ALMA MATER STUDIORUM · UNIVERSITÀ DI BOLOGNA

Scuola di Scienze
Dipartimento di Fisica e Astronomia
Corso di Laurea Magistrale in Fisica

Big Data Analytics towards Predictive Maintenance at the INFN-CNAF computing centre

Relatore:
Prof. Daniele Bonacorsi

Correlatore:
Dott.ssa Barbara Martelli

Presentata da:
Simone Rossi Tisbeni

Anno Accademico 2018/2019

Abstract

La Fisica delle Alte Energie (HEP) è da lungo tra i precursori nel gestire e processare enormi dataset scientifici e nell'operare alcuni tra i più grandi data centre per applicazioni scientifiche. HEP ha sviluppato una griglia computazionale (Grid) per il calcolo al Large Hadron Collider (LHC) del CERN di Ginevra, che attualmente coordina giornalmente le operazioni di calcolo su oltre 800k processori in 170 centri di calcolo e gestendo mezzo Exabyte di dati su disco distribuito in 5 continenti.

Nelle prossime fasi di LHC, soprattutto in vista di Run-4, il quantitativo di dati gestiti dai centri di calcolo aumenterà notevolmente. In questo contesto, la HEP Software Foundation ha redatto un Community White Paper (CWP) che indica il percorso da seguire nell'evoluzione del software moderno e dei modelli di calcolo in preparazione alla fase cosiddetta di High Luminosity di LHC. Questo lavoro ha individuato in tecniche di Big Data Analytics un enorme potenziale per affrontare le sfide future di HEP.

Uno degli sviluppi riguarda la cosiddetta Operation Intelligence, ovvero la ricerca di un aumento nel livello di automazione all'interno dei workflow. Questo genere di approcci potrebbe portare al passaggio da un sistema di manutenzione reattiva ad uno, più evoluto, di manutenzione predittiva o addirittura prescrittiva.

La tesi presenta il lavoro fatto in collaborazione con il centro di calcolo dell'INFN-CNAF per introdurre un sistema di ingestione, organizzazione e processing dei log del centro su una piattaforma di Big Data Analytics unificata, al fine di prototipizzare un modello di manutenzione predittiva per il centro. Questa tesi contribuisce a tale progetto con lo sviluppo di un algoritmo di clustering dei messaggi di log basato su misure di similarità tra campi testuali, per superare il limite connesso alla verbosità ed eterogeneità dei log raccolti dai vari servizi operativi 24/7 al centro.

Contents

List of Figures

List of Tables

Introduction

High Energy Physics (HEP) has been a driver in managing and processing enormous scientific datasets and the largest scale high throughput computing centers. Part of the HEP community developed a large scientific computing Grid for the Large Hadron Collider (LHC) at CERN in Geneva, that nowadays regularly overview the operations of 800k computer cores from over 170 sites in 42 countries, and half of an exabyte of disk storage distributed on 5 continents.

With the upcoming start of Run-3, and especially Run-4, the amount of data managed by the Tiers centres is expected to massively increase. In this view, the HEP Software Foundation — that published a Community White Paper providing a roadmap for the future research in software and computing in preparation for the HL-LHC — highlighted the enormous potential advantages that might come from applying Big Data Analytics techniques in the LHC physics endeavour.

One of the advancements discussed in that context is in Operations Intelligence in computing systems. A team effort has recently been promoted by a group of HEP computing teams, and one of the goals is to explore the logging data and apply data mining and analytics approaches to extract actionable insight and hence increase the level of automation in various systems. This effort, once applied to computing centers, may result into stepping out of Reactive maintenance to a Predictive or even Prescriptive approach.

This thesis will present the preliminary steps taken to introduce a unified Log ingestion and analytics platform at the INFN-CNAF Tier-1 across its services, with the purpose of exploring possible solutions for the development of a Predictive Maintenance model to detect and anticipate failures. This thesis explores the design and deployment of such an infrastructure as well as the development of proper algorithms to extract insight from log data.

This dissertation is divided in two parts: in *Part I* this work is contextualised, introducing the evolution in HEP, the structure of WLCG and discussing the concept of maintenance. *Part II*, instead, will focus on the exploratory work started at INFN-CNAF Tier-1 in the last year revolving around the analysis of the log data of the centre and their possible use to define a Predictive Maintenance model to apply in the future.

In particular, **Chapter 1** will discuss the upgrade schedule of LHC and the plan defined by the HSF to prepare for the future improvements in software and computing. The main concepts of Big Data, Analytics and Machine Learning will be briefly presented as well as an overview of the interest of LHC experiments in these fields.

Chapter 2 will focus on the WLC Grid, presenting the various tiers and later focusing on the INFN-CNAF Tier-1; the structure of the computing centre will be discussed, highlighting the main challenges as a major WLCG Tier-1.

Chapter 3 will close Part I, and will offer an overview on the different Maintenance classes, reviewing their characteristics and use cases; some examples of effective use of various maintenance models in data centres will be briefly presented, concluding with a description of most advanced Prescriptive Maintenance models.

Chapter 4 will open Part II by showcasing the types of data logged at INFN-CNAF; it will present the recent developments of a Log Repository solution to unify and collect all the log sources of the data centre; it will also include an overview about the log files from StoRM that will be analysed in the last chapter.

Chapter 5 focuses on the set up of a first Log ingestion and processing distributed platform to analyse the data stored in the Log Repository; the chapter also includes a brief description of the open source technical components used to design and deploy the platform.

Finally, **Chapter 6** concludes this work by presenting an algorithm for clustering in an unsupervised way the verbose event log files from StoRM, using text-based similarity measures. The Chapter will begin by discussing the previous work done during the CNAF Summer School program, it will then follow the development of the log clustering algorithm discussing the choice of similarity measure and the parameters that can influence the results. The developed procedure will be tested on different log files and the results compared to highlight their ability to detect possible anomalous behaviours. The concluding section in the chapter will focus on future developments of this work.

Part I

Big Data Analytics on WLCG Tiers

Chapter 1

HEP Computing evolution and Big Data Analytics

1.1 Computing in HEP in the next decades

In the long path that will bring the existing LHC systems towards new life in the High Luminosity regime, several technical upgrades are envisaged in the course of the next few years. The upgrade plans towards the HL-LHC are pictorially depicted in Figure 1.1.

The LHC is currently in its Long Shutdown 2 (LS2), scheduled to last until 2020 included, and in which experts are busy on the Phase-I upgrade of the LHC injectors, with aim of reaching the 14 TeV centre-of-mass energy and increasing the instantaneous luminosity by a factor of 3, for the start of the Run-3 data taking period. Subsequently, the Phase-II upgrade will take place between 2024 and 2026, and will eventually complete the increase in luminosity, yielding to HL-LHC with a peak luminosity of about $L = 7 \times 10^{34} cm^{-2} s^{-1}$. An integrated luminosity of $250 fb^{-1}$ per year will be made possible thanks to the installation of new elements including new focusing magnets and crab cavities. In the final luminosity scenario, the expected mean number of 200 interactions per bunch crossing will be reached.

Such a large and wide scope project requires large investments also in the R&D of software and computing systems needed to acquire, manage, process,

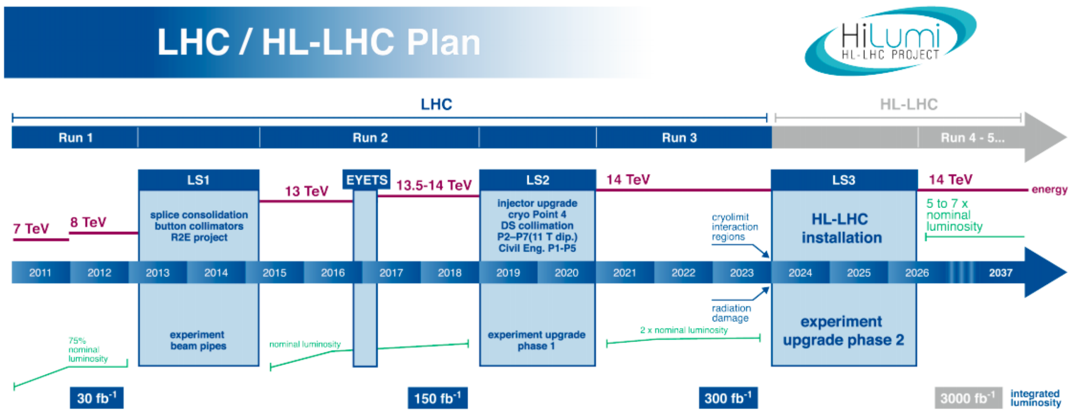


Figure 1.1: Upgrade schedule for the LHC/HL-LHC. Shutdown periods and data-taking periods, each with its intended instantaneous luminosity and centre-of-mass energy, are shown [1].

and analyse the huge amount of data that will be recorded. In fact, despite the programme evidently requires investments in detector hardware - both to build new facilities or experiments, and to upgrade existing ones - it is also true (and often mistakenly underestimated) that a commensurate financing effort is needed on software and computing in order to efficiently and successfully exploit the future facilities.

In this view, the HEP Software Foundation (HSF) [2] made a planning exercise in 2016 and published a Community White Paper (CWP) [3]. The document provides a roadmap for software and computing R&D in preparation for the HL-LHC and for other HEP experiments on a similar timescale: it clearly identifies areas of work and priorities in the investments. Quoting from [3], these priorities are:

- to achieve improvements in software efficiency, in its scalability and performance, and make use of all possible advances in CPU, storage and network technologies in order to cope with the challenges in front of the HEP community;
- to enable new approaches to software and computing that could radically extend the physics reach of the upgraded detectors;

- to ensure the long-term sustainability of the software through the entire lifetime of the HL-LHC;
- to ensure data (and knowledge) preservation beyond the specific lifetime of each individual experiment;
- to attract the required new expertise by offering appropriate career recognition to physicists that specialise in software development and/or computing systems design and development, and by effective training efforts able to target all software-level contributors in the community.

A description of the activities that started already and will bring the LHC community to stage that will ensure that software and computing will still be - as they have been in Run-1 and Run-2 - an enabling technology for HL-HLC, will bring us far out of the scope of this thesis. In this thesis, instead, a focus will be maintained on two specific aspects of this vast community effort, namely the effectiveness of Big Data Analytics and the raise of Machine Learning and Deep Learning.

1.2 Big Data Analytics

Big Data (BD) nowadays is a widespread term, defining almost any type of data analysis and statistics performed on large amount of data, but most of the time is used as a buzzword with almost no relation to the original meaning.

Gandomi et al. [4] follow the definition of BD back to sparse references in the mid-nineties through its widespread use since 2011, and suggests that as low as 4 different definitions are mainly used in the industry to understand BD.

The current widespread use of the term can be attributed to the promotional initiatives of IBM that also brought to the mass public attention the concept of the "five Vs". First defined by [5], the original definition described BD by just three characteristics that elaborate on the dimensions of challenges in data management.

The original report listed the three following Vs:

- *Volume* - Refers to the magnitude of data, reported in multiple terabytes or even petabytes. The definition of BD volumes is not strict since it strictly correlates to Variety: two datasets of the same size may require different magnitude of work if one of them is strictly numerical values, while the other are structured objects.

- *Variety* - Refers to the structural heterogeneity in a dataset. Structured data refers to the tabular data found in databases while unstructured data are for example raw images, text, numerical values etc. A characteristic of BD is the high level of variety between structured and unstructured data and their content.

- *Velocity* - Velocity refers to the rate at which data are generated and the speed at which it should be analysed and acted upon. Traditional data analysis systems are not able to handle fast flowing data streams, while BD technologies should be developed to ingest large amount of data quickly.

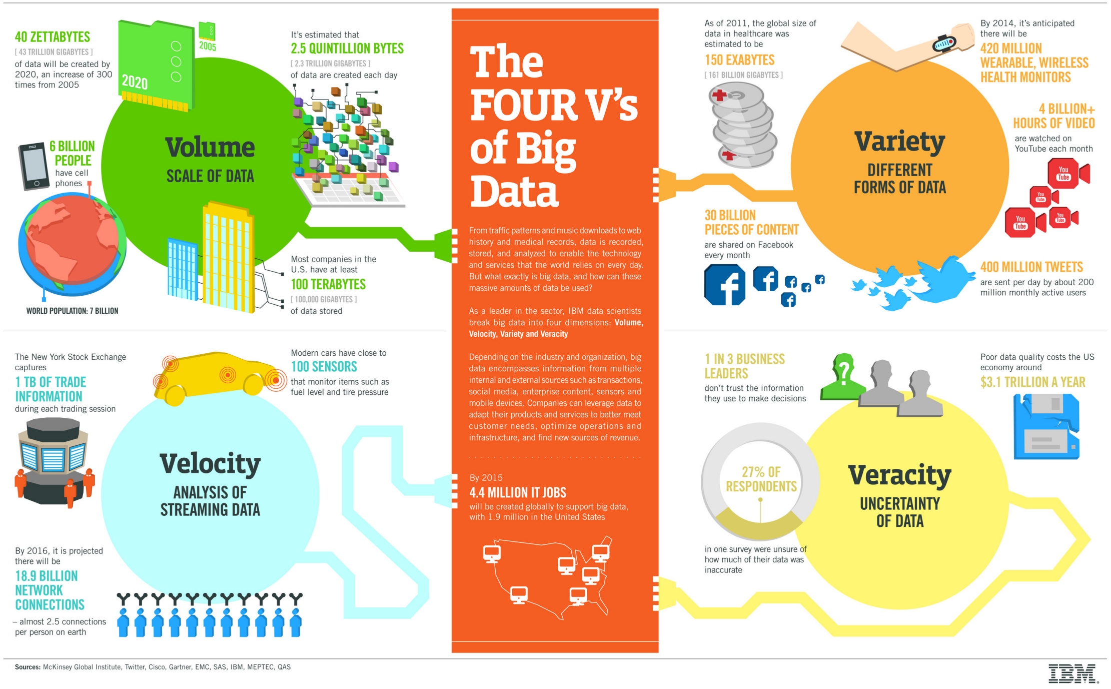
An additional term is commonly used, after IBM coined it in [6]:

- *Veracity* - Represents the unreliability inherent in some sources of data. Another big factor in defining BD is the need to deal with imprecise and uncertain data, often derived from human subjectivity and lack of standards. This situation is usually managed through the use of data mining and fuzzy matching.

More recently Oracle [7] introduced a new term as a defining attribute of BD:

- *Value* - Refers to the ability to extract value from the data. The assumption is that the value density in BD is low, since the information is spread into an higher volume of data.

Based on these definition, the data source from LHC respect the concept of Variety, Volume and Velocity; it is harder to find correspondence for the other Vs, the output data from the experiment is strictly structured and follow a defined



40 ZETTABYTES
[43 TRILLION GIGABYTES]
of data will be created by 2020, an increase of 300 times from 2005
2005
2020
Volume
SCALE OF DATA
It's estimated that
2.5 QUINTILLION BYTES
[2.3 TRILLION GIGABYTES]
of data are created each day
6 BILLION PEOPLE
have cell phones
WORLD POPULATION: 7 BILLION
Most companies in the U.S. have at least
100 TERABYTES
[100,000 GIGABYTES]
of data stored
The FOUR V's of Big Data
From traffic patterns and music downloads to web history and medical records, data is recorded, stored, and analyzed to enable the technology and services that the world relies on every day. But what exactly is big data, and how can these massive amounts of data be used?
As a leader in the sector, IBM data scientists break big data into four dimensions: Volume, Velocity, Variety and Veracity
Depending on the industry and organization, big data encompasses information from multiple internal and external sources such as transactions, social media, enterprise content, sensors and mobile devices. Companies can leverage data to adapt their products and services to better meet customer needs, optimize operations and infrastructure, and find new sources of revenue.
By 2015
4.4 MILLION IT JOBS
will be created globally to support big data, with 1.9 million in the United States
As of 2011, the global size of data in healthcare was estimated to be
150 EXABYTES
[161 BILLION GIGABYTES]
By 2014, it's anticipated there will be
420 MILLION WEARABLE, WIRELESS HEALTH MONITORS
Variety
DIFFERENT FORMS OF DATA
4 BILLION+ HOURS OF VIDEO
are watched on YouTube each month
30 BILLION PIECES OF CONTENT
are shared on Facebook every month
400 MILLION TWEETS
are sent per day by about 200 million monthly active users
The New York Stock Exchange captures
1 TB OF TRADE INFORMATION
during each trading session
Modern cars have close to
100 SENSORS
that monitor items such as fuel level and tire pressure
Velocity
ANALYSIS OF STREAMING DATA
By 2016, it is projected there will be
18.9 BILLION NETWORK CONNECTIONS
– almost 2.5 connections per person on earth
1 IN 3 BUSINESS LEADERS
don't trust the information they use to make decisions
Poor data quality costs the US economy around
$3.1 TRILLION A YEAR
27% OF RESPONDENTS
in one survey were unsure of how much of their data was inaccurate
Veracity
UNCERTAINTY OF DATA
Sources: McKinsey Global Institute, Twitter, Cisco, Gartner, EMC, SAS, IBM, MEPTEC, QAS
IBM.

Figure 1.2: Infographic break down of the 4 Vs, as defined by IBM data scientists: volume, variety, velocity and veracity. For higher resolution see [6].

pattern that is not subject to Veracity and most of the information is relevant with high Value density.

With the definition of Big data often comes the term Big Data Analytics (BDA). BDA encompasses the process of decision making and analysis that extract value from BD. The overall process can be broken down into the five stages shown in Figure 1.3. These five stages are divided into two processes: data management and analytics. Data management involves processes and supporting technologies to acquire the data, store it in a structured database, and prepare it for analysis by extracting the relevant features, cleaning the data and integrating the missing values. Analytics, on the other hand, refers to techniques used to analyse and interpret the information from BD; this process can either involve data mining techniques, classical statistical methods or the more recent Machine Learning approaches.

The application of BDA of interest in this thesis is related to LHC experiments at CERN. As it will be discussed and introduced in Chapter 2, LHC is a

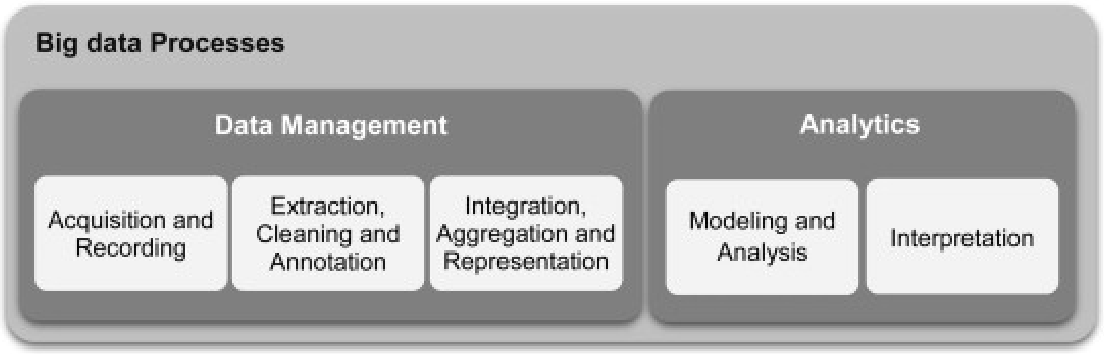

Big data Processes
Data Management
Acquisition and Recording
Extraction, Cleaning and Annotation
Integration, Aggregation and Representation
Analytics
Modeling and Analysis
Interpretation

Figure 1.3: Processes for extracting insights from big data as defined by [4].

major worldwide scientific enterprises, one of the largest and most complicated pieces of experimental apparatus that were ever constructed. At the same time, the LHC experiments are enormous, extremely complicated detectors that are able to extract physics information from proton-proton and heavy-ion collisions at exploiting the most advanced detecting techniques existing nowadays. In addition, the overall data handling and processing relies on a scientific computing Grid that now regularly operates $O(1M)$ processor cores and half of an exabyte (EB) [8] of disk storage located on 5 continents through hundreds of connected computing facilities, and orchestrating all this through a large set of different software services.

This extreme complexity, at various levels, is concretely able to eventually produce physics results via the capability to manage -in a wide sense - extremely large volumes of data.

The application of BD techniques in the LHC physics endeavour is hence crucial. At the same time, such application is different in face of a very diverse variety of existing LHC data. The data produced at LHC is of different types, e.g. original collisions data (coming from particle collisions), non-collisions data (e.g. alignment and calibration data), simulation data (coming from Monte Carlo techniques), derived data (coming from users' manipulation of original data through selections, skimming, reconstructions, etc), and various type of additional data like logging data (data coming from messages of running systems and servers) and meta-data (namely, data that contains information on the data itself).

In this thesis, the focus is on the application of BDA techniques on logging data coming from services running at the INFN-CNAF computing centre. While this might seem a limitation to a relatively small subset of LHC BD, a few numbers should suffice to clarify that actually this is indeed a subset, but its size is far from being small and easy to manage.

The CNAF centre hosts of the orders of $O(100)$ different services of various complexity. Some consist of just one machine, and some other of dozens or hundreds of different nodes. Each node/machine regularly prompts a log with most relevant operational messages. To give an example - as it will be described in more details in Chapter 4 - the machines that are currently logging and whose log data are being kept into the BD infrastructure described in this thesis are a total of 1130; each one archives several types of logs, on average around 17 different log files; the size of each log ranges from few MBs to tens of GB; all this logging happens daily: a rough count gives an average of 12 GBs of logs daily, for an estimate of 4 to 5 TB of data each year.

The work presented in Part 2 of this thesis will focus on the development of a BDA framework to ingest this type of data, and the development of an experimental algorithm for extracting information from the log files, this falls into the Data Management process of BDA described above; future development would likely involve a dive into Analytics tool, possible with the aid of Machine Learning approaches.

1.3 Machine Learning in HEP and at LHC

Machine learning (ML) is defined as the study of statistical models used to perform a specific task without using explicit instructions, but by inferring and relying on pattern in the data.

Machine Learning (ML) is a very rapidly evolving approach over recent years for building data-driven models, as it is able to characterise any class of data and extract information from it in the form of pattern extraction and eventually predictions. Despite it is being massively used so far, but in specific HEP applications, it is worth saying that ML is - in general - an approach to data that brings the potential to eventually change radically how HEP data is conceived, reduced and analysed.

The application of ML techniques in HEP are diverse in scope and objectives. A recent review and prospect of future usage of ML in HEP can be found in [9]. Some applications are aimed directly at qualitatively improving the physics reach of datasets. Other applications serve physics in a more indirect way, e.g. they allow a much more efficient use of all processing and storage resources, which are expensive and complex to set-up and run (in terms of hardware and people), thus indirectly extending the physics reach of experiments by effectively allowing the HEP community to achieve the same physics "throughput" by spending less money on computing.

As a first approximation, the majority of HEP community is not re-writing entirely all code to meet the ML paradigm, but it is actually building/upgrading its own, domain-specific applications on top of existing toolkits and ML algorithms and frameworks. The latter have not been developed in-house, and HEP faces such toolkits and frameworks as a customer community, exactly as the bioinformatics and genomics community, the earth sciences community, as well as commercial and finance sectors, etc. Examples of such tools in the ML domain are scikit-learn [10], Tensorflow [11], Keras [12], Pytorch [13], etc.

These tools have been developed by computer scientists, data scientists, and scientific software developers from outside the HEP world. A large part of the work that is being done in this context is to understand where HEP problems do not actually map well onto existing paradigms and solutions, and how such HEP problems might be recast into an abstract formulation that might be of more general interest, and thus meet possibly existing solutions.

1.4 The raise of operation intelligence

An advancement towards more AI in HEP experiments recently came on the floor in the discussions around what is being called "Operations Intelligence" (OpsInt).

In this Section a brief overview of this topic is given, and connections to the work presented in this thesis are discussed.

It is becoming evident that, in the near future, very large international scientific collaborations that operate in the High Energy Physics and Astro-particle Physics sectors will face unprecedented computing challenges in terms of quantity of data to manage, quality of software, requirements on computing operations, etc. Some concrete examples can be given.

