6.1614,
        "median" : 6.0644,
        "min" : 5.9488
      }
    },
    "start_at" : "Wed Mar 29 13:02:41 2023"
  },
  "run1" : {
    "duration" : 628,
    "end_at" : "Wed Mar 29 13:26:02 2023",
    "report" : {
      "log" : "ok",
      "wl-scores" : {
        "gen" : 49.9976
      },
      "wl-stats" : {
        "avg" : 6.2497,
        "count" : 8,
        "max" : 6.2893,
        "median" : 6.2441,
        "min" : 6.1805
      }
    },
    "start_at" : "Wed Mar 29 13:15:34 2023"
  },
  "run2" : {
    "duration" : 628,
    "end_at" : "Wed Mar 29 13:36:30 2023",
    "report" : {
      "log" : "ok",
      "wl-scores" : {
        "gen" : 50.0597
      },
      "wl-stats" : {
        "avg" : 6.2575,
        "count" : 8,
        "max" : 6.2972,
        "median" : 6.2617,
        "min" : 6.192
      }
    },
```

```
            "start_at" : "Wed Mar 29 13:26:02 2023"
         },
         "run_info" : {
            "copies" : 8,
            "events_per_thread" : 5,
            "threads_per_copy" : 1
         },
         "version" : "v2.1",
         "weight" : 1
      }
   },
   "environment" : {
      "arch" : "x86_64",
      "end_at" : "Wed Mar 29 13:36:30 2023",
      "singularity_version" : "3.10.2",
      "start_at" : "Wed Mar 29 13:02:41 2023",
      "system" : "Linux lzurkova-x1 5.19.0-38-generic #39~22.04.1-Ubuntu SMP
          PREEMPT_DYNAMIC Fri Mar 17 21:16:15 UTC 2 x86_64"
   },
   "score" : 46.221,
   "settings" : {
      "container_exec" : "singularity",
      "method" : "geometric_mean",
      "name" : "TestBenchmark",
      "reference_machine" : "CPU Intel(R) Xeon(R) CPU E5-2630 v3 @ 2.40GHz",
      "registry" : "docker://gitlab-registry.cern.ch/hep-benchmarks/hep-workloads",
      "repetitions" : 3,
      "replay" : false,
      "scaling" : 355
   },
   "status" : "success",
   "wl-scores" : {
      "atlas-gen-bmk" : {
         "gen" : 49.9976,
         "gen_ref" : 384
      }
   }
}
```


Listing C.1: JSON output after running hep-score locally

```
[ lzurkova: ~/public/hep-test/hepscore-dp ]$ condor_q
-- Schedd: schedulernode123.cern.ch : <123.345.789.012:3456?... @ 03/29/23 15:50:15
OWNER   BATCH_NAME    SUBMITTED  DONE   RUN    IDLE   HOLD  TOTAL JOB_IDS
lzurkova ID: 1234567 3/29 15:46     _      _      _      1      1 1234567.0
Total for query: 1 jobs; 0 completed, 0 removed, 0 idle, 0 running, 1 held, 0 suspended
Total for lzurkova: 1 jobs; 0 completed, 0 removed, 0 idle, 0 running, 1 held, 0 suspended
Total for all users: 3797 jobs; 934 completed, 0 removed, 1893 idle, 968 running, 2 held, 0 suspended

[ lzurkova: ~/public/hep-test/hepscore-dp ]$ condor_q -hold
-- Schedd: schedulernode123.cern.ch : <123.345.789.012:3456?... @ 03/29/23 15:51:34
 ID        OWNER         HELD_SINCE HOLD_REASON
1234567.0  lzurkova       3/29 15:47 Error from slot1_21@node123.cern.ch: STARTER at 666.123.456.789:0123
failed to send file(s) to <188.185.121.235:9618>: error reading from
/pool/condor/dir_13205/TestBenchmark.json: (errno 2) No such file or directory; SHADOW failed to receive
file(s) from <666.123.456.789:0123>

#[...]

[ lzurkova: ~/public/hep-test/hepscore-dp ]$ condor_q
-- Schedd: schedulernode123.cern.ch : <123.345.789.012:3456?... @ 03/29/23 17:00:07
OWNER   BATCH_NAME    SUBMITTED  DONE   RUN    IDLE  TOTAL JOB_IDS
lzurkova ID: 1234568 3/29 16:59     _      1      _      1 1234568.0
Total for query: 1 jobs; 0 completed, 0 removed, 0 idle, 1 running, 0 held, 0 suspended
Total for lzurkova: 1 jobs; 0 completed, 0 removed, 0 idle, 1 running, 0 held, 0 suspended
Total for all users: 8266 jobs; 934 completed, 0 removed, 6462 idle, 870 running, 0 held, 0 suspended
```

Listing C.2: HTCondor output