The task of processing and storing very large (multi-PetaByte) datasets will require ad advanced, globally federated computing infrastructure of distributed resources. Despite the existing system has proven to be working, and to be mature enough to meet current data taking challenges and to deliver physics results in a timely manner (e.g. Run-1 and Run-2 at LHC), all post data taking analyses indicate a heavy crucial contribution from manual interventions by computing experts and shifters on duty. In other words, the existing infrastructures are able to work coherently and deliver experimental results as expected, but the cost in terms of operations of such heterogeneous infrastructures is far from being negligible.

At the same some, the computing operations of so many systems and services yielded a massive amount of logging information, and all this data has been archived on BD systems like ElasticSearch [14], Hadoop [15], no-SQL databases, etc. With respect to few years ago, when all this data was indeed archive but rarely - or never - accessed by anyone, this data has recently started to be explored by various experts in HEP computing teams with a shared goal: try to extract actionable insight from this massive amount of data. The initial goal is evidently to explore the data with data mining and analytics approaches, but the ultimate goal is to be able to understand the data well enough to model how experiments run computing operations, and eventually develop next-generation, intelligent, adaptive systems.

Such an ambitious plan, needless to say, can only be realized in subsequent steps. At the time of this thesis, there are various effort in this direction in different experiments, and all these efforts are recently steered into a unique “OpsInt” team, who kicked-off its activities in Spring 2019 under the umbrella of WLCG [16] and the HEP Software Foundation (HSF) [17]. The goal of the team is to explore the richness of information in all logging data and work on specific project that increase the level of automation in computing operations in various systems. Given the need of data-driven modelling, this work implies the use of adequate techniques, such as ML or even Deep Learning (DL), with solutions tailored to attack specific problems.

Examples of these efforts exist from the past, before this group was created. For example, ML models have been designed and applied to the prediction of data transfer efforts, with goal to predict a possible network congestion. Similar ML methods were applied for the prediction of data access patterns across a distributed storage systems, aiming at increasing the efficiency in resource exploitation. Time-series applications have been used to estimate the time needed for the completion of certain tasks, such as processing a certain number of events. Anomaly detection techniques have been employed to predict system failures. The act of recording and analysing shifters' actions started to be used to automatise tasks such as creating tickets (i.e. notifying issues with a problem tracking system) to computing centers' administrators, as well as suggesting possible solutions to typical issues that can be encountered.

In general, all these efforts aim at increasing the efficiency of HEP computing operations, i.e. the overall throughput of the experiments' distributed computing infrastructures (in other words, to perform "more computing at the same cost"). This effort will result in stepping from a Reactive approach to maintenance, were the intervention is performed after the rise of a problem, to a Predictive or possibly Prescriptive maintenance, were the human intervention is guided or even replaced by the Analytics framework. This argument will be further discussed in Section 3.1.

The following chapter will present an overview of the WLCG Project, with focus on the INFN-CNAF Tier-1 system, that will be the subject of our work in Part 2 of this thesis.

Chapter 2

LHC Computing Tiers

2.1 The WLCG project

The Worldwide LHC Computing Grid (WLCG) project is a global collaboration of more than 170 computing centres in 42 countries, linking up national and international Grids to build and maintain of a distributed computing infrastructure arranged in so-called "tiers" – giving near real-time access to LHC data [16].

The WLCG cooperates with several Grid projects such as the European Grid Infrastructure (EGI) [18] and Open Science Grid (OSG) [19]. WLCG provides seamless access to computing resources which include data storage capacity, processing power, sensors, visualisation tools and more. The main Grid building blocks the infrastructure is composed of are:

- the Computing Element (CE) service, that manages the user requests;
- the Worker Node (WN), the resource where the computation actually happens;
- the Storage Element (SE), that gives access to storage and data. Data can be stored on disks, for quick data access (e.g. end-user analysis), or on tapes, for long term storage and custody;
- the User Interface (UI), the front-end (FE) resource on which users interact with other Grid components and services;

Users can submit jobs which is turn are made of a complex set of atomic requests — such as a request for data in input, a request to apply algorithms and hence need for processing power, a request to write and save the outcome of such processing, etc — and this can happen from one of the many entry points into the system. The Grid systems check the credentials of the user, and search for available sites that can provide resources adequate to satisfy the user's requests, without any need for the user to have a deep technical knowledge on the available computing resources and systems that are serving her/his needs behind the scenes. For a omni-comprehensive overview in more details about Grid services, see e.g. [20]. Focus in this thesis is given the operational needs of grid computing centres, which are described in the next section.

2.2 Tiers in the Grid

The WLCG is composed of computing centres organised by levels, called "Tiers". Each Tier provides specific services, storage capacity, CPU power and network connectivity. This structure made by Tiers was formalised in the work done by the "Models of Networked Analysis at Regional Centres for LHC Experiments" team (MONARC), which produced a model still referred today as "the MONARC model" [21]. Following this model, the Tier levels are labelled with numbers from 0 to 3, with the number intended to indicate the level and complexity of services offered by the centre: namely, the lower the number the higher the level and quantity of services.

- Tier-0 is responsible for the safe-keeping of the RAW data (first copy), first pass reconstruction, distribution of RAW data and reconstruction output to the Tier-1s, and reprocessing of data during LHC down-times. The Tier-0 centre for LHC is located at the European Organisation for Nuclear Research (CERN) laboratory in Geneva, Switzerland. The role of the Tier-0 is to receive RAW data from detectors and store them onto tapes for safe-keeping (first copy), as well as perform first pass reconstruction. The Tier-0 also distributes the RAW data and the reconstructed output to Tier-1s.

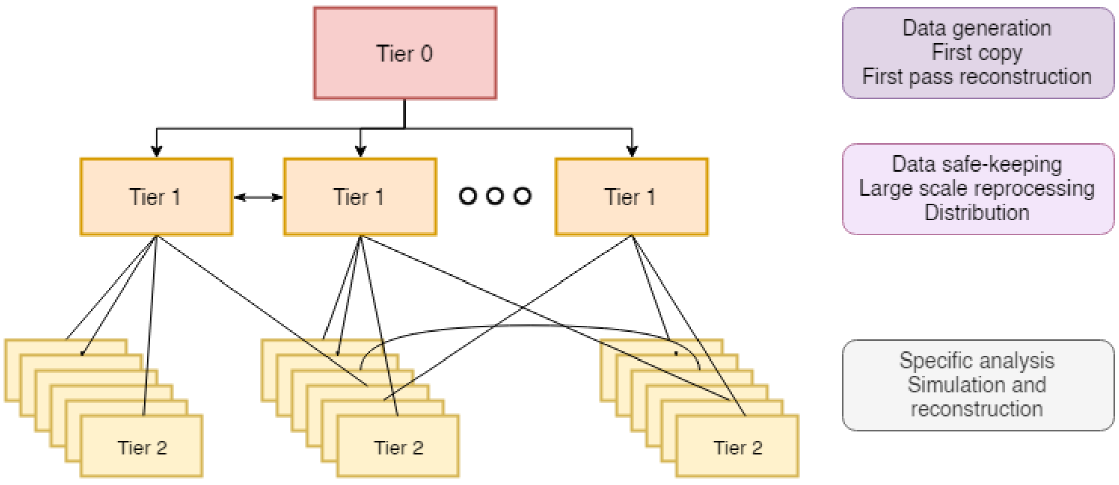


Figure 2.1: Schematic representation of the Tier levels, as defined by the MONARC model [21]. See text for details.

The integrity and availability of the data is guaranteed by a redundancy mechanism that stores the data (at least) in two copies: one at CERN (the so-called “cold” copy) and (at least) one distributed to Tier-1s (the so-called “hot” copy).

To securely transfer data and ensure high performance, a dedicate high-performance network infrastructure based on fiber optic - called LHCOPN [22] has been designed and deployed to connect the Tier-0 to all the Tier-1s and later expanded to also cover all the Tier-1 to Tier-1 connection, adding a crucial redundancy to the overall system.

Despite all the data from LHC passes through this central hub, the Tier-0 itself provides a relatively low fraction of the total Grid computing capacity, of the order of less than 20%.

- Tier-1 centers are thirteen large computer centres, spread all around the globe, with sufficient storage capacity to store LHC data round-the-clock support for the Grid. They are responsible for storing a proportional share of RAW and reconstructed data, the “hot copies”, on both disk caches (for high performing access) and tapes (for persistent archive). They manage large-scale reprocessing and safe-keeping of corresponding output and distribution of data to Tier-2s and safe-keeping of a share of simulated data produced at these Tier-2s.

- Tier-2s are typically universities and other scientific Institutes, which can store sufficient data and provide adequate computing power for specific analysis tasks. There are currently around 160 Tier-2 sites in WLCG, covering most of the globe capable of handling analysis requirements and a proportional share of simulated event production and reconstruction.
- Tier-3 centers typically refer to local computing resources through which individual scientists access the WLCG facilities. Unlike Tier-2 centres, Tier-3 sites do not sign any Memorandum of Understanding (MoU) with WLCG, meaning there is no formal engagement between WLCG and Tier-3 resources. This implies no check on their availability is regularly made, and resources should not be relied upon on a stable basis — despite some can be extremely important for the needs of local communities of analysts.

A schematic representation of the Tier levels, as defined by the MONARC model, is shown in Figure 2.1.

2.3 The INFN-CNAF Tier-1

The National Institute for Nuclear Physics (INFN) is the Italian research agency dedicated to the study of nuclear, particle and antiparticle physics [23]. INFN carries out research activities in many facilities across Italy, including the National Centre for Research and Development in Information Technology (INFN-CNAF), based in Bologna [24].

CNAF is one of the central actors in the overall INFN strategy on computing in Italy. Since the creation of the WLCG distributed computing system, CNAF has been deeply involved in the development of its middleware and in taking its share of the load in managing part of this efficient and complex infrastructure. Since 2003, CNAF also hosts the Italian Tier-1 data centre, providing the resources, support and services needed for data storage, data distribution, data processing and data analysis, including a high load of Monte Carlo simulation, for all 4 LHC experiments, but also for more than 20 other experiment in both experiment and theoretical high-energy physics, astroparticle physics, geophysics, and beyond [24].

The Tier-1 operations inside CNAF are structured in few major groups: the Farming group, the Data Management group, and the Network group will be described below; the Facility Management group manages the general facilities of the centre, and the Experiment Support group provides support to all researchers while exploiting the deployed infrastructure.

2.3.1 Farming

The bulk of the computing resources (WNs, UIs and grid services) are operated by the Farming group.

The computing farm infrastructure acts as the service underlying the workload management system of all experiments connected with the processing resources at CNAF. The jobs running on the farm have direct access to files stored at CNAF, with the disk storage system organised by several file systems directly mounted on the WNs. Each experiment has at least a dedicated queue, and the computing resources are centrally managed by a unique batch system.

On average, a total of the order of 100k batch jobs are executed every day at CNAF, with resources available 24/7.

2.3.2 Storage

Disk and tape storage services, together with the data transfer services - that constitute a middleware layer on top of them - are operated by the Data storage group.

Magnetic tapes are used as the main long-term storage medium, with more than 50 million experiment files stored as of today. The high-speed disks are mostly SATA, a good-performance/low-cost solution, with SSDs used for applications with higher requirements in I/O access.

The storage is managed by StoRM [25], a Storage Resource Manager solution developed at CNAF, based on a home-made integration of IBM General Parallel File System (GPFS) for disk access with the IBM Tivoli Storage Manager (TSM) for tapes. More details on StoRM are available in Section 4.3.

The amount of data currently stored and being processed at CNAF is in the order of tens of Petabyte, with a breakdown - at the time of this thesis - of about 23 PB of data on disk, and 48 PB of data on tape. This is expected to grow massively in the following years[24].

2.3.3 Networking

All the network connectivity and network security related activities are in charge of the Network group of the Tier-1.

Network resources must be distinguished between Wide Area Network (WAN) and Local Area Network (LAN): CNAF is connected to the WAN with links at various level of performances and redundancies, and the WAN connectivity is provided by GARR (the Italian National Research and Educational Network) [26]; Ethernet is the main technology used for the local interconnection LAN, designed and continuously updated to allow all the data to be efficiently distributed internally to the site.

The needs for the processing of huge amount of data with a great number of CPUs led to recent effort to provide a data access bandwidth of the order of hundreds Gigabit per second, and to the implementation of the Datacenter Interconnection with CINECA at a rate of 400 Gbps [24].

A more in-depth description of the WLCG Tiers in general, or of the INFN-CNAF Tier-1 centre in particular, from either the technical set-up point of view or the experiment applications point of view, would go beyond the needs and scope of this thesis. In fact, the brief description offered in this chapter is sufficient for the second part of this thesis, which will focus on the design and implementation of a new Big Data Analytics infrastructure inside the INFN-CNAF Tier-1 computing centre.

The work described in this thesis is part of a major INFN-CNAF project, namely a first coherent and Tier-1 wide effort towards equipping the centre with a modern analytics framework that would be able to efficiently process the vast (and increasing) amount of data coming from services operated at a Tier-1 centre, with the purpose of extracting insight as of how to monitor, support and maintain the computing centre in the most effective way.

Chapter 3

Overview of maintenance strategies

The Oxford dictionary define Maintain as *the act of keeping something in good condition by checking or repairing it regularly.* Webster extends this definition to *the act of preserving it from failure and decline.*

In trying to define Maintenance, one needs to answer the following questions: What defines the good condition of a system? When is a system efficiency declining or failing? Moubray [27] answers these question by determining that every system is put into service because someone wants it to do something. As such, he proposes the following definition for maintenance:

> *Ensuring that physical assets continue to do what their users wants them to do.*

This user-centred view stands at the basis of the industry standard definitions of maintenance processes. Over time, multiple different terms have been used to categorised the various practices, depending on their cost, intervention time, criticality and resources needed.

3.1 Maintenance classification

While it is easy to qualitatively distinguish maintenance processes based on these factors, standard definitions have never been established and the incorrect use of adopted names and neologisms in literature can be a barrier for the definition and understanding of clear concepts when deciding on the most appropriate maintenance type to apply for a system.

Trojan et al. [28] reviewed multiple traditional classifications and presented a proposal for dividing types of maintenance in four distinct groups: Reactive (or Corrective) maintenance, Preventive (or Proacting) maintenance, Predicting maintenance and Advanced maintenance. The Reactive class includes every maintenance model in which the maintenance procedure follows the failure of the system. The intermediate classes operate based on preventive detection of failure, the difference between the two being determining the course of action based on expected behaviour and lifetime for the former, or by constantly monitoring the behaviour of the system for the latter. The Advanced class includes models that requires higher planning and investments. These include Reliability-centred maintenance processes, in which every single aspect of a product line is monitored to ensure a high reliability, and prescriptive models, where the system adapts automatically to avoid stress-related failures and reduces the need of manual intervention.

These classes of maintenance processes show a sort of chronological evolution, where the complexity of the model increases with the growth of the industry expectations and needs (Figure 3.1). But while the efficiency of the latest models is indisputable, the older reactive models are still the most widely used.

The following sections review the different classes listed so far, and present an example of the more advanced prescriptive models and their possible future applications in fields such as OpsInt (See Section 1.4).

3.2 Reactive Maintenance

Reactive Maintenance (RM) is a type of maintenance process that occurs after failure or loss of device performance. In this operation, intervention is performed

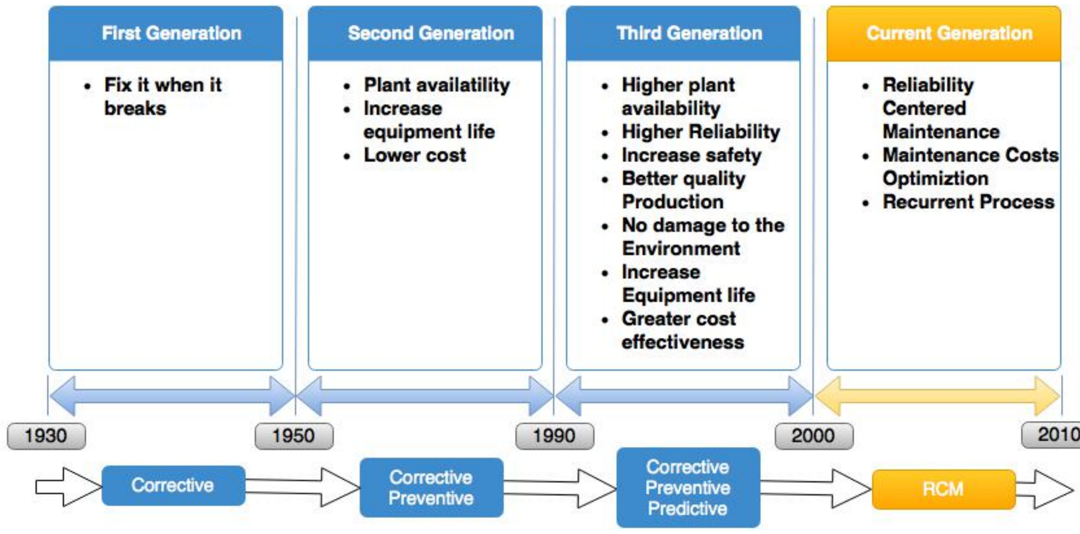

First Generation
• Fix it when it breaks
Second Generation
• Plant availatility
• Increase equipment life
• Lower cost
Third Generation
• Higher plant availability
• Higher Reliability
• Increase safety
• Better quality Production
• No damage to the Environment
• Increase Equipment life
• Greater cost effectiveness
Current Generation
• Reliability Centered Maintenance
• Maintenance Costs Optimiztion
• Recurrent Process
1930
1950
1990
2000
2010
Corrective
Corrective Preventive
Corrective Preventive Predictive
RCM

Figure 3.1: Timeline of maintenance generations [28]. See text for details.

as needed and reacts to emergencies as they arise; it is often defined as a "fire fighting" strategy [29].

When a breakdown occurs, the operator must move fast and make the decisions necessary to *minimise* the effect of the breakdown on the production system: it is worth underlining that "minimise" here is indeed the keyword. In fact, in many cases, temporary repairs may be made so that the facility can return to function quickly and be repaired — correctly, or permanently — at a later time.

The cost of RM is virtually zero, and is based on a loss control policy that minimises downtime. *If* a repair is needed, the cost of maintenance — being either an actual cost of repair or replacement, or a cost in manpower — is accounted for, otherwise the maintenance cost is nonexistent. The downside being that, in case of a serious breakdown or more time-consuming repairs, the system is more prone to unplanned downtimes, whose duration is completely unpredictable.

Another disadvantage to RM is the inefficient use of staff resources, that often need to be redirected to maintenance upon needs, unexpectedly stopping standard production activity.

Even with all the apparent disadvantages, RM is still the most recurrent maintenance policy applied in industries [30] and data centers [31].

3.3 Preventive Maintenance

The Preventive Maintenance (PM) class includes all the maintenance model based on recurrent preemptive maintenance operations that are performed on a system to improve its reliability, reduce the chance of breakdowns and in general increase its lifetime expectancy.

On a PM model, the health of a system is checked on a schedule, with the objective of having the equipment make it from one service to the next without any failures. This services may include partial or complete overhauls or replacements at specific periods [29].

3.3.1 Planned Preventive Maintenance

When the schedule of the preventive maintenance services is based only on the age of the system it is commonly referred as Planned Preventive Maintenance (PPM).

PPM is pre-planned, at fixed time intervals, on specific dates, or based on the equipment running hours. The scheduled service ensures that the system is operational and fixes the faulty equipment to avoid breakdowns and failures.

The cost of the single maintenance service is often low but, being based on a fixed timetable, often time involves unnecessarily maintenance services on perfectly functioning systems.

3.3.2 Condition-based Maintenance

A slight improvement over standard Planned maintenance models is Condition-based Maintenance (CBM). It is the first maintenance model in this brief review in which Data Analytics is actually used in support of maintenance operations [27].

The principle of operation of CBM is to use real-time data to observe the health of the system, in a process known as Condition Monitoring. While still often based on a pre-planned schedule, in CBM maintenance is only performed if the Condition Monitoring system deems it necessary.

CBM models have great advantages over standard PPM models: the maintenance cost is lower, mainly because of the lower number of unnecessary maintenance services performed. This also reduces downtimes, improving the reliability of the system [29].

On the other side, this type of models brings several challenges. In order to be reliable, the system requires monitoring instrumentation on all the equipment that needs to be maintained, adding a high initial cost. Also, the raw data from the monitoring system may not be directly correlated to the health of the equipment, requiring a prior analysis of the data. The CBM installation, moreover, adds itself to the number of parts that need to be maintained.

3.4 Predictive Maintenance

A step forward from CBM, Predictive Maintenance (PdM) methods utilise measurements on the actual equipment in combination with measurement of process performance, measured by other devices, to trigger equipment maintenance. PdM uses this data in conjunction with analysed historical trends to evaluate the system health and predict a breakdown before it happens [32].

The "predictive" component of PdM stems from the goal of predicting the future trend of the system's condition. This approach uses principles of statistical process control to determine at what point in the future maintenance activities would be appropriate. The ultimate goal of the approach is to perform maintenance at a scheduled point in time only when the maintenance activity is most cost-effective and before the system loses performance, going below an acceptable threshold.

PdM can be *offline* when the monitoring is periodic, or *online* when the monitoring is performed continuously from a stream of data.

The PdM approach is based on fast analysis of large amount of data, as such it needs accurate and complete information for optimal decision-making. As big data technology is getting mature, systematic analysis of massive streaming data have been developed for exploring hidden patterns.

The literature on PdM techniques is vast and varied. These models are increasingly widely used in industry and science. Some of the most recent works on PdM focus on large-scale data centres, and highlight the most effective techniques applied. For example, [32] and [33] use disk sensors' data to predict the likelihood of disks failures in large-size data centres; [34] applies time series based PdM to predict the future workload of the Wikimedia Grid data centre; [35] uses a large openly available industrial data centre's dataset to perform prediction on catastrophic data centre's failure.

3.4.1 Log-based Predictive Maintenance

A more involved approach to monitoring the system status used in the field of computer management and intelligence, is the log file analysis.

Modern equipment is usually operated via software applications. These applications produce logs while they operate. Such logs reflect the developers' original ideas about what are the valuable events to report out, and hence contain informational/warning/error messages, but also internal states, or exceptions. By analysing a service's log files, one can trace back how a system has performed. Mining such rich information may result in a precious tool to detect symptoms of issues in advance with respect to their occurrence. Log datasets can be large and have the advantage that they can be collected for PdM techniques without any special need to implement specific monitoring equipment or sensors.

It must be underlined, though, that since logs are mainly used for debugging purposes, they rarely contain explicit information that could be used for failure prediction. Moreover, they contain heterogeneous data including symbolic sequence, numeric time series, categorical variables and unstructured text and, in general, can be very verbose (and hence result in large size files), posing some computational challenges to the log analysis efforts, especially if a relatively low latency requirements do apply [36]. Another point to consider is that, due to their purpose, log files are usually written in a human-readable form, that includes redundant data and descriptive text that may not be needed for any PdM. To make use of this data, as a consequence, a first step is usually to interpret the logs by filtering out all the irrelevant text that acts as noise with respect to the predictive features that one wants to extract.

Some recent examples of Log-based PdM including the following approaches: the use of simple parsing techniques to extract data from event log to predict failures in medical equipment [36]; the use of clustering techniques to assign different IDs to the log messages to predict failures in telecommunication data centres [37]; the use of dictionaries to classify system event logs in order to predict switch failures in the network of a data centre [38].

A well-designed PdM program can minimise unscheduled breakdowns of all equipment in the system and ensure that the system performance is always kept above an acceptable threshold. The method can also identify problems before they become serious. If the problem is detected early, major repairs can usually be prevented. Note however that PdM is not a substitute for the more traditional Planned maintenance routines. It is, instead, a valuable addition to a comprehensive, total maintenance program.

As for CBM, the downside of the PdM approach is the cost of installation, the necessity of performing deep analysis on the data, and often, for bigger system, the implementation of a scalable big data management infrastructure to store not only the most recent measurements from the equipment but also all the relevant historical data.

It should be noted that the investment in PdM is made earlier than when those costs would be incurred if equipment were run until failure. Even though the cost will be incurred earlier, and may even be larger than the cost for RM methods, the benefits in terms of system availability should be substantially greater from doing preventive tasks. These solutions, although, are not usually affordable for small centres: often they require specialists to have PdM processes in place, and to be able to understand and digest the vast amount of logging data [39].

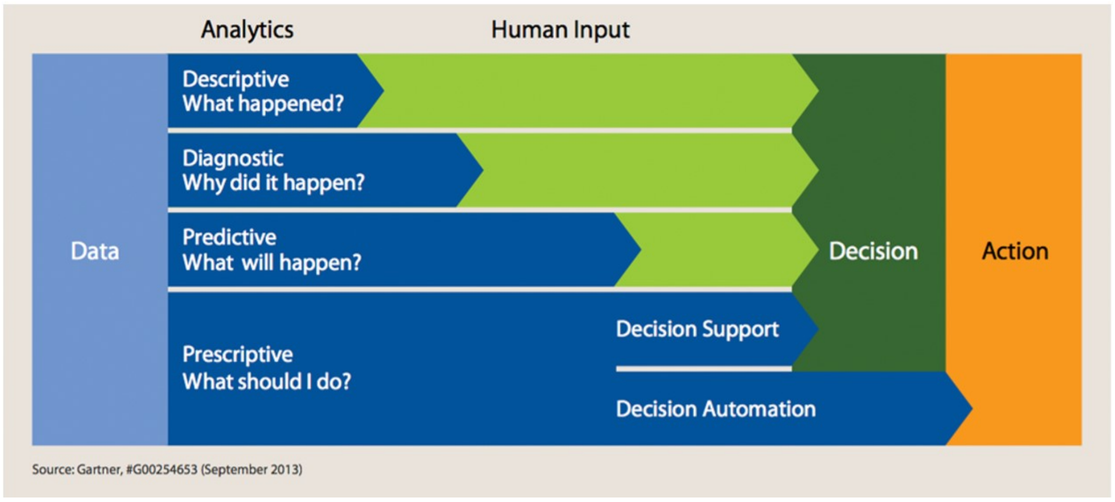


Figure 3.2: Stage model for characterising data analytics use cases [40].

3.5 Prescriptive Maintenance

A further advancement over PdM is the so called Prescriptive Maintenance (PsM). This maintenance methods fall under the Advanced maintenance class presented in Section 3.1. AI-enabled PsM is unique in that instead of just predicting impending failure it strives to produce outcome-focused recommendations for operations and maintenance from analytics.

When a change in the system occurs, PsM will not only show what and when a failure is going to happen, but why it is happening. Taking it one step further, PsM will take the analysis and determine different options and the potential outcomes, suggesting a plan of action to the support team. Leading up to the maintenance action, the data will be continuously analysed, constantly adjusting the potential outcomes and making revised recommendations, improving the accuracy of the results. Once the maintenance activity is completed, the analytics engine will continue to monitor the system and determine if the maintenance activity was effective.

PsM systems are able not only to make recommendations (Decision support) but also to act on those recommendations (Decision automation). See Figure 3.2.

For example, a PdM solution might recommend that a system get checked based on analysis of log data and measurements, but a PsM system would kick off a work order to support staff based on this information and oversee the entire maintenance workflow. Systems like these must be able to understand the causes of an event, and differentiate between the services normally applied in response to a given signal. This technology makes large use of big data analytics techniques and machine learning algorithms.

It is straightforwards that, in principle, PsM can be a great improvement over a standard maintenance system, as it would greatly improve the reliability and reduce the human intervention for maintenance, which in turn leads to reduced costs of operation. On the other side, the disadvantage is - as discussed for previously mentioned methods - the high cost of implementation, in terms of needed equipment as well as in the necessary knowledge to put the process in place and keep it effective.

With respect to other approaches, the literature on PsM is relatively scarce, as it is a fairly recent advancement in maintenance technologies, with industry being the major stakeholder and actor. To quote a couple of examples, a proposal on how to implement a PsM strategy in manufacturing companies can be found in [41]; a commercial PsM model acting as a 'maintenance control centre', which presents an application in the automotive sector, can be found in [42].

Part II

Towards predictive maintenance at CNAF

Chapter 4

Logging data at CNAF

As briefly discussed in Chapter 2, the INFN-CNAF Tier-1 manages and stores tens of Petabytes of data from various scientific experiments. Among all this data, only a small percentage is produced by the data centre in the form of service logs and monitoring data. Still, the amount of data produced is sizeable and requires a proper infrastructure to be efficiently and effectively managed and analysed.

4.1 Metrics and Logs

Most of machines in the centre are monitored to extract metrics of their operation (e.g. the percentage of memory in use, the number or synchronous and asynchronous requests, the average load, etc). All these metrics are stored in a relational database.

In addition to these metrics, every machine and service running on them prompts the writing of one or multiple log files containing various sort of information (e.g. the type of requests send and their status, the shell command launched, debug information on the condition of the machine, etc).

All these log files present heterogeneous structures, often non standard and specific for the machine they originate from. Frequently, even when the type of log files is the same across different machines, its generation is different based on which team developed that logger.

4.1.1 Rsyslog

The system used to collect the log files from every machine is rsyslog.

The Rocket-fast system for Log processing (rsyslog) is an open source project used for forwarding log messages in a network [43]. The service is based on syslog which is a common standard for messaging logging.

The standard consists in adding to the log message information such as the hostname, the timestamp of the logging and in some case a tag corresponding to the severity level (i.e. INFO, WARN, ERROR etc.).

The log messages collected by the system are usually directed to a text file in the machine that produced the log.

4.1.2 InfluxDB

Some information not included in log messages are acquired by a monitoring system, that collects various metrics from the machines in the centre and stores them into a database called InfluxDB.

InfluxDB is a time-series database used to store large amount of timestamped data [44]. The data collected from the monitoring system is stored into different tables in the database, one for each metric acquired.

A database table is indexed by the timestamps and contains a column corresponding to the hostname of the machine to which that measure corresponds and the value of the measure. Various tags are used to group the different machine corresponding to the same service or experiment.

The system grants the use of different retention policies according to the lifetime of the data — i.e. the measurements are logged every 5 minutes for a week, with the 'one_week' retention policy; every 15 minutes for one month, with the 'one_month' retention policy, etc.

The data collected this way is then visualised through the use of a Grafana web interface. Grafana is an open source platform used to visualise InfluxDB data (and data from other sources) into dynamic dashboards [45]. The dynamicity of the dashboard allows the user to filter the data coming from the InfluxDB databases without the need of manually typing queries.

4.2 Log Repository

In April 2019, the CNAF management, started exploring the development of a single Log repository to be used in the centre. The purpose of this repository is to unify the collection of log files and store them into a single storage system both for safe-keeping and to ease the collection and analysis effort in view of future improvement to the maintenance process of the centre.

The system developed so far makes use of rsyslog to redirect the flow of logs from every machine in the centre to the same repository. The repository is structured in the way shown in Table 4.1.

```
├── host 1
│   └── year
│       └── month
│           ├── day 1
│           │   ├── log file 1
│           │   └── log file 2
│           └── day 2
│               └── log file 3
└── host 2
    └── year
        ├── month 1
        │   └── day
        │       └── log file 4
        └── month 2
            └── day
                └── log file 5
```

Table 4.1: Sample directory structure for the Log Repository. Details in the text.

This platform allows for easier retrieval of the log files. The access to the repository is allowed via Network File System (NFS), a distributed file system that allows to access files over network as if they were in local storage. The NFS is an open standard defined in a Request For Comments (RFC) [46].

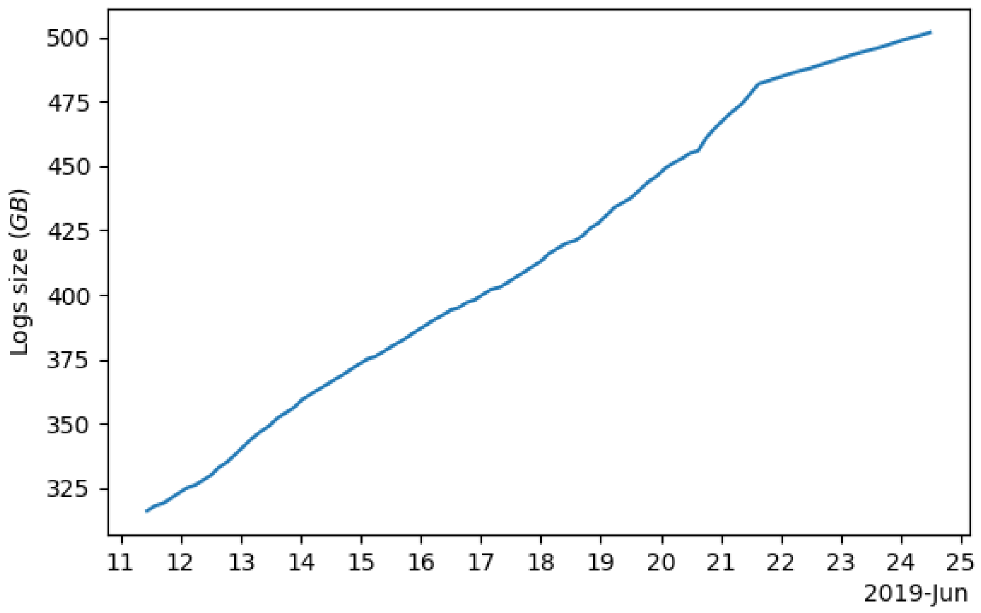


Figure 4.1: Total size of the log files stored into the Log Repository as a function of time.

The number of log files redirected into the Log Repository increased gradually in the last month, while more and more machines were configured to start logging into the repository. At the time of this thesis, the machines that are logging into the Log Repository are 1133 and include all of the Storage hosts, all of the Network hosts and all of the Farm hosts with every Worker Node included. Each machine stores daily an average of 16 log files, for an average of 14 GB and a maximum of 19 GB of data daily. Figure 4.1 shows the amount of data stored in the Log Repository in the last two weeks. Every day more than 18000 log files are stored; the number of log files in the repository at the time of recording was 577585, for a total of 502 GB of data.

Few characteristic log files from StoRM will be briefly presented in the following section. These log types are relevant since they are the type of log files studied in the last part of this thesis.

4.3 StoRM logs

StoRM is a storage manager service for generic disk-based storage systems [25]. StoRM is the storage management solution adopted by the INFN-CNAF Tier-1, and it has been developed in the context of WLCG computational Grid framework with the specific aim of providing high performing parallel file systems to manage the resource distribution for the storage of data in disks. StoRM has a multilayer architecture made by two stateless components, called Frontend (FE) and Backend (BE), and one database use to store the SRM requests and its metadata. A simple StoRM architecture is shown in Figure 4.2.

StoRM supports the Storage Resource Manager (SRM) protocol dividing the operation int synchronous and asynchronous requests. The asynchronous requests include srmPrepareToPut, srmPrepareToGet, srmBringOnLine, srmCopy; while synchronous requests are Namespace operations (srmLs, srmMkdir,etc.), Discovery operation (srmPing) and space operations (srmReserveSpace, srmGetSpaceMetadata, etc.).

The logging activity represents an important functionality of both BE and FE StoRM components, and of the services linked to them. There are different kind of files in which specific information is stored. A summary of the roles of the two main components is discussed below, with example of their log files.

4.3.1 Frontend server

The FE provides the SRM web service interface available to the user, manages user credentials and authentication, stores SRM requests data into a database, retrieves the status of ongoing requests, and interacts with the BE.

Frontend services logs into two different files: *storm-frontend-server.log* and *monitoring.log*. The former has been the first log type analysed throughout Section 6.2; a sample line of the log file is shown in Table 4.2.

When a new SRM request is managed, the FE logs a new line on the file. A log line contains information on the type of request, the user, the success of failure of the request and the token that links it to the BE process.

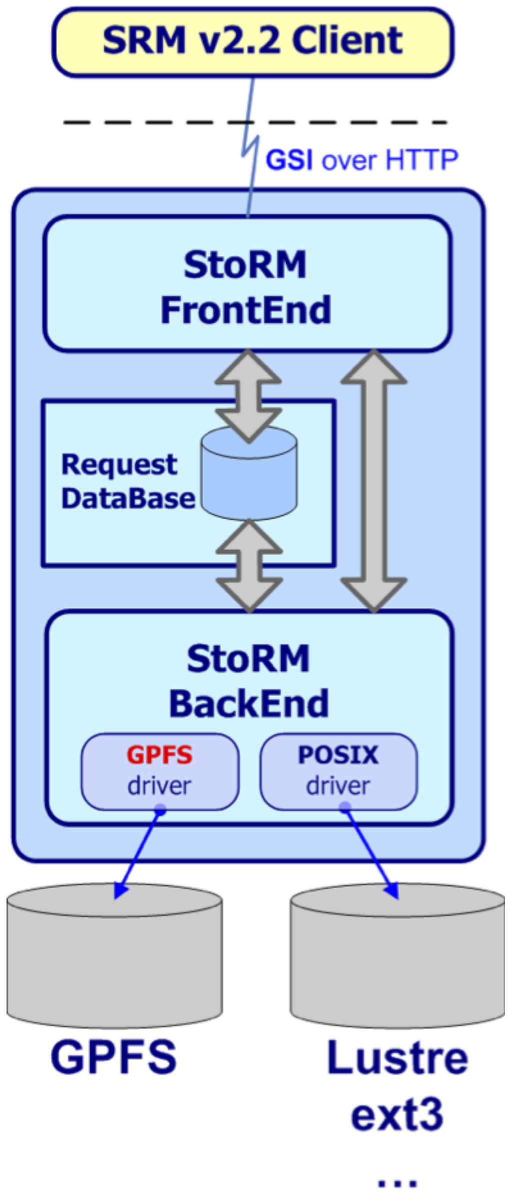


Figure 4.2: Simple StoRM Service Architecture schema with one Backend and one Frontend.

```
12/07 03:16:09.282 Thread 28 - INFO [0c79ddfe-07cd-48e8-
8547-53d9ca38d154]: Result for request 'PTP' is
'SRM_REQUEST_QUEUED'. Produced request token: 'cae6a710
-d292-4404-9bdf-c7a67dad4759'
```

Table 4.2: Sample log line from *storm-frontend-server.log* from 8/12/2018.

```
00:00:00.144 - INFO [xmlrpc-5916] - srmRm: user ¡/DC=ch/DC=
cern/OU=Organic Units/OU=Users/CN=atlpilo1/CN=614260/CN=
Robot: ATLAS Pilot1¿ operation on [SURL: srm://storm-fe
.cr.cnaf.infn.it/atlas/atlasdatadisk/rucio/mc15_13TeV/bf/
b0/log.16290046._051554.job.log.tgz.1.rucio.upload]
failed with: [status: SRM_INVALID_PATH: File does not exist]
```

Table 4.3: Sample of log line from *storm-backend-server.log* from 8/12/2018.

```
00:00:17.827-synch.ls [(count=10669590, m1_rate=329.633,
m5_rate=362.073, m15_rate=501.721) (max=613.200, min
=9.381, mean=41.350, p95=171.993, p99=378.712) ]
duration_units=milliseconds , rate_units=events/minute
```

Table 4.4: Sample of log line from *storm-backend-metrics.log* from the 8/12/2018.

4.3.2 Backend server

The BE is the core of StoRM service since it executes all synchronous and asynchronous SRM functionalities. It processes the SRM requests managing files and space, it enforces authorisation permissions and it can interact with other Grid services.

Table 4.3 shows a sample line of the *storm-backend-server.log* file.

The BE server log files provide information on the execution process of all SRM requests, error or warning. BE logging is based on the logback framework. Logback provides a way to set the level of verbosity depending on the use case. The level supported are FATAL, ERROR, INFO, WARN, DEBUG. At the INFO level, the BE logs for each SRM operation who has requested the operation (DN), on which files (SURLs) and with which result.

Backend logging also consists of the *heartbeat* log file, that contains information on the number requests processed by the system from its startup, adding new information at each beat, and the *storm-backend-metrics.log* log file. An example of the latter is shown in Table 4.4.

The information stored in this logfile are the type of operation, the number of operation in the last minute, the number of operation from last startup, the maximum, minimum and average duration of the operation from the last minute and the highest duration of the last 95th and 99th percentile of operation.

Chapter 5

Infrastructure setup

As explained in the previous section, the amount of data being regularly stored in the Log Repository averages to 12 GB of data per day. This amount of data, while still being manageable is more and more close to the definition of BD, discussed in Section 1.2.

To elaborate this amount of data, a standard computer may prove insufficient. A single machine can perform elaboration in parallel on its cores, but oftentimes this parallelisation is not enough for BD. A step forward from core parallelisation is distributed computing: A distributed system is a system whose components are located on different networked computers, which communicate and coordinate their actions to divide a problem into many tasks, each of which is solved by one or more computers.

In many cases it may be more cost-efficient to obtain the desired level of performance by using a cluster of several low-end computers, in comparison with a single high-end computer.

Since April 2019, CNAF started studying different system concept to develop a distributed system for Log Ingestion and Analysis, loosely inspired by previous works at CERN and at the Fermi National Accelerator Laboratory (FNAL).

I.e at CERN a complex system of heterogeneous log ingestion and analysis has been put in place to consolidate the data managing structure. A schematic representation of the framework in use at CERN is shown in Figure 5.1.

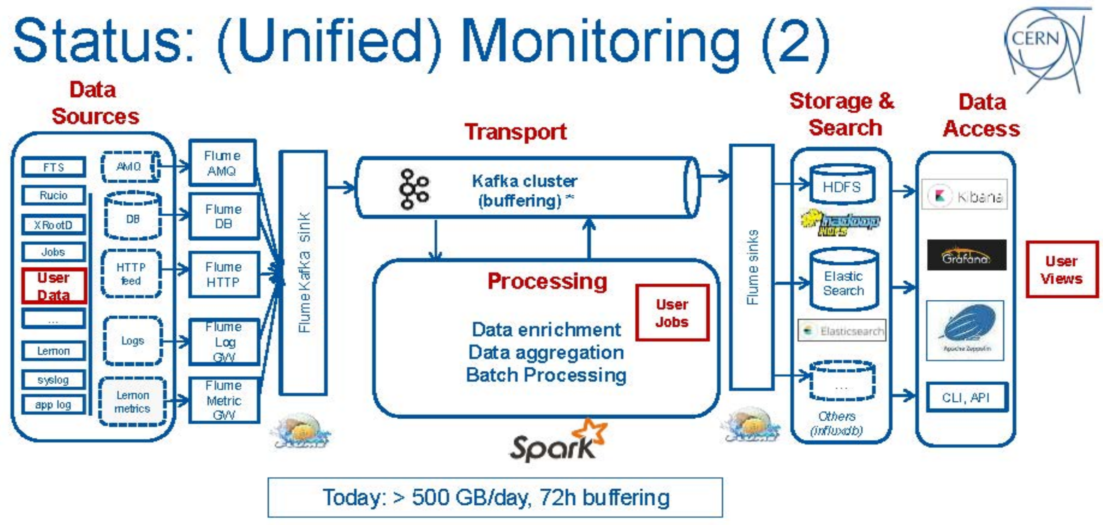


Figure 5.1: Schematic representation of the unified monitoring system in use at CERN [47].

It is worth mentioning that the tools and techniques used to extract and analyse data from such a large and complicated "dataset" at CERN are many, and very different in scope and functions. While some have been developed by HEP experts for handling big HEP datasets, there is an increasing need within the field to make an effective use of industry developments, i.e. tools that have not been developed in-house but - given the synergy among HEP and non-HEP on BD-related need - they nevertheless seem to offer exactly what HEP needs.

The adoption of industry techniques that have not been developed in-house by HEP experts implies (at least) a training curve and (for sure) additional work to implement some level of adaptation of such tools to the specific details of HEP work-flows, which is often a far from trivial work. At the same time, industry techniques, mainly if open source, can rely on a large and open developers communities, whose knowledge is capable to streamline technology adoption and troubleshooting on a stable basis and in a sustainable fashion, which is a hard to meet goal for CERN specific products, which suffer from risk related to inability to guarantee stable funding schemes for hardware, software and human resources on a multi-year scale.

Following the same structure, albeit with less complexity, the framework de-

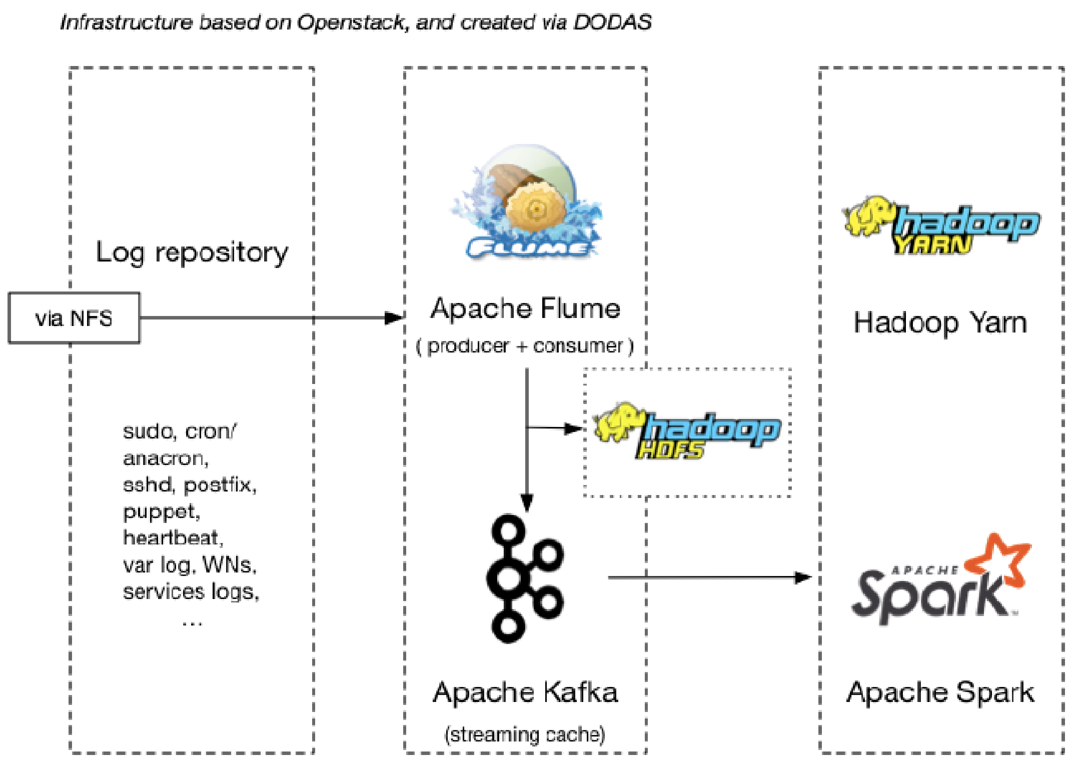

Infrastructure based on Openstack, and created via DODAS
Log repository
via NFS
sudo, cron/ anacron, sshd, postfix, puppet, heartbeat, var log, WNs, services logs, ...
Apache Flume
(producer + consumer)
hadoop HDFS
Apache Kafka
(streaming cache)
hadoop YARN
Hadoop Yarn
Apache Spark
Apache Spark

Figure 5.2: Schematic representation of the distributed log ingestion and analysis platform being implemented at CNAF.

veloped to satisfy the need of the new CNAF log repository is schematically represented in Figure 5.2.

A complete description of the technical components of this framework is beyond the scope of this thesis, but few highlights on the major software toolkits in use is given in the following sections.

5.1 DODAS

The infrastructure developed is based on Openstack [48], a cloud operating system that controls and manages the resources of the datacentre. The instances created through Openstack form a cloud network over which is distributed the software that defines the framework described above.

The distribution is done via the Dynamic On Demand Analysis Service (DODAS), a Platform as a Service tool which allows to instantiate on-demand container-based clusters [49]. DODAS as been developed by the INDIGO-DataCloud H2020 project [50] and has been funded in the scope of the European Open Science Cloud hub (EOSC-hub) Horizon 2020 project[51].

The purpose of DODAS is to reduce the learning curve, as well as the operational cost of managing community specific services running on distributed cloud, by automating the process of provisioning, creating, managing and accessing computing and storage resources. DODAS can deploy both HTCondor batch systems, and platforms for Big Data analysis based on Spark or Hadoop such as our case.