```
APEL-individual-job-message: v0.2
Site: RAL-LCG2
SubmitHost: ce01.ncg.ingrid.pt:2119/jobmanager-lcgsge-atlasgrid
LocalJobId: 31564872
LocalUserId: atlasprd019
GlobalUserName: /C=whatever/D=someDN
FQAN: /voname/Role=NULL/Capability=NULL
WallDuration: 234256
CpuDuration: 2345
Processors: 2
NodeCount: 2
StartTime: 1234567890
EndTime: 1234567899
MemoryReal: 1000
MemoryVirtual: 2000
ServiceLevelType: Si2k
ServiceLevel: 1000
%%
...another job record...
```

Listing C.3: APEL job record message format v0.2

```
APEL-individual-job-message: v0.4
Site: SOME-SITE
SubmitHost: host.ac.uk/cluster
LocalJobId: 9aef372d-e26f-42ce-7acb-5e1c479dc47f
LocalUserId: bob
GlobalUserName:/DC=ac/DC=uni/DC=/DC=vac
FQAN: /host.org/Role=NULL/Capability=NULL
WallDuration: 47248
CpuDuration: 46871
Processors: 1
InfrastructureDescription: APEL-CREAM-HTCONDOR
InfrastructureType: grid
StartTime: 1531869580
EndTime: 1623693622
ServiceLevel: {hepspec: 11.4, HEPscore23: 15.3}
%%
...another job record...
```

Listing C.4: APEL job record v0.4

```
APEL-summary-job-message: v0.3
Site: RAL-LCG2
Month: 3
Year: 2010
GlobalUserName: /C=whatever/D=someDN
VO: atlas
VOGroup: /atlas
VORole: Role=production
SubmitHost: test06.ral.ac.uk:8443/cream-pbs-GRID_ops
Infrastructure: grid
Processors: 1
NodeCount: 1
EarliestEndTime: 1267527463
LatestEndTime: 1269773863
WallDuration: 23425
CpuDuration: 2345
NormalisedWallDuration: 244435
NormalisedCpuDuration: 2500
NumberOfJobs: 100
%%
...another summary job record...
```

Listing C.5: APEL summary job record message format v0.3

```
APEL-normalised-summary-message: v0.4
Site: SOME-SITE
SubmitHost: host.ac.uk/cluster
Month: 9
Year: 2022
GlobalUserName:/DC=ac/DC=uni/DC=/DC=vac
WallDuration: 47248
CpuDuration: 46871
NormalisedWallDuration: {hepspec: 519728, hepscore23: 708720}
NormalisedCpuDuration: {hepspec: 515581, hepscore23: 703065}
Processors: 1
NumberofJobs: 3
InfrastructureType: grid
EarliestStartTime: 1531869580
LatestEndTime: 1623693622
%%
...another normalised summary record...
```

Listing C.6: APEL normalised summary record v0.4

```
APEL-summary-job-message: v0.4
Site: SOME-SITE
SubmitHost: host.ac.uk/cluster
Month: 9
Year: 2022
GlobalUserName:/DC=ac/DC=uni/DC=/DC=vac
WallDuration: 47248
CpuDuration: 46871
Processors: 1
NumberofJobs: 3
InfrastructureType: grid
EarliestStartTime: 1531869580
LatestEndTime: 1623693622
ServiceLevel: {hepspec: 11.4, HEPscore23: 15.3}
%%
...another summary job record...
```

Listing C.7: APEL summary job record v0.4

```
{
   "accountinggroup" : "<VO_group>.<user>",
   "args" : "<arguments>",
   #[...]
   "clusterid" : <cluster id>,
   "cmd" : "<command to run the job>",
   "err" : "<path to file to log errors>",
   "globaljobid" : "node123.cern.ch#<job id>",
   #[...]
   "match_cpus" : 1,
   "match_datacentre" : "meyrin",
   "match_exp_cpus" : "1",
   "match_exp_datacentre" : "meyrin",
   "match_exp_hepspec" : "377",
   "match_exp_hostlogicalcores" : "56.0",
   "match_exp_job_hepspec" : "6.732142857142857E+00",
   "match_hepspec" : 377,
   "match_totalcpus" : 56,
   #[...]
   "out" : "<path to file to log output>",
   "owner" : "<user>",
   #[...]
}
```

Listing C.8: Example of a formatted and ommited line from a file in the DATABUCKET