DODAS has already been integrated by the Submission Infrastructure of CMS, as well as by the Alpha Magnetic Spectrometer (AMS-02) computing environment.

5.2 Hadoop-HDFS-Yarn

Apache Hadoop is a collection of open-source software utilities that facilitate using a distributed platform to run jobs [15]. The main software that compose this toolkit are:

- *Hadoop Distributed File System* (HDFS) - a distributed file-system that stores data on machines, with a very high bandwidth across the cluster; It has many similarities with existing distributed file systems but is designed to be deployed on low-cost hardware with a high failure tolerance. It is tuned to manage vast volumes and can easily scale to hundreds of nodes. The main difference between HDFS and other file systems such as NFS, is that it follows the assumption that is better to migrate the computation closer to where the data is located rather than moving the data to where the application is running. As such is suitable for application with large data sets, supporting tens of millions of files in a single instance.

- *MapReduce* - MapReduce is the implementation of a programming model for processing and generating big data sets with parallel, distributed algorithms on the cluster. A MapReduce framework is defined by two operations: Map: a function that each worker node applies to the local data, writing the output to a temporary storage and Reduce: a function applied to the data based on the result of the mapping function. The role of the MapReduce is to redistribute the data resulting from the map function such as all the data with the same result is Reduced by the same worker node.

- *Yet Another Resource Negotiator*(YARN) - YARN is the middleware between HDFS and the processing engines being used to run applications and can dynamically allocate resources to applications as needed. By decentralising the execution and monitoring of processing jobs YARN removes performance bottlenecks and scalability problems that appears as the cluster sizes and the number of applications increased.

Using YARN to separate HDFS from MapReduce allows the Hadoop environment to be more suitable for real-time processing and online applications that would suffer from serialisation of processes [52].

5.3 Flume-Kafka

On the log ingestion side of the framework, two software manages the stream of logs:

- *Apache Flume* - Apache Flume is a distributed, software for efficiently collecting, aggregating, and moving large amounts of log data [53]. It has an architecture based on streaming data flows. It uses a simple extensible data model that allows for online analytic application. Flume can be used to transport massive quantities of event data in a reliable way: A Flume source consumes events delivered to it by an external source; the events are then staged in a channel, and are removed from it only after they are stored in the next channel or in the terminal repository. This ensures that the set of events are reliably passed from point to point in the flow.

- *Apache Kafka* - Apache Kafka is a distributed streaming platform to manage streams of data [54]. The platform can publish and subscribe to streams of records, similar to a message queue, reading incoming data and outputting data on the same stream. The software allows also to process the stream as it occurs making it possible to transform data as it arrives.

The combination of these two software, Flume for the stream of the logs and Kafka for its ability to transform the data of the stream as it occurs allows to process event with sub-second latency and scales greatly with large amount of data [55].

5.4 Spark

The previous software discussed, following the definitions of Figure 1.3, deals with the Data Management part of a BDA process.

Apache Spark is a fast and general-purpose cluster computing system used to perform Analytics investigation on data [56]. It provides high-level APIs in four different coding languages for data management and analysis: Java, Scala, Python and R. In addition, it provides an optimised engine for graphs, higher-level tools for dealing with relational databases and implements an efficient Machine Learning library.

- *Spark Core* - Spark Core is the base of the framework. It manages the dispatch of distributed and basic I/O functionalities.

- *Spark SQL* - Spark SQL is a component on top of Spark Core that introduced a DataFrames based data abstraction, which provides support for structured and semi-structured data and standard SQL queries for extract data.

- *Spark MLlib* - Spark MLlib is a distributed machine-learning framework on top of Spark Core. Many common machine learning and statistical algorithms have been implemented simplifying large-scale machine learning pipelines

All of these functions are optimised to run on distributed computing following the MapReduce model.

The system presented is still in development, and as such it is not yet in active use and no software has been yet developed that fully makes use of the platform. This type of infrastructure is still being experimented on at CNAF, with low prior knowledge and experience: using open source tools with industry wide acknowledgement and distribution is, therefore, a great advantage for the success of this type of experimentation.

The next and final chapter of this dissertation presents the development of a BDA algorithm for the clustering of Log files. The script presented is developed with the objective of presenting a preliminary work that could be performed on the distributed platform described above.

However, due to the still in-development state of the system, the algorithm is not yet optimised to be run on a distributed system and future efforts are needed to adapt the code presented and employ it on the platform in a real use-case scenario.

Chapter 6

Log files analytics

In Summer 2018, the INFN-CNAF management decided to launch an exploratory effort to investigate the feasibility of the collection of various log data from CNAF services and systems and the potential extraction of useful and actionable information. The form of this effort happened to be based on a bottom-up approach, in terms of technical choices and manpower: namely, a small group consisting of 3 young physicists with computer science skills (the author of this thesis has been one of them) were hired for a two-months period in the CNAF Summer School program, with the precise goal to investigate the aforementioned direction, under the guidance of technologists from the centre and personnel from the University of Bologna.

The CNAF Summer School program focused on the investigation of log files collected by StoRM, that proved to be the most well documented and readily accessible service. More details on the service are discussed in Section 4.3 with an overview of the log files that will be discussed in the following.

This CNAF Summer School program paved the way for further efforts in the development of a consolidated log ingestion pipeline that resulted in the Log Repository presented in Section 4.2. The rest of this thesis will discuss a first example of an Analytics work that can be performed on this new platform and as a consequence of the preliminary activities done in the context of the CNAF Summer School program.

6.1 Preliminary investigation of CNAF log files

The Summer School period was from the beginning of September 2018 to the end of October 2018 (even if a fraction of activities continued also beyond). In this section, the work done by the students' team is briefly described. The preliminary exploration of a first log ingestion pipeline at CNAF is presented in Section 6.1.1. The development of a series of algorithms for the parsing of log files is discussed in 6.1.2. The definition of a prototype of Machine Learning algorithm for feature selection is shown in Section 6.1.3. These works have been presented at the International Symposium on Grids & Clouds 2019 (ISCG 2019) and released in the proceedings book of the conference [57, 58].

6.1.1 Log ingestion and visualisation

This work focused on the creation of an indexed system with structured information from CNAF system logs by using the Elastic stack (ELK), an open source project developed by the Elastic company [14] that is the union of three different software: Elasticsearch, Logstash and Kibana.

Logstash allowed the creation of a well defined pipeline collecting log files from the Storage Manager, the different files were then parsed using various filters. Based on Regular Expression, these filters selected a specific portion of text by creating a series of patterns. Some patterns were predefined and universal (i.e. IP address, timestamp, etc.), but the majority were custom made for each log file.

Then, using the Kibana User Interface, the information extracted from the log parsing were plotted with several histograms and charts.

Elasticsearch provided a tool with Machine Learning functionality to predict a future time interval given a specific portion of past data for training.

The tool available in the free version of the software, although, was not optimal and no parameters could be adjusted, resulting in a system that did not take into account any possible quick fluctuation of the examined metric; as such, for a predictive scenario and a proactive identification of failures, this did not prove to be the optimal solution.

TIME - **STRING** [(count= **NUM**, m1_rate= **NUM**,m5_rate= **NUM**, m15_rate= **NUM**) (max= **NUM**, min= **NUM**, mean= **NUM**, p95= **NUM**, p99= **NUM**)] duration_units=milliseconds, rate_units= events/minute

Table 6.1: Log line from storm-backend-metrics.log with fields replaced by identifiers.

6.1.2 Log parsing and structuring

The activity of the second student, fully independent from the ELK implementation discussed in the previous section, focused instead on the parsing of the raw log files into structured .csv files.

The logs collected by the system vary a lot, and their content depends on the machine from which such logs are acquired. Most log files consist of a heterogeneous array of measurements, events, descriptor keys and separators. Usually one event log file is written in a way that makes it more readable for users, which turns into ad advantage for a human reader but also adds redundancy which is not meaningful information for a machine.

In order to extract the relevant information from a log file, a parsing algorithm can be applied to pre-process the data and convert the log text file into a form that is more adequate for database-like manipulation applications.

Depending on its source and purpose, log files themselves can follow very different structures. The log line in Table 4.4 in Chapter 4, for example, is from the Backend (BE) metrics file from a storage machine managing the resources for the ATLAS experiment. This log line includes numerous descriptors that define the subsequent metrics, and every relevant metric is divided into blocks with the use of separation symbols such as parenthesis and square brackets. Every line in this log follows the same structure, with an apparent template easy to extract from the information fields (Table 6.1).

The result of the manipulation is a csv file (shown in Table 6.2), that is easy to read and, more importantly, easy to manipulate for any statistical analysis (even machine learning techniques).

timestamp	type	count	m1_rate	m5_rate	m15_rate	max	...
00:00:17.827	synch.ls	10669590	329.633	362.073	501.721	613.200	...

Table 6.2: Truncated storm-backend-metrics parsed .csv file.

12/01 03:48:09 Thread 59 - INFO [e701...]: ns1__srmPutDone : Request: Put done. IP:...
12/01 03:48:09 Thread 12 - ERROR [d857...]: Result for request 'Release files' is 'SRM...
12/01 03:48:10 Thread 41 - INFO [41ee...]: Request 'PTP status' from Client IP='200...
12/01 03:48:15 Thread 3 - INFO [76f8...]: ns1__srmPing : Request: Ping. IP: 2001:145...

Table 6.3: Four sample lines from storm-frontend-server.log.

The log in Table 4.4, in Chapter 4, is an example of one of the easier log file to parse: the template is well structured and maintained throughout the entire log file, with little to no weird lines.

Other logs, although, follow a looser structure. Service message log files are more verbose, the log lines are heterogeneous, and the relevant information are hidden between descriptors with varying length and in different orders.

For StoRM, an example of this type of log file is the service log for the frontend (FE). Details on this log files are available in Section 4.3.1. Table 6.3 shows some sample lines from the log file. While a loose structure can be inferred from the log lines, one could notice how — beyond the timestamp, the information on the thread and the type of log (INFO, WARN, ERROR) — the verbose message containing the information on the service does not follow a strict template.

Because of this, a log file of this type is loosely parsed, and the information present in the message are to be retrieved manually. The results of the parsing via the same script described above are shown in Table 6.4.

Most service events' log files follow a structure similar to the one shown in Table 6.3, with a verbose message written without a strict structure, with heterogeneous log lines.

The repository containing the algorithm described in this Section is available in [59].

The results of this algorithm were employed in the third preliminary work, to

timestamp	thread	type	request_id	message
12/01 03:48:09	59	INFO	e701...	ns1__srmPutDone : Request: Put done. IP...
12/01 03:48:09	12	ERROR	d857...	Result for request 'Release files' is 'SRM_F...
12/01 03:48:10	41	INFO	41ee...	Request 'PTP status' from Client IP='200...
12/01 03:48:15	3	INFO	76f8...	ns1__srmPing : Request: Ping. IP: 2001:14...

Table 6.4: Sample of a truncated storm-frontend-server.csv file.

further extract information from the FE log files. The following Section discussed this last example.

6.1.3 Log features extraction with ML

This study analysed two different sources of log files from the Storage Resource Manager system of CNAF and correlated the extracted features in two different critical periods.

The work focused on some InfluxDB metrics from the ATLAS hosts, and the corresponding log files from the FE. For both sources, using various Machine Learning algorithms, the authors defined the most important features for the discrimination between "good" and "bad" days.

The critical periods were manually labelled after investigating the system for anomalies and reading ticket reports.

The log files coming from StoRM for FE have been parsed using the same algorithm presented in 6.1.2, and after studying the content of the log lines, some key fields in the messages were selected as the input features for successive steps of the process.

The parsed message was then converted through one hot encoding to determine if those specific value were present or not. This allowed to prepare the log lines for Machine Learning algorithm, by converting the large, verbose and unstructured message into a feature vector.

The resulting data was used to train various supervised learning algorithm to determine which of the extracted features correlated more with the supervised labels.

The work presented followed a strictly supervise approach, that required the input from experts to label the critical periods and a careful study of the log files to manually define a method to extract the interested fields.

6.2 Case study: text-based log clustering

All the previously discussed approaches, designed and developed during the Summer School period in 2018, required a deep study of the log files, and an appropriate amount of custom filtering on the logs to extract the information. The approach presented in this thesis builds on those experiences, and it is based on a performing and effective algorithm to clusterise the log entries of all logs into classes based on the textual content of the log message.

The algorithm should be able to:

- Perform the clusterisation in a totally unsupervised way: no prior knowledge on the logs, and no manual support in the classification should be needed;
- Maintain a good resolution in the classification reducing at minimum the loss of information;
- Scale well with the size of the log files;
- Perform the clusterisation in a relatively short time;
- Be applicable to different types of log files with basically no adjustments.

By dividing the log lines into clusters, a log file can be analysed in a quantitative way, providing direct information on the type of messages and their frequency in a given log file. If this process is performed every day, after collecting the log lines of a given service from the day before, the team working on the support for such a service would benefit from receiving a daily summary of the status of a service; the added information can be used in conjunction with the already available data,

```
INPUT: parsed_log
PARAMETERS: MinimumSimilarityThreshold
INIT clusters = {}
FOR every log line msg IN parsed_log
    Anonymise msg
    INIT similarities = {}
    FOR every cluster{i} IN clusters
        similarity{i} = similarity(cluster reference, msg)
        IF similarity{i} is min(similarities)
            IF similarity{i} > MinimumSimilarityThreshold
                Add msg to cluster{i}
    IF no cluster matched msg
        Add new cluster
        Add msg as new cluster reference
```

Table 6.5: Pseudocode of the log clustering algorithm. Details in the text.

and can provide a quick glance on the behaviour of a service, possibly highlighting anomalous trends.

To reach this goal (i.e. treating the service support team as a customer to satisfy), the proposed algorithm must perform the clustering on the log file for a specific day in a way that the clustering processing completes in less than 6 to 8 hours, as to be performed around midnight and and be ready by the morning of the next day.

The algorithm is shown in pseudocode in Table 6.5. The algorithm has been developed in Python 3.6, using the open source scientific computing package NumPy [60] and the data structures and data analysis tools from the Python Data Analysis Library (Pandas) [61]. The repository containing the algorithm developed for this thesis work is available at [62].

In the following sections the main steps of the algorithm are discussed.

Unless specified otherwise, the images and tables shown refer to the results of the clustering performed on *storm_frontend_server20181202.log*. This is the log file corresponding to the StoRM FE service for the ATLAS host for the 2/12/2018. See Section 4.3.1 for details on this specific log file.

srmPing : Request: Ping. IP: **2001:1458:301:4a::100:5a**
srmLs : Client DN: **/DC=ch/DC=cern/OU=Organic Units**: ATLAS Data Management
Request 'PTP' Requested '1' SURL(s): **srm://storm-fe.cr.cnaf.infn.it/atlas/**

Table 6.6: Sample log messages with highlighted fields to be masked.

srmPing : Request: Ping. IP: **IP**
srmLs : Client DN: **PATH** ATLAS Data Management
Request 'PTP' Requested '1' SURL(s): **SURL**

Table 6.7: Sample log messages with masked fields.

6.2.1 Log anonymisation

Service log files, such as those from FE and BE of StoRM, can contain numerous information on the user and the machine from which the service was launched. These information are useful for tracking the source of a problem, but translates into an added level of complexity for a clustering algorithm.

Every information on a verbose log, indeed, is an additional word to compare to determine the similarity between cluster members. To improve the performance of this algorithm, these types of information must be masked. This procedure is often defined as "anonymisation", since is usually applied to hide personal information and sensible data.

A sample of log lines to be anonymised is shown in Table 6.6. To perform the anonymisation the software uses compiled regular expression to identify the underline structure of this fields. The process is quick and, since the masked values are general enough, is easily adaptable to different logs and can be often used as is.

The result of the masking process is shown in Table 6.7.

6.2.2 Comparison of similarity measures

The algorithm clusterises the log messages based on the similarity between the message to clusterise and the reference messages of the clusters.

Three different similarity measurements have been applied, testing the speed of the algorithm, the number of clusters created and the cluster variance, that defined how similar members of the same cluster are between each other.

$\mathcal{J}$: Jaccard index

The first distance evaluated is the Jaccard Index. It is defined as the ratio between the size of the intersection of two sets and the size of the union of the same sample.

$$\mathcal{J}(A,B) = \frac{A \cap B}{A \cup B} \tag{6.1}$$

The resulting value $0 \leq \mathcal{J}(A,B) \leq 1$ measure the similarity between the two sets: with 0 corresponding to two sets with no intersection and 1 corresponding to two coincident sets.

When the two sets are a series of words, the union corresponds to the set of all the unique words in the strings, and the intersection corresponds to the common words between the two strings.

Jaccard similarity has been extensively used with sets of strings to compare search queries ([63] [64] [65]), with positive results when used on small sets. Its use is therefore applied to large collection of small sets, such as search queries or bit vectors.

One of the main advantages of this method is its speed, requiring only $O(n)$ to find the intersection between the two sets, with n the size of the union of the two sets.

The disadvantage of this method is that it does not take into account multiple occurrences of the same words in a set, and it loses information on the order of the words.

$\mathcal{C}$: Cosine similarity

The cosine similarity is a measure of similarity between two non-zero vectors. It works by computing the cosine of the angle between the two vectors:

$$\mathcal{C}(A, B) = cos(\theta) = \frac{A \cdot B}{\|A\|\|B\|} \tag{6.2}$$

as such the cosine similarity is bounded by $-1 \leq \mathcal{C}(A, B) \leq 1$, with 1 corresponding to two parallel vectors with the same direction, 0 to two orthogonal vectors and -1 two parallel vectors with opposite directions.

When the two sets are series of words, the vectors are defined by the occurrences of every words in the two sets. The cosine similarity is naturally bounded between $[0, 1]$ since negative values are impossible.

Cosine similarity is widely used in text analysis, especially to compare documents and large corpus based data mining ([66] [67]). In [68], cosine similarity is the metrics of choice for the log clustering algorithm this work is based on.

Cosine similarity algorithm are very efficient for sparse vectors, and as such are used to compare documents. Their efficiency is lowered when used with small sets, and as such prove to be slow when used with large collection of small sentences.

The time complexity of cosine similarity is, in fact, $O(n^2)$ with n the dimensions of the vectors.

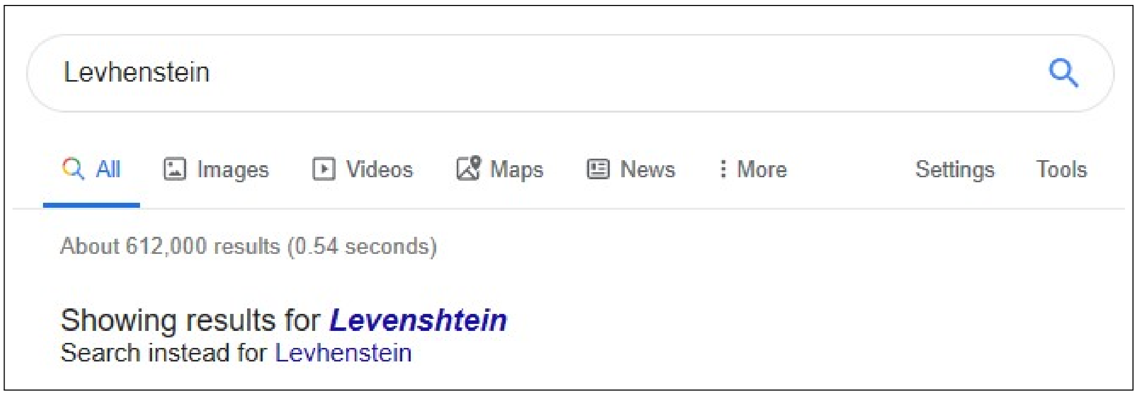
Again, while keeping track of the occurrences of every word in the sentences, this algorithm does not take into account the relative position and the order of appearance of the words.

This algorithm was implemented in the code through the scikit-learn library [10].

$\mathcal{L}$: Levenshtein distance

The last similarity measure analysed is based on the Levenshtein distance. The Levenshtein distance is defined as the distance between two string sequences, as the minimum number of character edits necessary to change a sequence into the other.

Given $|S|$ the length of a string S, the distance between two strings A and B


Levhenstein

All Images Videos Maps News More Settings Tools

About 612,000 results (0.54 seconds)

Showing results for ***Levenshtein***
Search instead for Levhenstein

Figure 6.1: Fuzzy string search in a common web search engine.

is given by:

$$\mathcal{L}_{A,B}(i{=}|A|\,,j{=}|B|) = \begin{cases} \max(i,j), & \text{if } \min(i,j){=}0 \\ \min \begin{cases} \mathcal{L}_{A,B}(i-1,j)+1 \\ \mathcal{L}_{A,B}(i,j-1)+1 \\ \mathcal{L}_{A,B}(i-1,j-1)+\mathbf{1}_{A_i \neq B_j} \end{cases} & \text{otherwise.} \end{cases} \tag{6.3}$$

where $\mathbf{1}_{A_i \neq B_j}$ is equal to 1 when the i-th letter of A and the j-th letter of B are different, and 0 otherwise; $\mathcal{L}_{A,B}(i,j)$ is the Levenshtein distance between the first i characters of A and the first j characters of B.

As per 6.3 the Levenshtein distance between two strings is measured recursively as the minimum amount of deletion (the first element in the minimum), insertion (for the second element) and edits (third element) needed to change A into B.

The Levenshtein distance is 0 if the two strings are identical, and at most is the size of the longer strings if every character mismatch.

Levenshtein distance is mostly used in fuzzy string search, where the objective is to find a string that approximately match a pattern (One common example is seen in Figure 6.1). In log files this distance is used to anonymise data [37], and to cluster search queries both in [69] and [63].

This algorithm is very efficient when comparing a short string with a large text, but is used also for comparing long strings to each other. The computational

Sample Phrases	$\mathcal{J}$	$\mathcal{C}$	$\mathcal{L}$
"Test sample" "Test trial"	0.333	0.500	0.667
"Test sample" "Sample test"	1.000	1.000	0.545
"Test sample" "Test smaple"	0.333	0.500	0.909
"Test sample test" "Test sample"	1.000	0.949	0.815

Table 6.8: Comparison of the results between different similarity measure applied on sample sentences.

complexity, although, increases since it is $O(nm)$, with n and m the respective length of the two strings compared.

This distance measure has the advantage of taking into account the order of appearance of the words.

Since this metric is a measure of distance between words, for our uses must be converted into a measure of similarity:

$$\mathcal{L}_{sym}(A, B) = 1 - \frac{\mathcal{L}(A, B)}{\max(|A|, |B|)} \tag{6.4}$$

This algorithm as been implemented through the use of the Levenshtein Python C extension module, available at [70].

Comparison between $\mathcal{J}$, $\mathcal{C}$ and $\mathcal{L}$

Although the similarity is measured in different ways, all three algorithms bound the similarity between 0 and 1, as the percentage of similarity between to texts.

Table 6.8 shows the results of the application of the three algorithms on some sample sentences.

The Table shows the different behaviour of each algorithm. The second row shows how neither $\mathcal{J}$ nor $\mathcal{C}$ weighs the order of the words. By comparing the third and first row one can infer how $\mathcal{J}$ and $\mathcal{C}$ do not consider small difference in the word, such as spelling error or difference between plural and singular. The last row shows the difference between the weighs of words occurrences in the sentences with $\mathcal{J}$ that does not account for repetitions at all.

By measuring the number of log lines clustered per second, the Cosine similarity was immediately discarded. Even with a low similarity threshold, that produces a small amount of clusters, this similarity greatly under-performed with respect to the other algorithms. A small sample log files of 20k lines was parsed in over a minute, while the other algorithms managed to parse the log file in less than 10 seconds. Table A.1, A.2 and A.3 in Appendix A.1 list the number of lines parsed per second by each algorithm at different similarity threshold. Assuming a linear time trend — assumption that will be verified in the next Section — the Cosine similarity, with a low threshold, could parse a log file of 2M lines in over 2 hours, resulting in only four clusters with a high variance (See Figure A.2).

The Jaccard index, on the other end, proved to be really fast, parsing the sample file in 8 seconds with the more restrictive similarity threshold, but resulted in a very large number of clusters even at low similarity threshold, each with an high variance despite their number (See Figure A.1).

For these reasons the measure of similarity selected is the Levenshtein similarity, that will be used throughout the rest of this work.

6.2.3 Time complexity

The algorithm is employed to parse single log files corresponding to a day of logged events, running it from a medium range laptop. A new cluster is created once a log line results too dissimilar to every existing clusters to be merged to any of them, as such the number of clusters increases sequentially.

The plot in Figure 6.2 shows how the number of clusters increases steadily at the very beginning of the file, and only few clusters are created in the middle and the end of the files. More specifically 80% of the clusters are created in the first 250000 lines of the log file. Appendix A.2 shows the same results for the log files from other days.

The number of log lines clustered, as a function of the running time of the

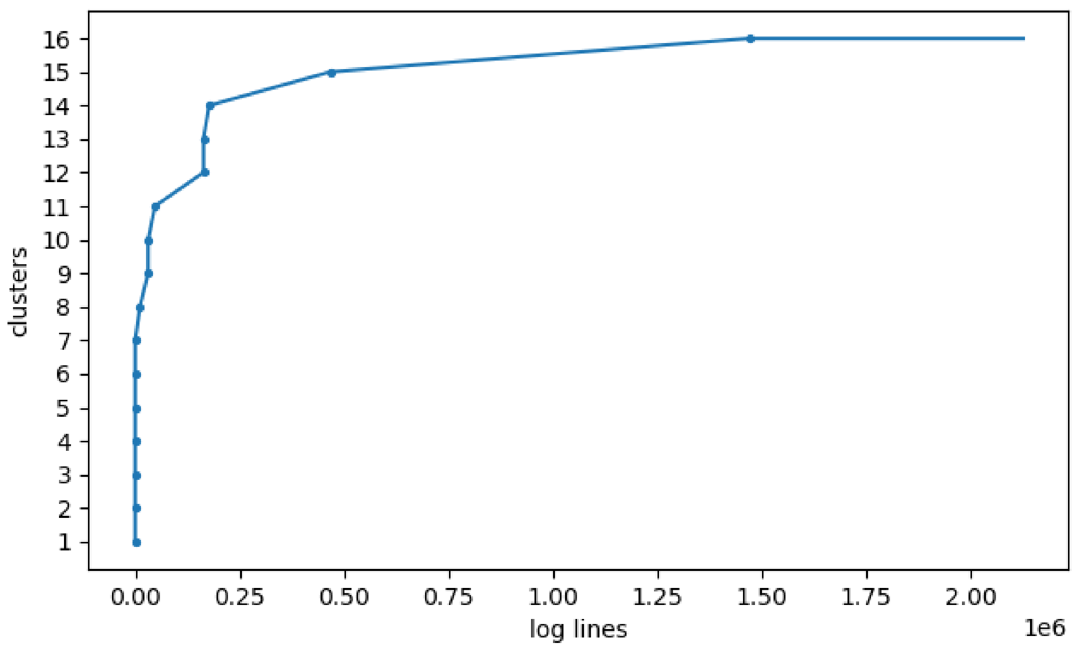


Figure 6.2: Number of clusters as a function of lines per log, in FE server log (02/12/2018) with similarity threshold = 0.7.

algorithm, follow an almost linear trend for the second half of the cluster. This can be seen in Figure 6.3.

The majority of the clusters are created in the first seconds of computation and, by plotting the same data for the first seconds of running time (Figure 6.4), it seems that the number of log lines clustered decreases every time a new cluster is created.

Since most of the log are created in the first seconds, in Figure 6.3 the number of log lines clustered per second seems to be linear with the maximum number of clusters.

Once the similarity measure to use has been determined - in this case, as discussed, the Levenshtein one - the only parameter in the algorithm left to be determined is what can be referred to as the "similarity threshold", i.e. the lower limit of similarity that a message must have with respect to a cluster to be merged with it.

As such, the similarity threshold is a key parameter as it determines the number

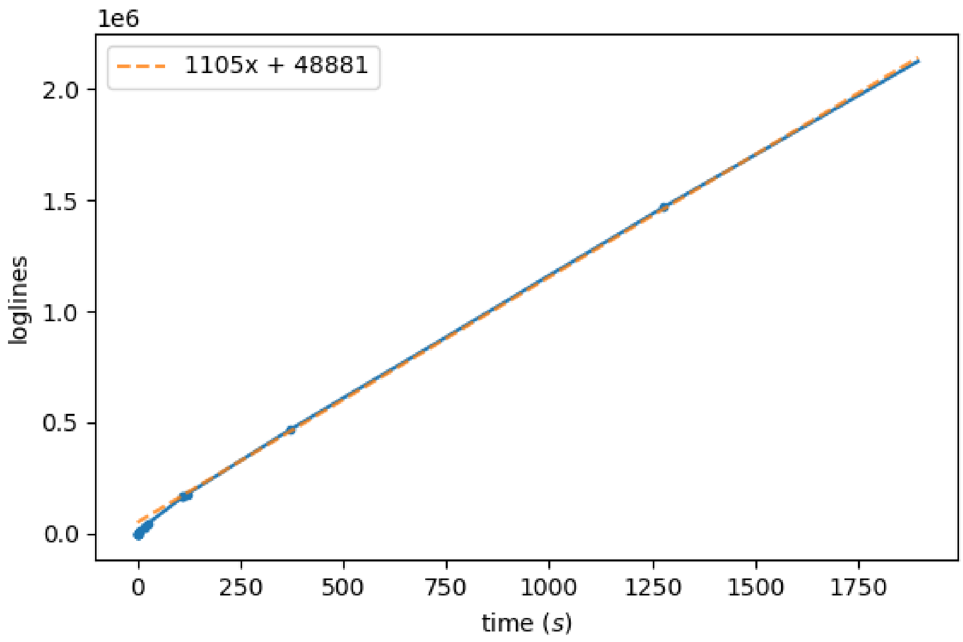


Figure 6.3: Number of log lines parsed as a function of the running time, and the matching fit line for FE server log (02/12/2018). The markers correspond to the creation of a new cluster.

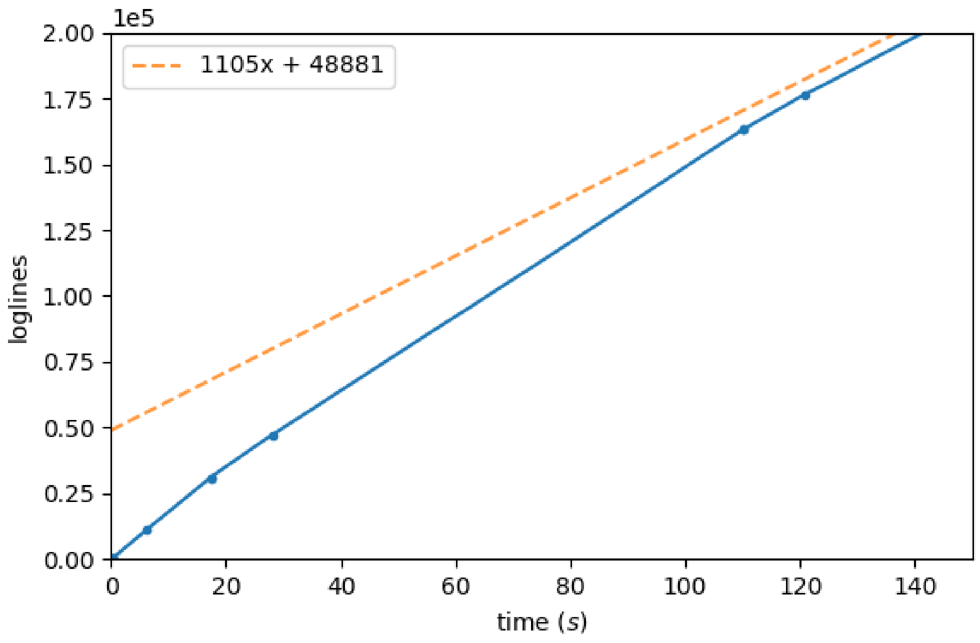


Figure 6.4: Close up of Figure 6.3 for the first seconds of runtime. The markers correspond to the creation of a new cluster.

of clusters that gets created from a log file. Figure 6.5 shows how the number of log lines clustered, as a function of the running time, is lower for higher similarity thresholds.

Table 6.9 shows the maximum number of clusters created for each similarity threshold and the slope of the fit line, that roughly corresponds to the average number of log lines clustered per seconds.

6.2.4 Levenshtein similarity threshold

The similarity threshold used determines the number of clusters and consequently the time needed to cluster the log file. Since there is no stochasticity in the clustering of a specific log file with a specific similarity threshold, this value also determines the size, the shape and the reference message line of each cluster.

The similarity threshold to be chosen should, therefore, minimise the loss in information in the clustering, by both having low overlap between clusters and

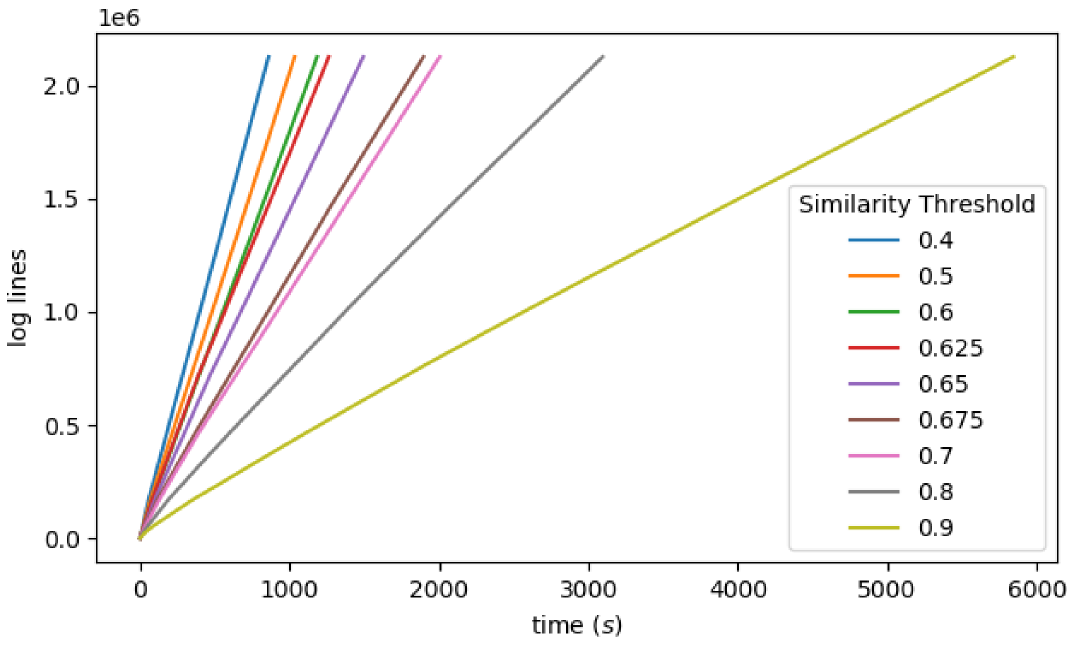


Figure 6.5: Number of log lines clustered as a function of the running time for the FE server log (02/12/2018). The slope of the curve decreases for higher similarity thresholds.

Similarity threshold	Clusters	Slope
0.40	5	2445
0.50	7	2031
0.60	9	1778
0.62	10	1655
0.65	12	1403
0.68	17	1105
0.70	19	1041
0.80	32	677
0.90	64	357

Table 6.9: Total number of clusters and slope of the fit line for the evaluation time of the FE server log (02/12/2018) at different similarity thresholds.

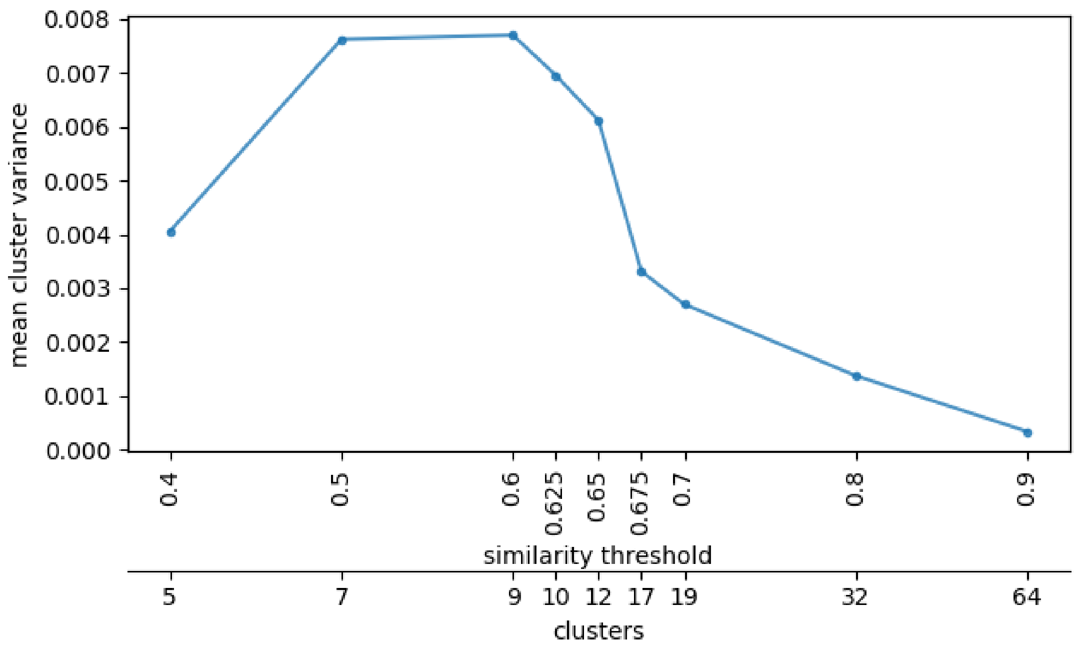


Figure 6.6: Mean in-cluster variance as a function of the similarity threshold and the corresponding number of cluster for the FE server log (02/12/2018).

small variance in the same cluster. The computation time should also be taken into account, since it should satisfy the requirement — discussed at the start of this Section — of clustering a daily log file in less than 6-8 hours. By that, it was meant the following: if the developed algorithm, when run even on a medium range personal laptop only (and not on a high end CNAF server), will end up in being able to perform clustering on an average log file within the 6-8 hours threshold, it will be considered a success. Indeed, there is not need to achieve higher speed in such prototype, as it will eventually run on more performing hardware (or even on a BDA platform).

Figure 6.6 shows the mean cluster variance as a function of the similarity threshold and the corresponding number of cluster. The variance is measured as the squared deviation from the mean of the members of a cluster's similarity from the reference message of the cluster. The mean cluster variance is the average of this value across all of the clusters created with a specific similarity threshold.

Figure 6.6 shows how the mean cluster variance is higher for lower similarity threshold and lower for higher threshold, as expected, but with a low value at similarity threshold of 0.4.

Since the criteria for the choice of similarity was a low cluster variance and low computation time, a similarity of 0.4 seems to be the best choice for this particular log file.

Although, by considering only the clusters containing the highest amount of log lines, the behaviour changes. Figure 6.7, shows the mean cluster variance as a function of the similarity threshold and consequently the number of clusters for different percentages of the log file. In the plot, the solid line (referred to as "p100") matches the line in Figure 6.6. The dashed and dotted lines corresponds to different percentiles of the data, i.e. "p80" corresponds to the mean cluster variance of the lowest number of most populated clusters that collect at least 80% of the log lines, and "p90" ("p95") corresponds to at least 90% (95%) respectively.

Even if a similarity of 0.4 seems to yield a small mean cluster variance, by removing the less populated clusters, the mean cluster variance increases drastically: the most populated clusters collect log lines with higher distance from the reference message. For higher threshold the mean cluster variance decreases noticeably, while the number of clusters increases. This indicates a high number of highly specific clusters; a similarity threshold of 1 would result in a number of cluster equals to the number of unique messages in the log files.

For this log file, the similarity threshold that seems to satisfy well the requirements is between 0.675 and 0.7; in both cases, in fact, the mean cluster variance is similar even for smaller percentile of data.

This result seems to be consistent with the same analysis performed on log files from the same source at different days; the results can be seen in Appendix A.3. The result of the clustering algorithm with similarity threshold 0.675 is visible in Figure 6.8 where every bar is labelled with the cluster's reference message. The same results are shown in logarithmic scale in Figure 6.9 to show the trend of the least populated clusters. Table 6.10 lists the clusters population and the corresponding reference message.

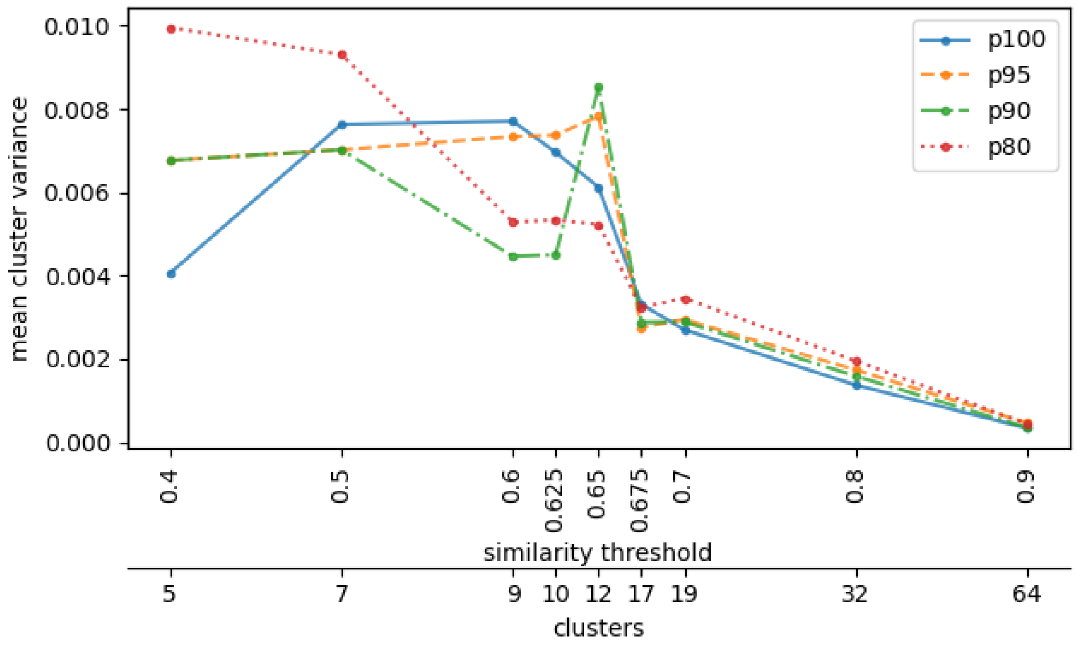


Figure 6.7: Mean cluster variance at different similarity levels for the entire log file (p100) for the 95th percentile (p95) for the 90th percentile (p90) and the 80th percentile (p80) for the FE server log (02/12/2018).

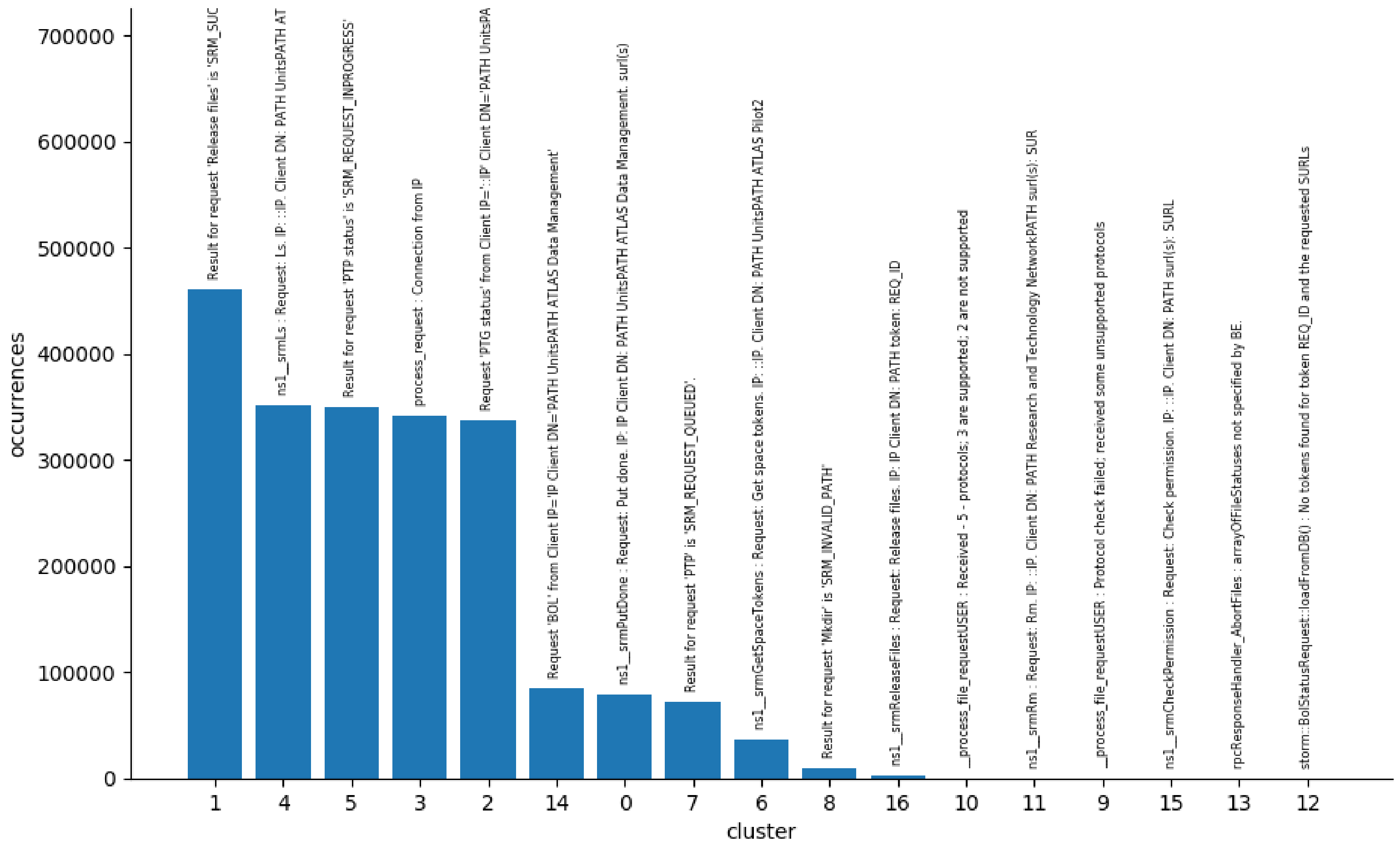


Figure 6.8: Clusters population for FE server log file (02/12/2018), with similarity threshold 0.675. Every bar is labelled with the cluster's reference message.

Cluster ID	Occurrences	Reference message
0	79117	ns1__srmPutDone : Request: Put done. IP: IP Client[...]
1	460708	Result for request 'Release files' is 'SRM_SUCCESS[...]
2	337391	Request 'PTG status' from Client IP=':IP' Client [...]
3	340976	process_request : Connection from IP
4	351179	ns1__srmLs : Request: Ls. IP: ::IP. Client DN: PAT[...]
5	349696	Result for request 'PTP status' is 'SRM_REQUEST_IN[...]
6	36551	ns1__srmGetSpaceTokens : Request: Get space tokens[...]
7	72440	Result for request 'PTP' is 'SRM_REQUEST_QUEUED'. [...]
8	9948	Result for request 'Mkdir' is 'SRM_INVALID_PATH'
9	87	__process_file_requestUSER : Protocol check failed[...]
10	174	__process_file_requestUSER : Received - 5 - protoc[...]
11	139	ns1__srmRm : Request: Rm. IP: ::IP. Client DN: PAT[...]
12	1	storm::BolStatusRequest::loadFromDB() : No tokens [...]
13	10	rpcResponseHandler_AbortFiles : arrayOfFileStatuse[...]
14	84975	Request 'BOL' from Client IP='IP Client DN='PATH U[...]
15	26	ns1__srmCheckPermission : Request: Check permissio[...]
16	3415	ns1__srmReleaseFiles : Request: Release files. IP:[...]