```
import os, gzip, json, copy, argparse
from datetime import datetime, timedelta
# [...] # I'm skipping the parsing part, but imagine something like
# dir day month year = '/home/lzurkova/s3fun-test/mocking/' 01 04 2023

dir_original = ('{}/original-{}').format(dir, day) # directory with original data
dir_mocked = ('{}/{}').format(dir, day) # directory with output (mocked) data
for filename in os.listdir(dir_original): # for each day from original data
    f = os.path.join(dir_original, filename)
    if os.path.isfile(f):
        with gzip.open(f,'rb') as f_gz_original:
            with gzip.open(os.path.join(dir_mocked, filename),'wt') as f_gz_mocked:
                for line in f_gz_original:
                    json_line = json.loads(line)
                    new_json_line = copy.deepcopy(json_line)
                    # copy all the lines with "hep" prefix
                    for key in json_line.keys():
                        if 'hep' in key:
                            value = json_line[key]
                            is_str = isinstance(value, str)
                            is_num = isinstance(value,(int, float, complex))
                            try:
                                # mock numerical values by multiplying
                                new_value = value*1.2 if is_num else float(value)*1.2
                            except ValueError:
                                # mock string values by replacing HEPSPEC by HEPSCORE
                                # [...]
                            # add the new key-value pair
                            new_json_line[key.replace('hepspec','hepscore')] =
                                new_value if not is_str else str(new_value)
                    # write the new line
                    f_gz_mocked.write(json.dumps(new_json_line))
                    f_gz_mocked.write("\n")
                print(("{} created").format(f_gz_mocked.name))
```

Listing C.9: mock.py file to mock the bucket data after you download the original data

```
#!/bin/bash
while getopts m:y:d: flag
do
case "${flag}" in
m) month=${OPTARG:=04};;
y) year=${OPTARG:=2023};;
d) day=${OPTARG:=01};;
esac
done
mkdir $day original-$day &&
echo "Getting data for $year-$month-$day" &&
s3cmd -c ~/.s3cfg get s3://REPORTSBUCKET/[...]/thooki/$year/$month/$day/* original
-$day &&
echo "Generating data for $year-$month-$day" &&
python3 mock.py --date $year-$month-$day &&
echo "Deleting temporary original data for $year-$month-$day from localhost" &&
rm -rf original-$day &&
echo "Uploading data for $year-$month-$day" &&
s3cmd -c ~/.s3cfg-test sync $day s3://TESTING-REPORTSBUCKET/[...]/thooki/$year/
$month/ &&
echo "Deleting generated data for $year-$month-$day from localhost" &&
rm -rf $day &&
echo "Data mocked for date $year-$month-$day"
```

Listing C.10: mock-day.sh file download original data, call mock.py, and upload mocked data to S3 a testing bucket

```
#!/bin/bash
while getopts s:e: flag
do
    case "${flag}" in
        s) start=${OPTARG:=2023-04-01};;
        e) end=${OPTARG:=2023-04-30};;
    esac
done

startdate=$(date -I -d "$start") || exit -1
enddate=$(date -I -d "$end")  || exit -1

d="$startdate"
while [ "$d" != "$enddate" ]; do
  read Y M D <<< ${d//[-]/ }
  ./mock-day.sh -m $M -d $D -y $Y
  d=$(date -I -d "$d + 1 day")
done
# last day
read Y M D <<< ${d//[-]/ }
./mock-day.sh -m $M -d $D -y $Y
```

Listing C.11: mock-month.sh file to call mock-day.sh several times

```
#[...]
df_aux = spark_utils.read_s3_data(self, start, end, bucket_path=self.condor_path)
df = (df_aux.withColumn('Date', Date(df_aux.completiondate, df_aux.enteredcurrentstatus))
           #[...]
           .withColumn('hepspec_cpu',
                    hepspec_cpu(sum([df_aux.remoteusercpu,
                                     df_aux.remotesyscpu]),
                                df_aux.match_hepspec,
                                df_aux.match_totalcpus))
           .withColumn('hepspec_wall',
                    hepspec_wall(df_aux.remotewallclocktime,
                                 df_aux.match_hepspec,
                                 df_aux.match_totalcpus,
                                 f.col('numcores')))
           #[...]
           )
#[...]
df = df.select(fields)
df = df.groupby(
    'Date',
    #[...]
    'numcores',
    'fqan').agg(
        #[...]
        f.sum('hepspec_cpu').alias('hs_cpu'),
        f.sum('hepspec_wall').alias('hs_wall'),
        #[...]
    )
grouped_values = {}
for i in data['Date'].unique():
    grouped_values[i] = [
        {   #[...]
            'hepspec_cpu': data['hs_cpu'][j],
            'hepspec_wall': data['hs_wall'][j],
            'numcores': data['numcores'][j],
            #[...]
        }
        for j in data[data['Date'] == i].index]
return grouped_values
```

Listing C.12: Ommited `_generate_daily_report` method from condor.py file from accounting-jobs project