Table 6.10: Cluster population and reference message for FE server log file (02/12/2018) with similarity threshold 0.675.

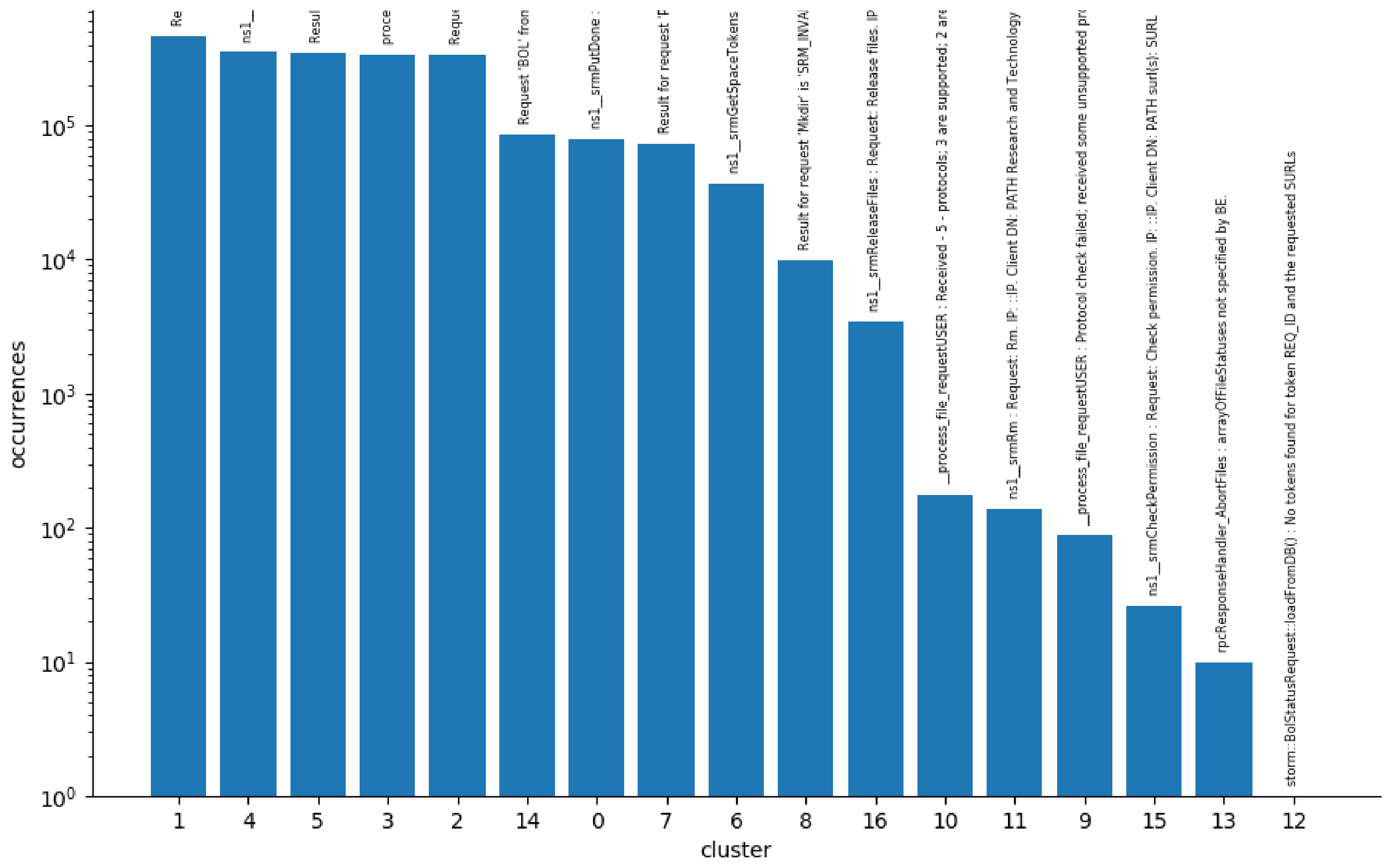


Figure 6.9: Clusters population for FE server log file (02/12/2018) with similarity threshold 0.675, in logarithmic scale. Details in the text.

6.2.5 Reference message

To determine if a log line is to be merged to a preexisting cluster, the algorithm compares the log line to the reference message of the cluster. The reference message of a cluster corresponds to the first log line added to that specific cluster; this is done to ensure that the time complexity of the algorithm does not increase exponentially with the number of log lines clustered.

The downside is that the reference message does not change during the run, and can result in a non-optimal choice, determined only by the order of appearance of the messages in the log file.

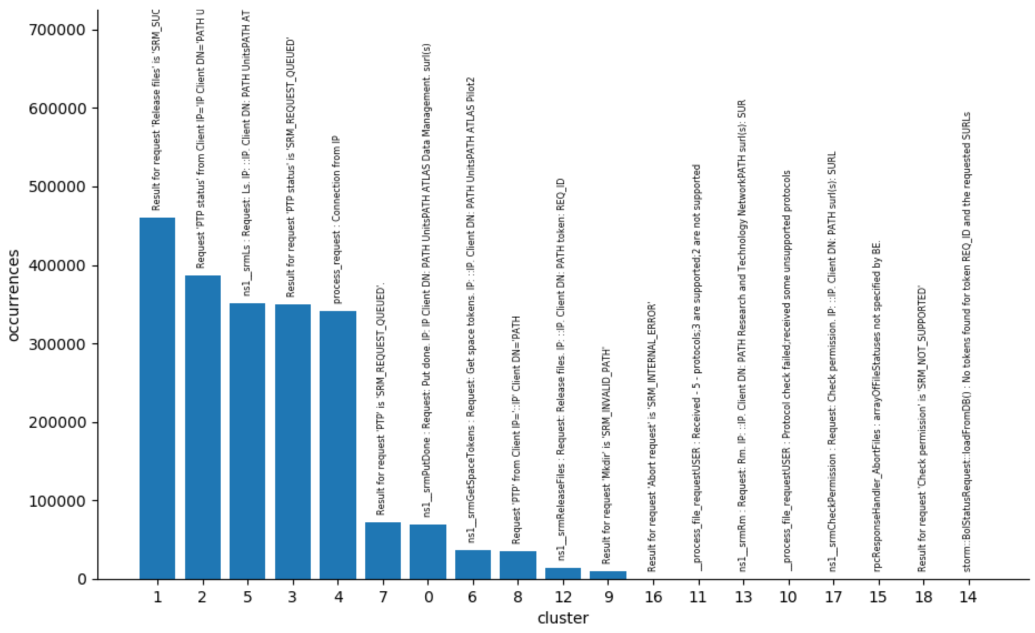
The assumption made is that, since a log message follows a template defined by the developer when printing the log, the number of unique type of log messages is finite; if the log file is clustered correctly, every message belonging to a specific cluster comes from the same template.

If this assumption is true, the order of appearance of the logs should not alter significantly the result of the clustering. By comparing the result of the algorithm for random samples of a log files, any significant difference due to the order of appearance of the log lines should be readily visible.

Figure 6.10 shows the clusters population for the FE log file of 2/12/2018, clustered with a similarity threshold of 0.7. The label on each bar corresponds to the cluster's reference message.

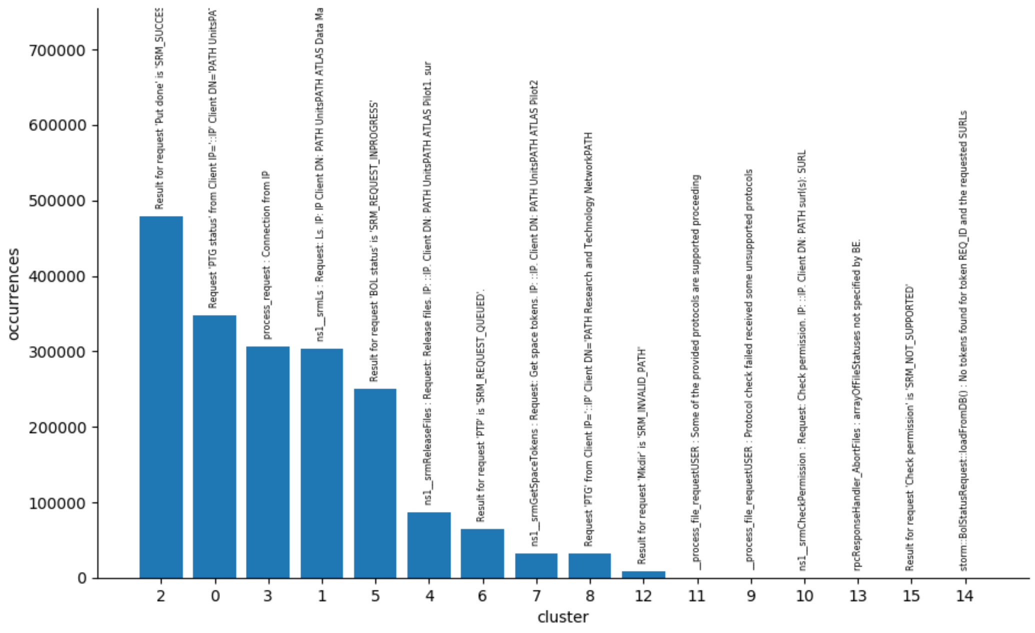
The same log file has been randomly sampled for 90% of its rows. Figure 6.11 shows the result of the same clustering algorithm performed to a sample of the log file.

The two plots follow a nearly identical distribution, and even if the reference messages differ from each other, the most popular clusters seem to refer to the same message template. This gives confidence that the usage of the cluster's reference message as from this work does not induce any relevant bias (more considerations on this point, regarding possible next steps of this work, can be found in 6.4. Additional sampling results are shown in Appendix A.4



```
Result for request 'Release files' is 'SRM_SUC
Request 'PTP status' from Client IP='IP Client DN='PATH U
ns1__srmLs : Request: Ls. IP: ::IP. Client DN: PATH UnitsPATH AT
Result for request 'PTP status' is 'SRM_REQUEST_QUEUED'
process_request : Connection from IP
Result for request 'PTP' is 'SRM_REQUEST_QUEUED'.
ns1__srmPutDone : Request: Put done. IP: IP Client DN: PATH UnitsPATH ATLAS Data Management. surl(s)
ns1__srmGetSpaceTokens : Request: Get space tokens. IP: ::IP. Client DN: PATH UnitsPATH ATLAS Pilot2
Request 'PTP' from Client IP=':::IP' Client DN='PATH
ns1__srmReleaseFiles : Request: Release files. IP: ::IP. Client DN: PATH token: REQ_ID
Result for request 'Mkdir' is 'SRM_INVALID_PATH'
Result for request 'Abort request' is 'SRM_INTERNAL_ERROR'
_process_file_requestUSER : Received - 5 - protocols;3 are supported;2 are not supported
ns1__srmRm : Request: Rm. IP: ::IP. Client DN: PATH Research and Technology NetworkPATH surl(s): SUR
_process_file_requestUSER : Protocol check failed;received some unsupported protocols
ns1__srmCheckPermission : Request: Check permission. IP: ::IP. Client DN: PATH surl(s): SURL
rpcResponseHandler_AbortFiles : arrayOfFileStatuses not specified by BE.
Result for request 'Check permission' is 'SRM_NOT_SUPPORTED'
storm::BolStatusRequest::loadFromDB() : No tokens found for token REQ_ID and the requested SURLs
```

Figure 6.10: Clusters population for FE server log file (02/12/2018), with similarity threshold 0.7. Every bar is labelled with the cluster's reference message.



```
Result for request 'Put done' is 'SRM_SUCCES
Request 'PTG status' from Client IP=':::IP' Client DN='PATH UnitsPA
process_request : Connection from IP
ns1__srmLs : Request: Ls. IP: IP Client DN: PATH UnitsPATH ATLAS Data Ma
Result for request 'BOL status' is 'SRM_REQUEST_INPROGRESS'
ns1__srmReleaseFiles : Request: Release files. IP: ::IP. Client DN: PATH UnitsPATH ATLAS Pilot1. sur
Result for request 'PTP' is 'SRM_REQUEST_QUEUED'.
ns1__srmGetSpaceTokens : Request: Get space tokens. IP: ::IP. Client DN: PATH UnitsPATH ATLAS Pilot2
Request 'PTG' from Client IP=':::IP' Client DN='PATH Research and Technology NetworkPATH
Result for request 'Mkdir' is 'SRM_INVALID_PATH'
_process_file_requestUSER : Some of the provided protocols are supported proceeding
_process_file_requestUSER : Protocol check failed received some unsupported protocols
ns1__srmCheckPermission : Request: Check permission. IP: ::IP. Client DN: PATH surl(s): SURL
rpcResponseHandler_AbortFiles : arrayOfFileStatuses not specified by BE.
Result for request 'Check permission' is 'SRM_NOT_SUPPORTED'
storm::BolStatusRequest::loadFromDB() : No tokens found for token REQ_ID and the requested SURLs
```

Figure 6.11: Clusters population for a random sample of 90% of the FE server log file (02/12/2018), with similarity threshold 0.7. Every bar is labelled with the cluster's reference message.

6.3 Backend log clustering

Since the algorithm was built to be employable on different log files with little to no manual intervention, it was tested on a different type of log from StoRM: the logs from the BE. Details on this type of logs are shown in Section 4.3.2.

A week of log files from the StoRM host related to the CMS experiment was analysed, in particular the week studied was the one from 22/05/2019 to 28/05/2019. In this week there was a noticeable trend in the log files as shown in Table 6.11. In particular, the file from one specific day was marked as critical on the Table, as it had a considerably larger size than the rest of the log files.

The clustering performed followed the same analysis discussed in the previous section: the similarity measure used to clusterise the log files is the Levenshtein similarity; to determine the correct threshold, the mean cluster variance and the running time of the algorithm were evaluated for a single day; the results are shown in Figure B.1 and B.2 in Appendix B.

The similarity threshold selected was 0.6. The plot shows that the mean cluster variance decreases form 0.6 to 0.7, but the mean cluster variance at lower percentile is almost the same; this may indicate that the 4 cluster difference between the clusters created with similarity threshold 0.6 and 0.7 are between the least populated, including less than 5% of the log lines, and probably results from the split of already low populated clusters.

The plots from the clustering with 0.6 similarity threshold are shown in Appendix B.1. The Figure 6.12 corresponds to the critical day, where the file size passed 4 GB, with the same plot in logarithmic scale in Figure 6.13 to highlight the distribution of the least populated clusters.

6.3.1 Results

By comparing the results from the critical day in Figure 6.13 with those shown in Appendix B.1 one could observe how the main difference between the two files, aside from the number of log lines, is not the distribution of the most populated clusters, but the occurrence of multiple low populated clusters, either not present

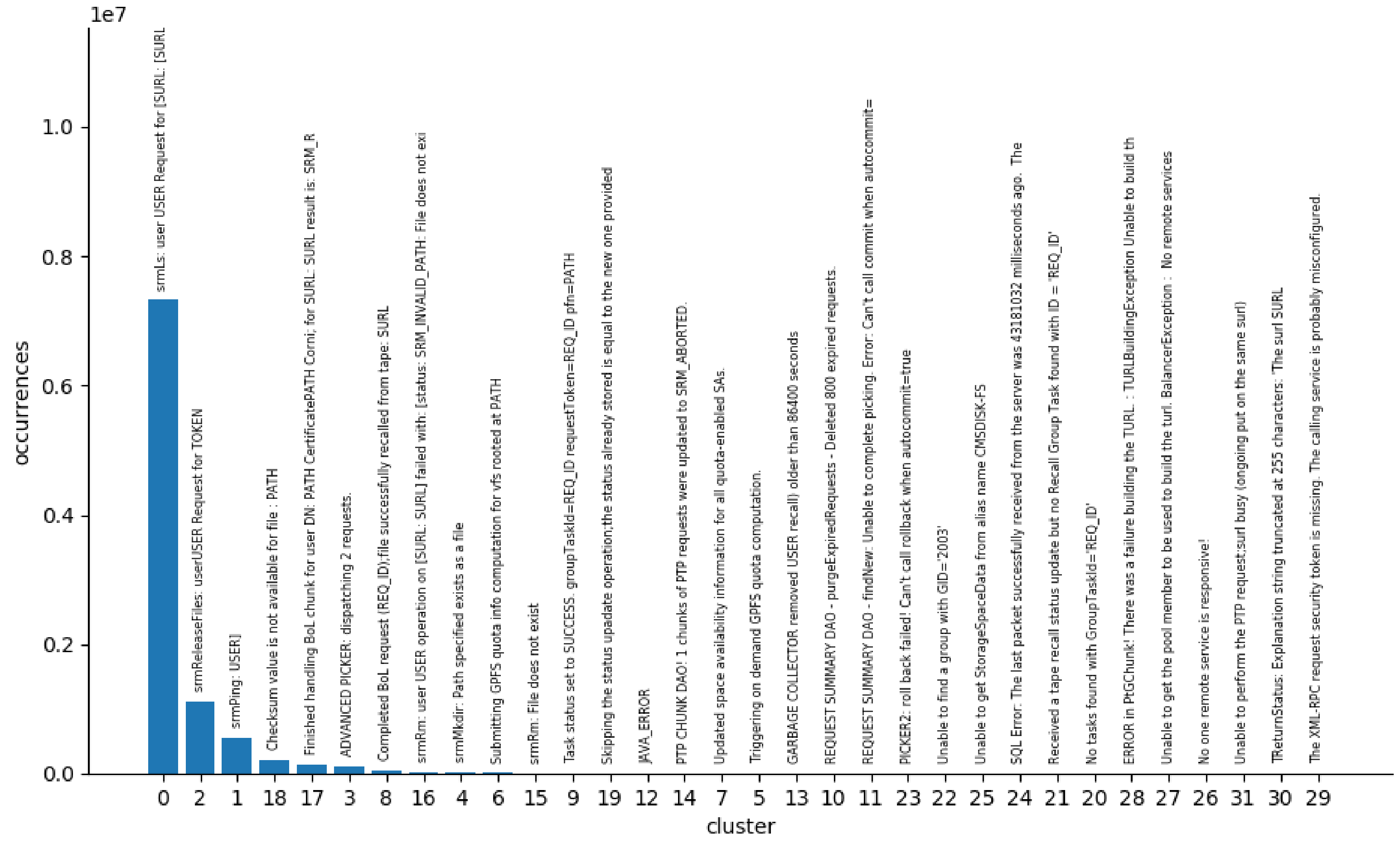


Figure 6.12: Clusters population for BE server critical log file (23/05/2019), with similarity threshold 0.65. Every bar is labelled with the cluster's reference message.

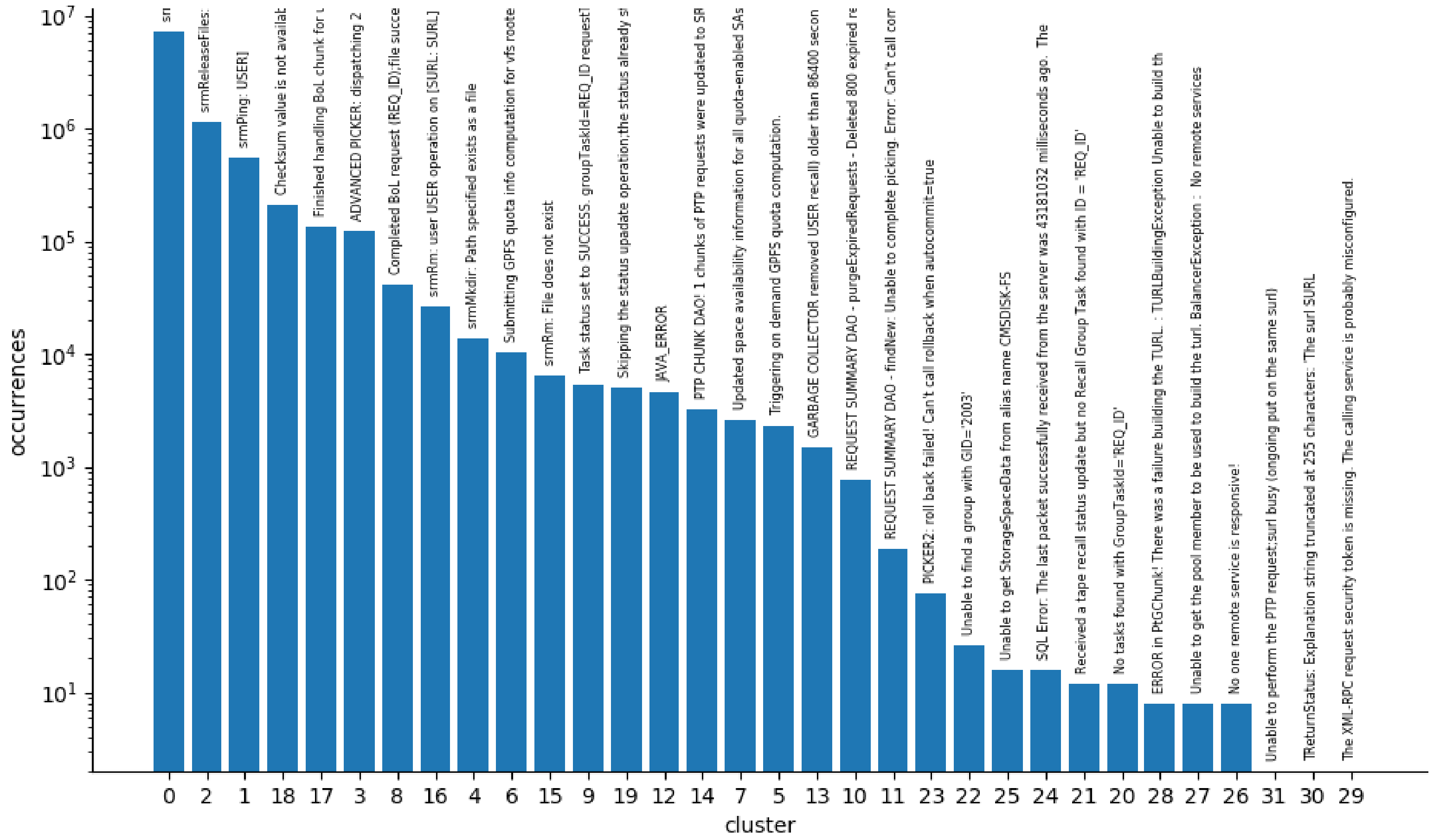


Figure 6.13: Clusters population for BE server critical log file (23/05/2019) with similarity threshold 0.6, in logarithmic scale. Details in the text.

Filename	Date	Size
2019-05-22-storm-backend.log	22/05/2019	1.541 GB
2019-05-23-storm-backend.log	23/05/2019	4.061 GB
2019-05-24-storm-backend.log	24/05/2019	1.756 GB
2019-05-25-storm-backend.log	25/05/2019	1.197 GB
2019-05-26-storm-backend.log	26/05/2019	1.263 GB
2019-05-27-storm-backend.log	27/05/2019	1.707 GB
2019-05-28-storm-backend.log	28/05/2019	1.568 GB

Table 6.11: File size of the log files from StoRM CMS's BE in the week from 22/05/2019 to 28/05/2019. The highlighted row marks the critical day. Details in the text.

in the other days, or scarcely populated.

This type of information, that can be provided each morning, summarising the most relevant anomalies in a log file can be incredibly valuable for the support team. While normally requiring to manually study the log files, an analytics software like this can provide a quick overview of the previous days.

To be informed of an anomaly the support team needs receive a ticket from a user — that usually does not provide any information on the source of the problem — or be aware of a critical failure in the system that can result in unnecessary downtime. A system such the one described in the previous Section will be able to provide an automatic warning when the clustering of the log files from the previous days varies from the usual trend.

While this type of analysis still does not provide information on the source of the problem or its gravity, can noticeably ease the debugging process by reducing the size of the data to analyse, and can be a useful approach to prepare the logging data for successive ML efforts.

The following Section will present some possible future steps to take to improve on this work.

6.4 Future developments

This work presented an experimental approach towards an unsupervised analytics tool to parse and clusterise verbose log files. The developed system proved to be able to handle different type of logs at a sufficiently high speed while yielding summarised and valuable information that improves the standard one obtainable from tedious and time-consuming human parsing of large logs.

Regarding the algorithms, the one used in this thesis showed to perform as necessary to a large extent. There are remarkable points to be improved, though, which should be part of the continuation of this work in the future. As an example, the definition and use of a cluster's reference message in the cluster creation procedure has been shown not to induce any particular bias but this choice could be improved in favour of designing one that by construction cannot induce biases.

This method can also be used to prepare data for further application of ML techniques. Algorithms such as those tested in [57], for example, require data structured into array. A clustering algorithm such as the one described in this chapter would allow to classify the information from the log files into a simple one hot encoding, where the cluster id is represented as a state in a vector where every value is a 0 except a single 1 in correspondence of the cluster id.

Visualisation is another area in which relatively easy improvements could be achieved. In the previous Section, we presented a way to represent the results of the clustering through a bar plot, that quickly showed the population distribution in the cluster and the corresponding reference messages. Another representation could be by visualising the log message at time of logging, by plotting the time on the x-axis of a scatter plot and the clusters id on the y-axis. An example of the resulting graph is visible in Figure 6.14. This kind of representation highlights the frequency of the more common clusters and showcase the order of appearance of the least populated. By studying this type of graph, it is possible to define patterns in the order of appearance of the logs. This type of analysis is performed by [68] and [71], where the repetition of series of sequential clusters is used to predict failure in error logs. Similar improvements in visualisation methods could be an asset in future directions to be pursued.

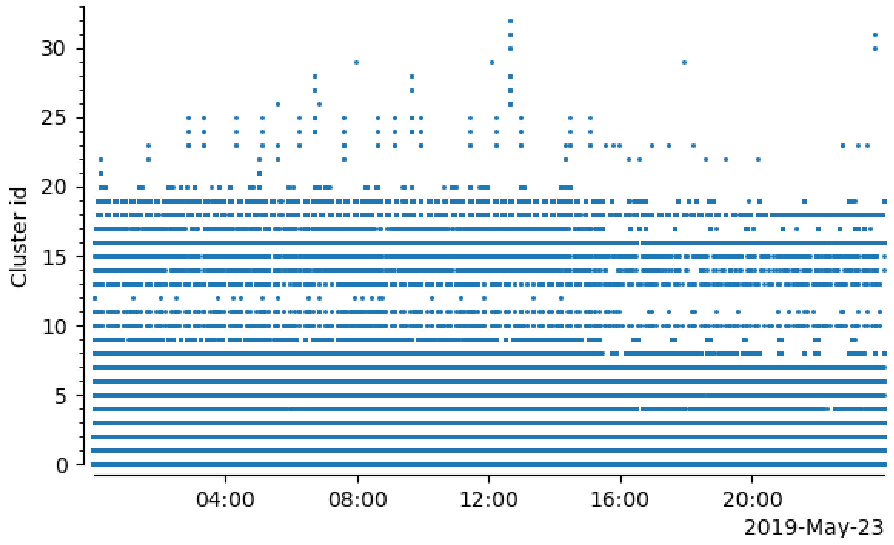


Figure 6.14: Log messages at time of logging, visualised after the cluster creation process. The y-axis refers to the clusters id generated by the cluster creation algorithm. The x-axis is the time. Every dot in the graph represents an actual message from the log, showing its mapping to one of the discovered clusters.

Another aspect that could bring improvements is the study of correlation across days (or even across log files). On the other hand, any study of correlation between the results of two different runs of the procedure is far from trivial, since the results are unlikely to share the same clusters, in the same order, or even with the same reference message. For example, one could consider computing the correlation as a measure of similarity between the reference message of each cluster in order to overcome this shortcoming. An example of this was actually investigated, and it is shown in Figure 6.15. In Figure 6.15 it is apparent how some clusters are strictly correlated, such as cluster 0 from the log file of 7/12/2018 and cluster 1 from 6/12/2018. From Table A.6 and A.7 is obvious how the two clusters follow the same exact template; cluster 14 from the 7/12/2018, instead, does not seem to be similar to any clusters from 6/12/2018. The Table indeed confirms that this log message did not occur on the previous day.

As a final remark, another direction to improve the results from this work could be adapting the code to run on a distributed platform. The framework presented

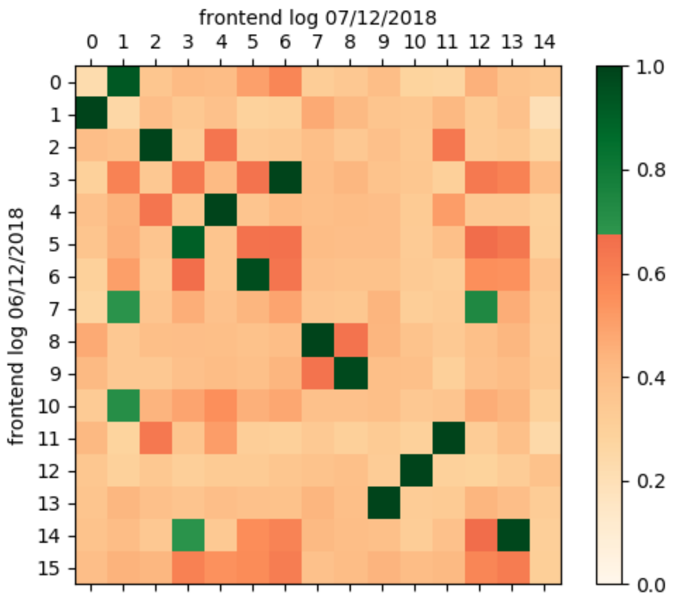


Figure 6.15: Correlation between clusters created from two different log days measured as the similarity between their reference message. Lighter red shades for similarities lower than the threshold of 0.675, darker greens for similarities higher than the threshold.

in Chapter 5 can run parallelised software through the use of the Spark engine and YARN. The code developed in this work would have a large benefit in performance if adapted for such purpose, but the effort is large i.e. most of the functions should be rewritten to follow the Map Reduce programming model. The program clusterises the log file line by line, as the clustering cannot be vectorised, since it depends on the results from the previous lines, but the measure of similarity is performed independently for each cluster and, as such, it can be a perfect target for parallelisation. This is considered to be a non trivial but largely promising investment of efforts for CNAF in the future.

All the mentioned improvements, some of which have already been explored in a preliminary manner, stand as concrete suggestions to any future effort to continue the work of this thesis, but go beyond the scope of the thesis itself. The thesis focused on the development of a preliminary infrastructure for analytics at CNAF on one side, and an experimental analytics algorithm for text-based log analysis on the other side. It is worth noting that part of the work suggested above is already being worked on by either CNAF developers or new students (both in Physics and in Data Science and Computation). It is crucial to underline that the INFN-CNAF centre is strongly interested in progressing on this project and in supporting efforts towards further developing more powerful and versatile monitoring systems able to automatically correlate log files and metrics coming from heterogeneous sources and devices [72].

Conclusions

This thesis presents the work done at the INFN-CNAF Tier-1 Centre aimed at the deployment of a platform for log ingestion and processing oriented towards the development of algorithms and procedures towards a Predictive Maintenance model for anomaly detection at the data centre. The original contribution of this work is the development of a clustering algorithm based on measures of text similarity that, after parsing and anonymising a verbose event log file, classifies each log lines into community according to their level of similarity.

After testing different text-based similarity measures, an algorithm based on the Levenshtein distance was selected as the one offering the highest number of significant clusters, while keeping the running time of the clustering algorithm below an acceptable threshold. A careful evaluation was performed to determine the best values for the clustering parameters, selecting the Minimum Similarity Threshold that resulted in the lowest mean cluster variance, both for every cluster and for just the most populated ones. This is done to ensure that the clusters have enough resolution to maintain information, while avoiding unnecessary separations.

The clustering algorithm has been run on two different type of log files, belonging to the two main components of the Storage Resource Manager (StoRM) of the INFN-CNAF centre. As a particular example, a week of log files of the backend StoRM server from the CMS experiment was analysed, as it presented an anomalously large log file size; the algorithm allowed to spot discrepancies in the number and type of clusters identified, enabling a clear visualisation of the anomalous behaviour to assist the service support team with an effective tool.

The results obtained prove that the algorithm satisfies the expected requirements: it follows a totally unsupervised procedure with no need of any manual intervention to classify the log lines; it is applicable to different types of log with minimum adjustments; it offers a good cluster resolution with low loss of log messages information; it scales well with the log file size, performing the clusterisation in less than 6-8 hours for the average CNAF log on a standard, medium range laptop.

This thesis also presents a distributed analysis platform based primarily on Apache and Openstack software that is being developed as part of the same efforts at CNAF towards Predictive Maintenance. As possible future development of this work, the algorithm could be adapted to run on such platform to increase its performance and open up to the possibility of streaming the log data directly into the clustering procedure.

Software like this could be of great help to the support team of the various services at CNAF, opening the path towards the implementation of various autonomous routines to perform maintenance actions, requiring less and less effort and time from the operators.

Appendices

Appendix A

Frontend plots

A.1 Measures comparison

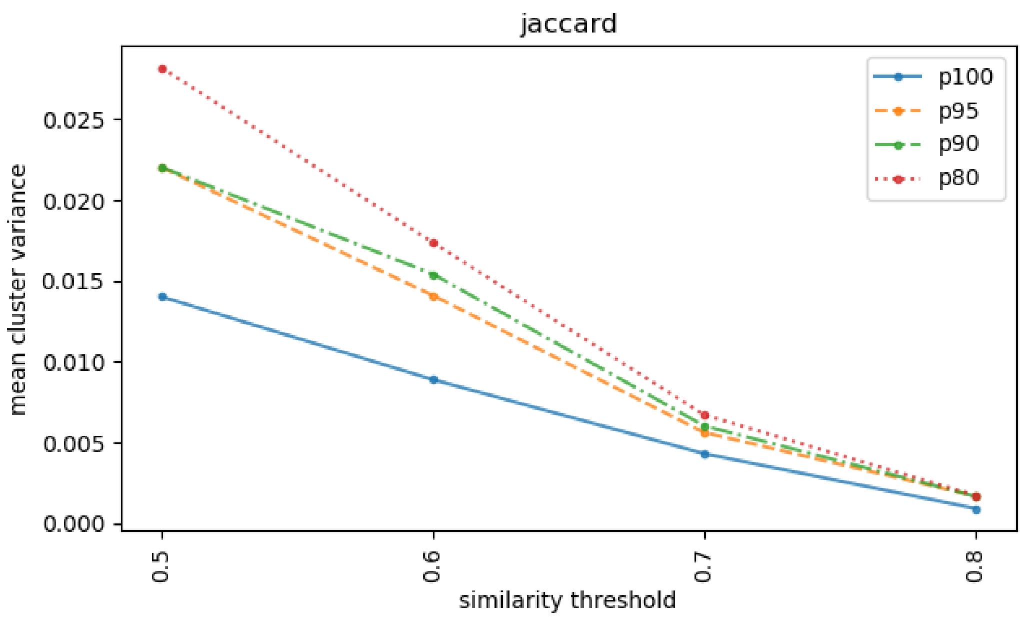


Figure A.1: Mean variance in clusters for Jaccard similarity applied on a sample file of 20000 lines from FE.

Similarity threshold	Clusters	Slope
0.50	17	4129
0.60	26	3763
0.70	37	3224
0.80	65	2511

Table A.1: Number of clusters and iteration per second for Jaccard similarity with different similarity thresholds on sample file of 20000 lines from FE.

Similarity threshold	Clusters	Slope
0.50	4	298
0.60	8	170
0.70	13	115
0.80	24	59

Table A.2: Number of clusters and iteration per second for Cosine similarity with different similarity thresholds on sample file of 20000 lines from FE.

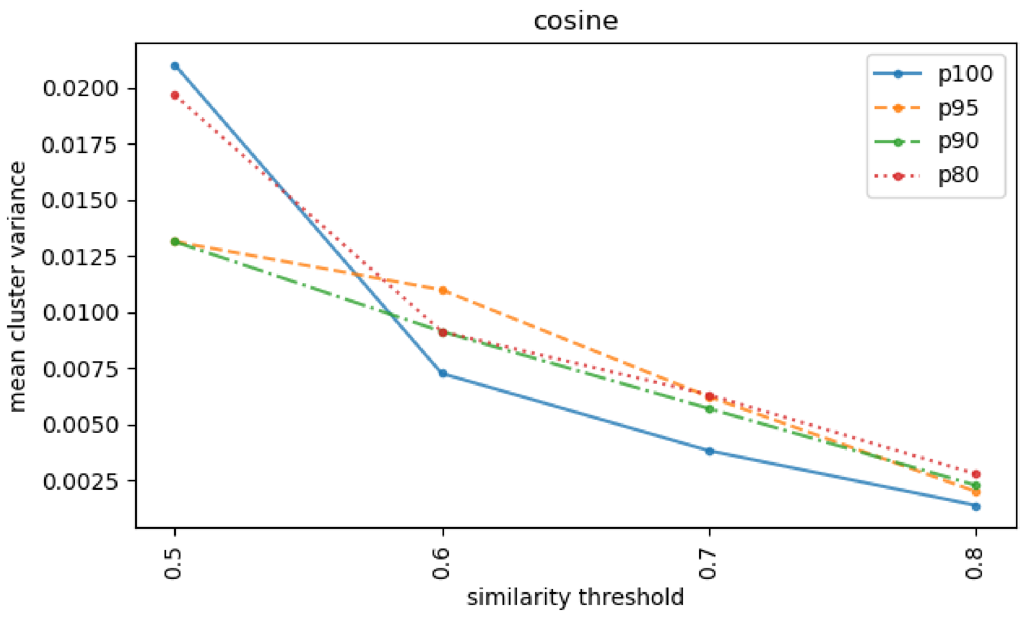


Figure A.2: Mean variance in clusters for Cosine similarity applied on a sample file of 20000 lines from FE.

Similarity threshold	Clusters	Slope
0.50	4	3471
0.60	6	2791
0.70	9	1846
0.80	19	966

Table A.3: Number of clusters and iteration per second for Levenshtein similarity with different similarity thresholds on sample file of 20000 lines from FE.

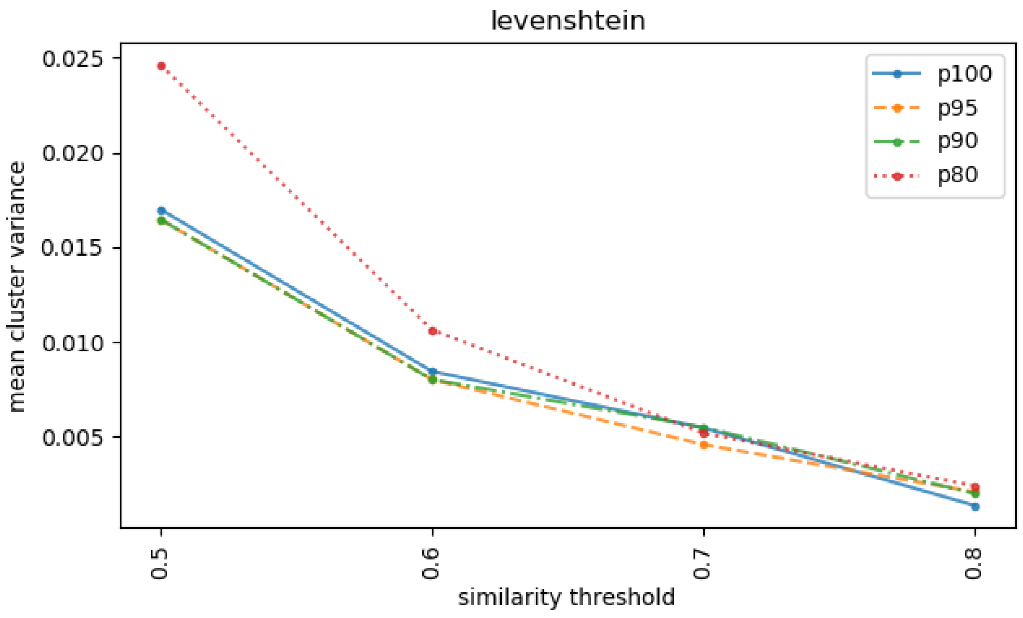


Figure A.3: Mean variance in clusters for Levenshtein similarity applied on a sample file of 20000 lines from FE.

A.2 Time trend

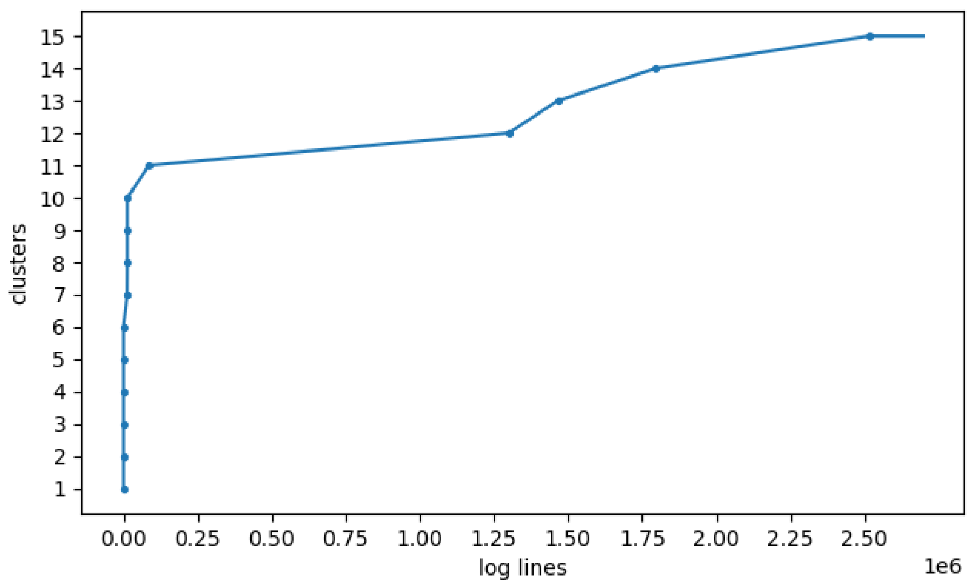


Figure A.4: Number of clusters as a function of lines per log, in FE server log (06/12/2018) with similarity threshold = 0.7.

Similarity threshold	Clusters	Slope
0.40	5	2338
0.50	7	1964
0.60	8	1761
0.62	9	1684
0.65	14	1267
0.68	16	1148
0.70	19	1027
0.80	34	610
0.90	73	328

Table A.4: Total number of clusters and slope of the fit line for the evaluation time of the FE server log (06/12/2018) at different similarity threshold.

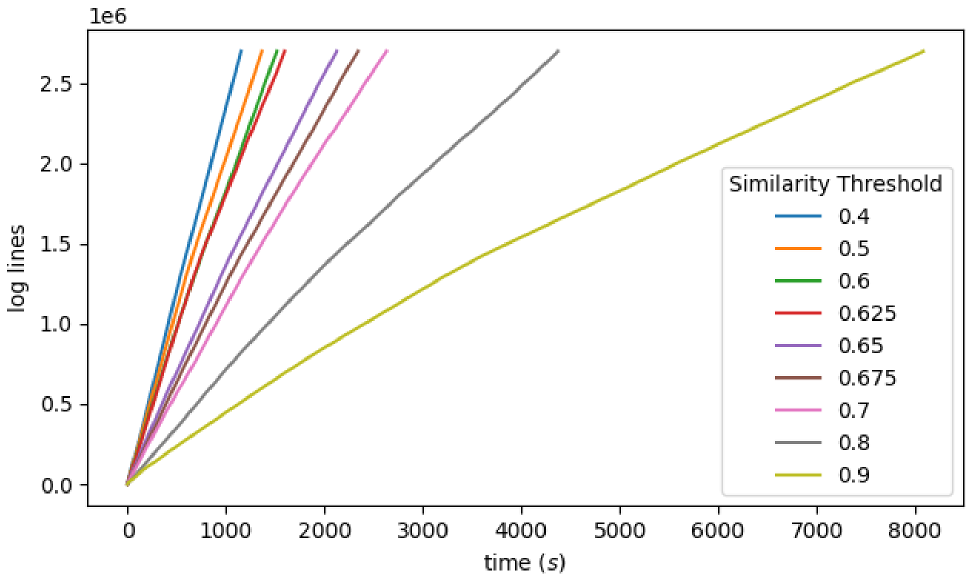


Figure A.5: Number of log lines clustered as a function of the running time for the FE server log (06/12/2018). The slope of the curve decreases for higher similarity threshold.

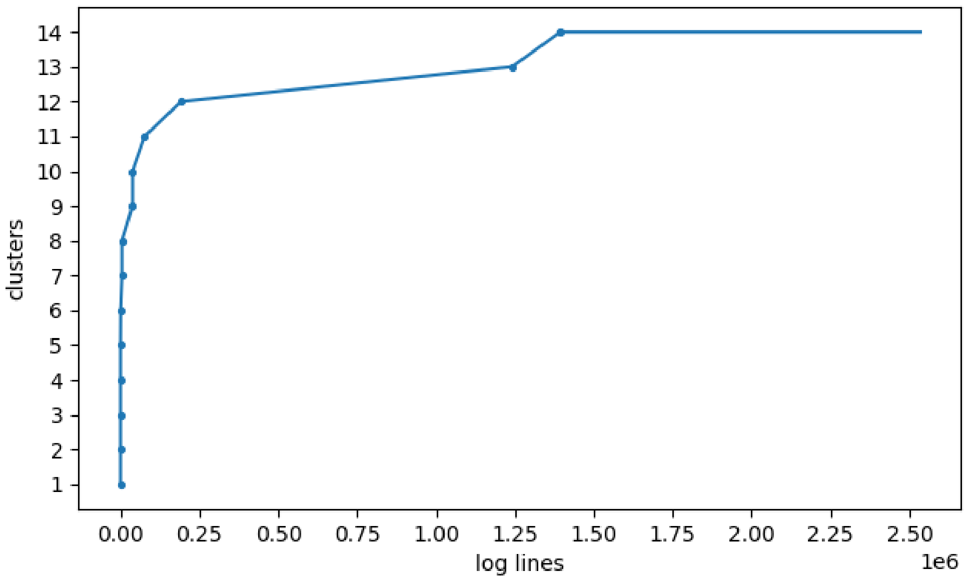


Figure A.6: Number of clusters as a function of lines per log, in FE server log (07/12/2018) with similarity threshold = 0.7.

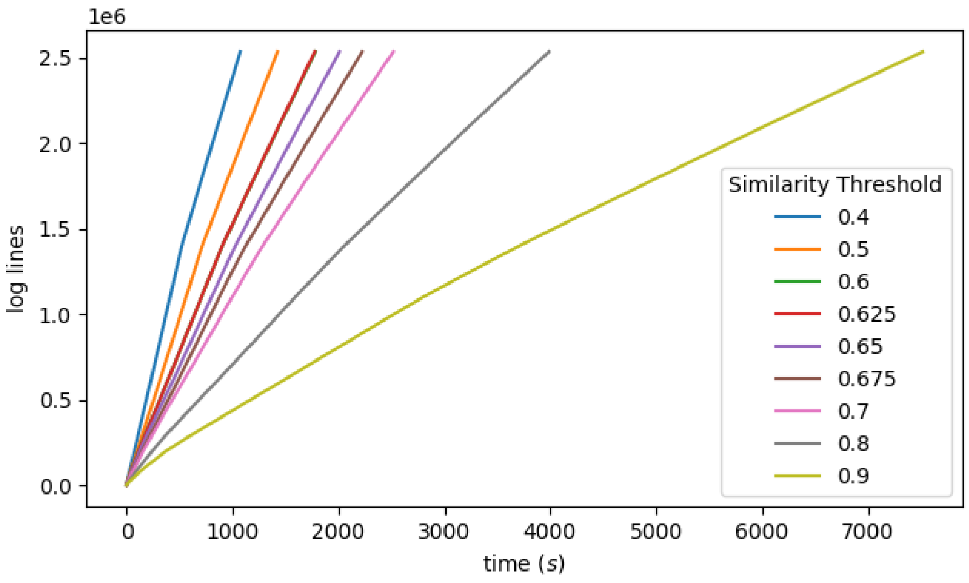


Figure A.7: Number of log lines clustered as a function of the running time for the FE server log (07/12/2018). The slope of the curve decreases for higher similarity threshold.

Similarity threshold	Clusters	Slope
0.40	5	2373
0.50	8	1787
0.60	10	1418
0.62	10	1424
0.65	13	1255
0.68	15	1138
0.70	18	999
0.80	34	629
0.90	72	331

Table A.5: Total number of clusters and slope of the fit line for the evaluation time of the FE server log (07/12/2018) at different similarity thresholds.

A.3 Mean cluster variance

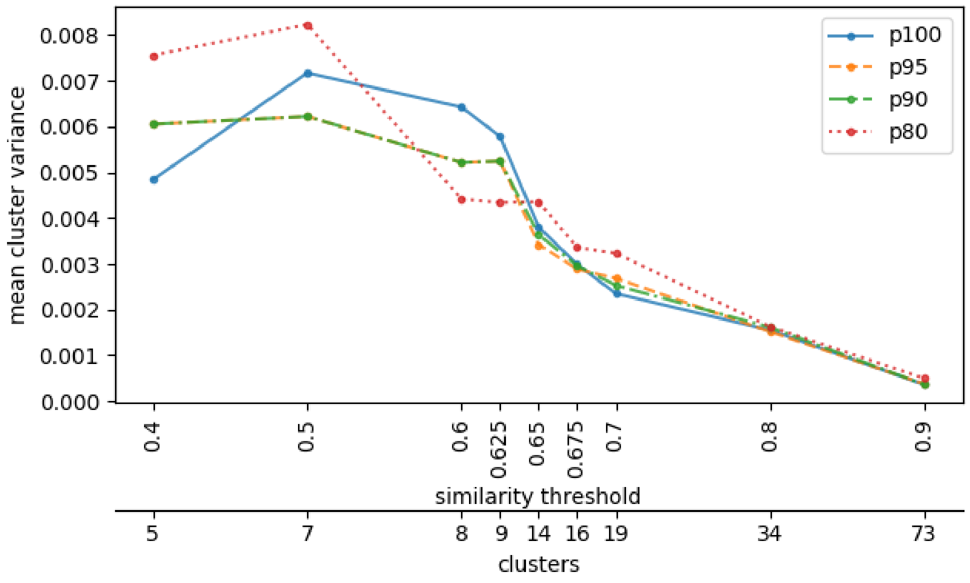


Figure A.8: Mean cluster variance at different similarity levels for the entire log file (p100) for the 95th percentile (p95) for the 90th percentile (p90) and the 80th percentile (p80) from the FE server log (06/12/2018).

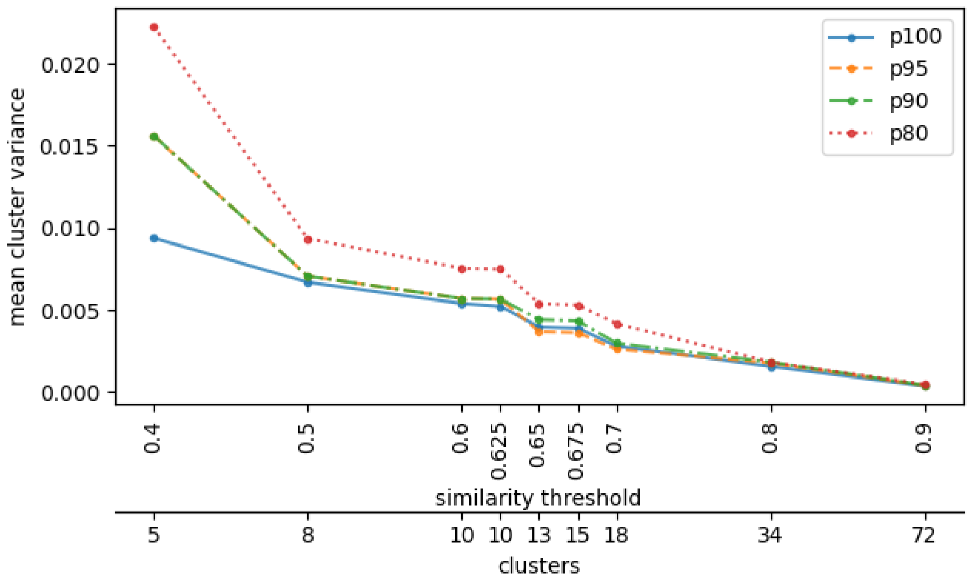


Figure A.9: Mean cluster variance at different similarity levels for the entire log file (p100) and for lower percentiles (p80, p90, p95) from the FE server log (07/12/2018).

A.4 Sampling

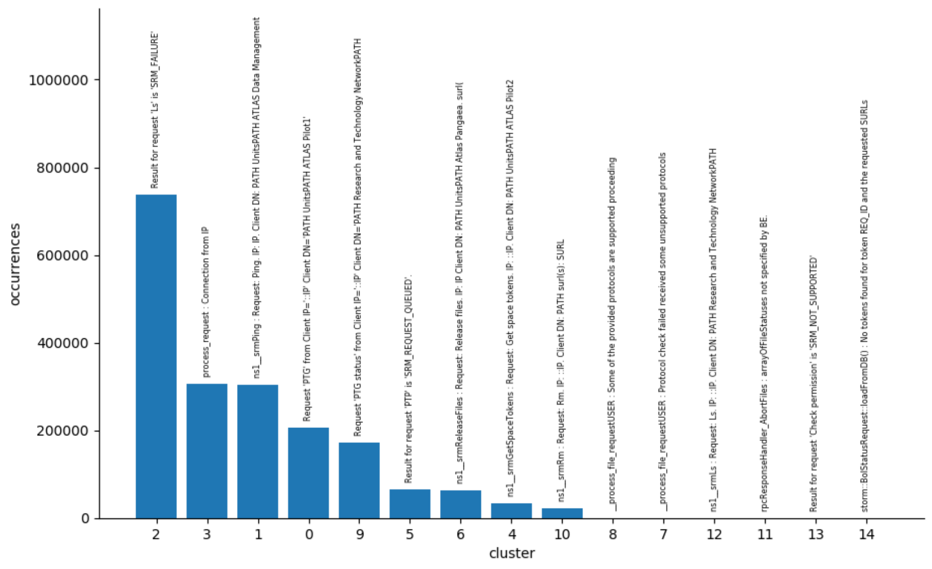


Figure A.10: Clusters population for a second random sample of 90% of the FE log file (02/12/2018), with similarity threshold 0.7.

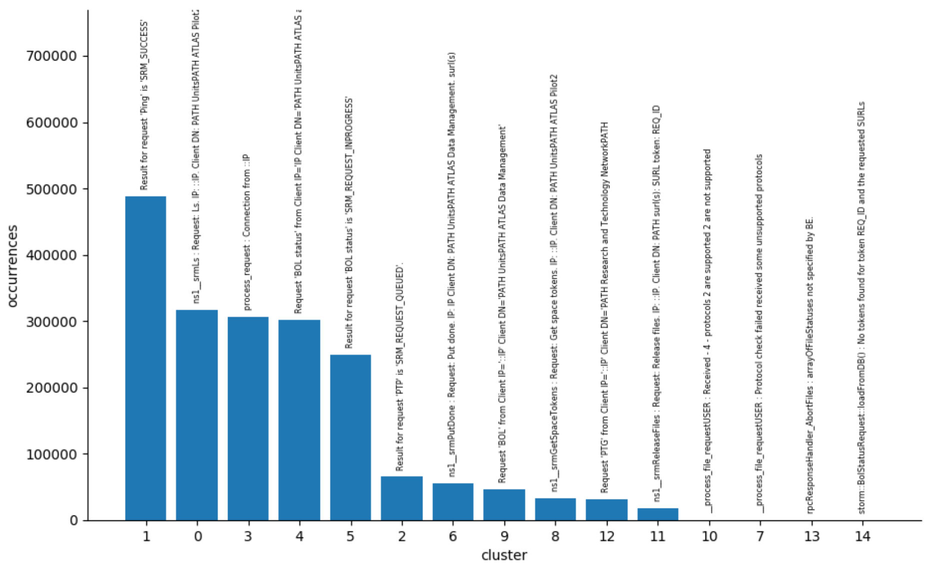


Figure A.11: Clusters population for a third random sample of 90% of the FE log file (02/12/2018), with similarity threshold 0.7.

A.5 Clusters histograms and tables

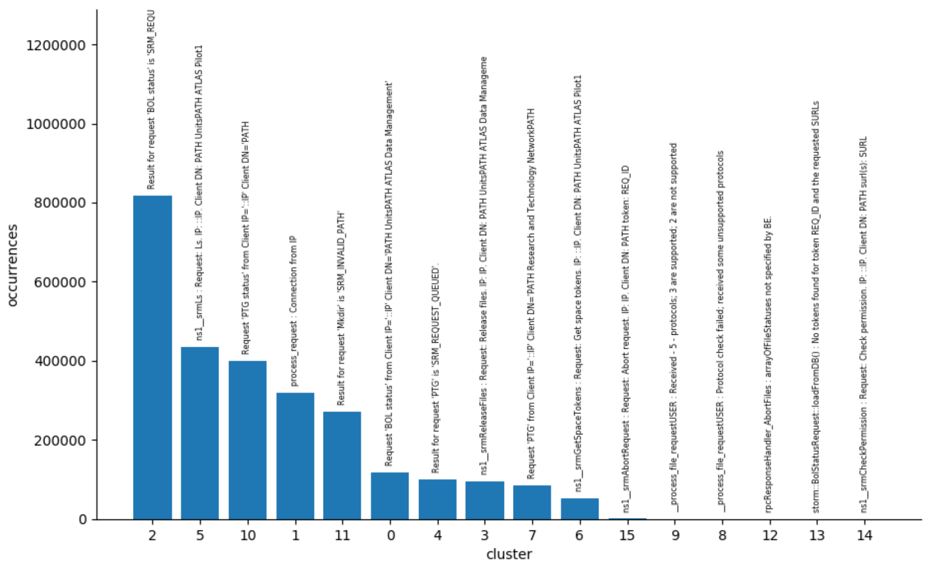


Figure A.12: Clusters population for FE server log file (06/12/2018), with similarity threshold 0.675. Every bar is labelled with the cluster's reference message.

Cluster ID	Occurrences	Reference message
0	118644	Request 'BOL status' from Client IP='::IP' Client [...]
1	320314	process_request : Connection from IP
2	817928	Result for request 'BOL status' is 'SRM_REQUEST_IN[...]
3	94810	ns1__srmReleaseFiles : Request: Release files. IP:[...]
4	99434	Result for request 'PTG' is 'SRM_REQUEST_QUEUED'. [...]
5	435536	ns1__srmLs : Request: Ls. IP: ::IP. Client DN: PAT[...]
6	52069	ns1__srmGetSpaceTokens : Request: Get space tokens[...]
7	84846	Request 'PTG' from Client IP='::IP' Client DN='PAT[...]
8	509	__process_file_requestUSER : Protocol check failed[...]
9	1018	__process_file_requestUSER : Received - 5 - protoc[...]
10	401040	Request 'PTG status' from Client IP='::IP' Client [...]
11	270841	Result for request 'Mkdir' is 'SRM_INVALID_PATH'
12	62	rpcResponseHandler_AbortFiles : arrayOfFileStatuse[...]
13	59	storm::BolStatusRequest::loadFromDB() : No tokens [...]
14	26	ns1__srmCheckPermission : Request: Check permissio[...]
15	1233	ns1__srmAbortRequest : Request: Abort request. IP:[...]

Table A.6: Cluster population and reference log for FE server log file (06/12/2018) with similarity threshold 0.675.

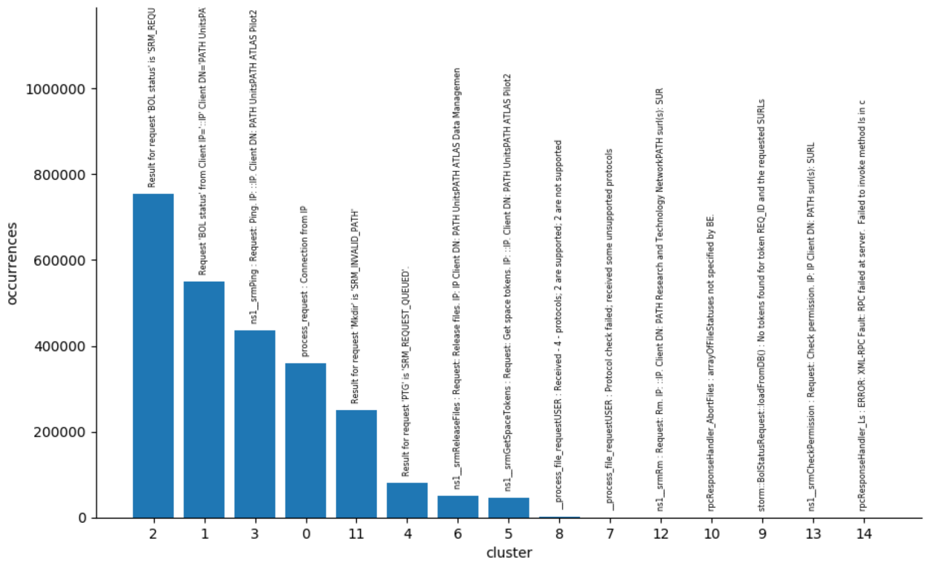


Figure A.13: Clusters population for FE server log file (07/12/2018), with similarity threshold 0.675. Every bar is labelled with the cluster's reference message.

Cluster ID	Occurrences	Reference message
0	359469	process_request : Connection from IP
1	550170	Request 'BOL status' from Client IP='::IP' Client [...]
2	754453	Result for request 'BOL status' is 'SRM_REQUEST_IN[...]
3	437153	ns1__srmPing : Request: Ping. IP: ::IP. Client DN:[...]
4	80704	Result for request 'PTG' is 'SRM_REQUEST_QUEUED'. [...]
5	47215	ns1__srmGetSpaceTokens : Request: Get space tokens[...]
6	51154	ns1__srmReleaseFiles : Request: Release files. IP:[...]
7	901	__process_file_requestUSER : Protocol check failed[...]
8	1802	__process_file_requestUSER : Received - 4 - protoc[...]
9	80	storm::BolStatusRequest::loadFromDB() : No tokens [...]
10	122	rpcResponseHandler_AbortFiles : arrayOfFileStatuse[...]
11	250777	Result for request 'Mkdir' is 'SRM_INVALID_PATH'
12	216	ns1__srmRm : Request: Rm. IP: ::IP. Client DN: PAT[...]
13	28	ns1__srmCheckPermission : Request: Check permissio[...]
14	23	rpcResponseHandler_Ls : ERROR: XML-RPC Fault: RPC [...]

Table A.7: Cluster population and reference log for FE server log file (07/12/2018) with similarity threshold 0.675.

Appendix B

Backend plots

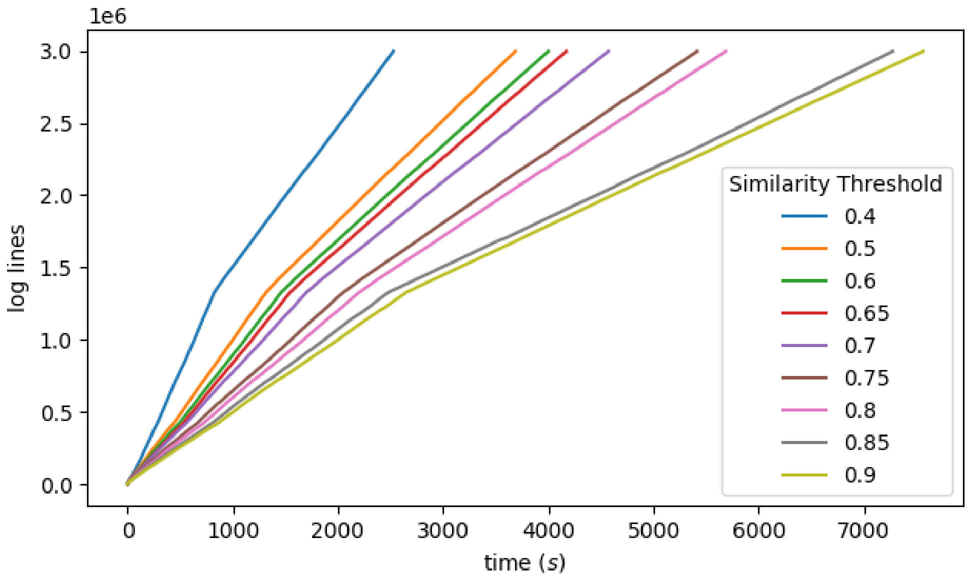


Figure B.1: Number of log lines clustered as a function of the running time for the BE server log (25/05/2019). The slope of the curve decreases for higher similarity thresholds.

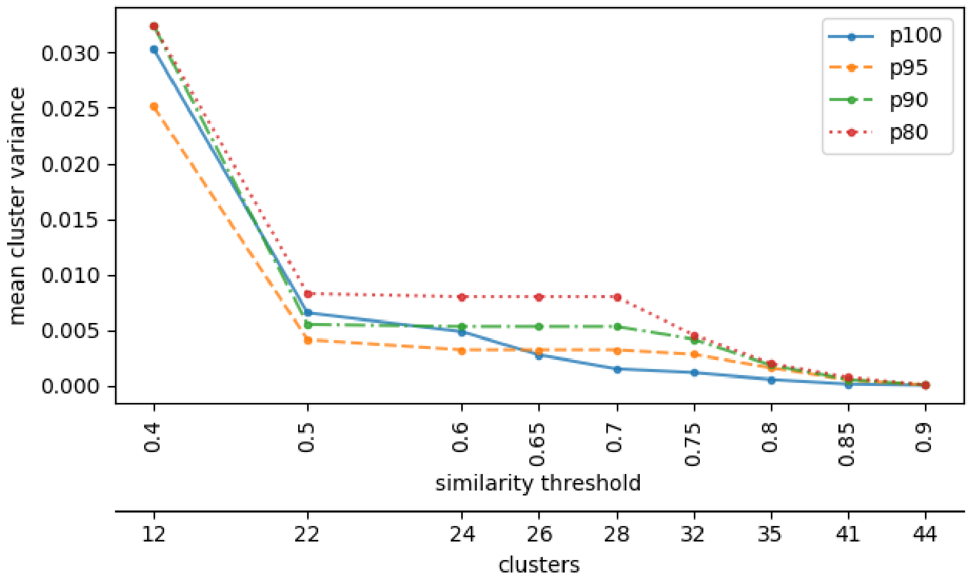


Figure B.2: Mean cluster variance at different similarity levels for the entire log file (p100) for the 95th percentile (p95) for the 90th percentile (p90) and the 80th percentile (p80) from the BE server log (25/05/2019)

B.1 Clusters histograms

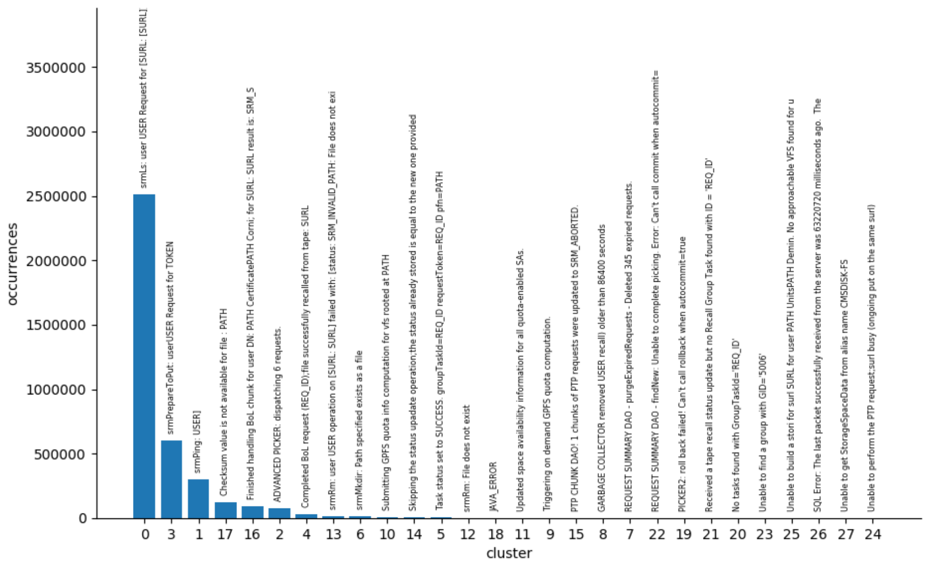


Figure B.3: Clusters population for BE log file (22/05/2019), with similarity threshold 0.6.

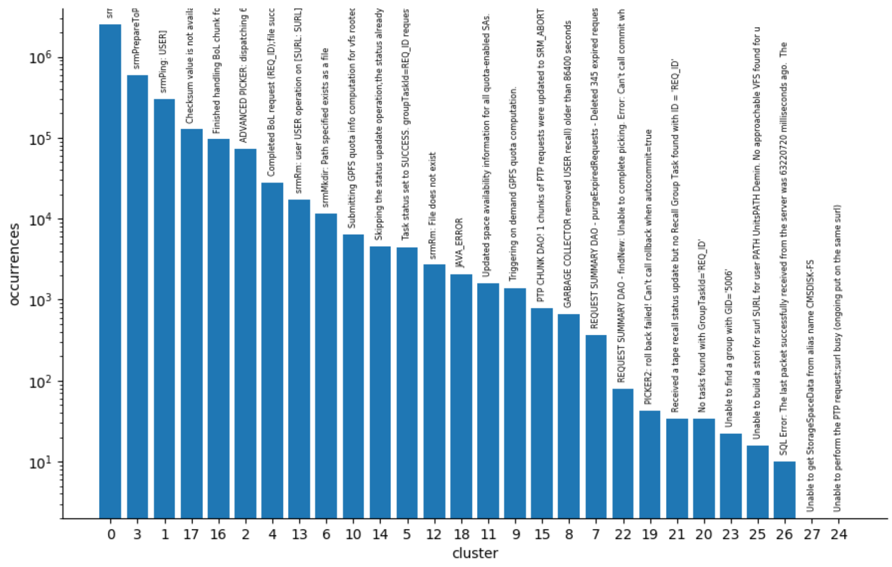


Figure B.4: Clusters population for BE log file (22/05/2019), with similarity threshold 0.6. Y-axis in logarithmic scale to highlight the least populated clusters.

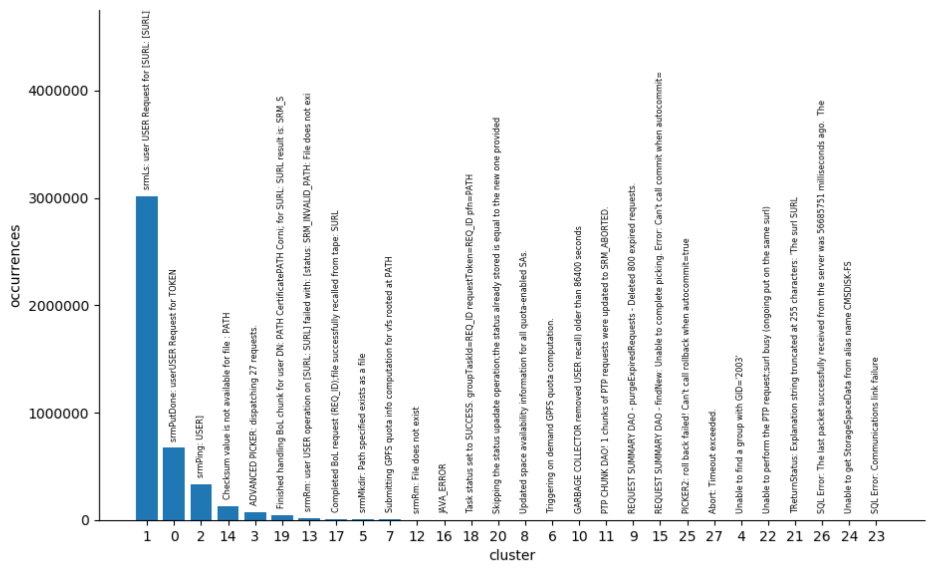


Figure B.5: Clusters population for BE log file (24/05/2019), with similarity threshold 0.6.

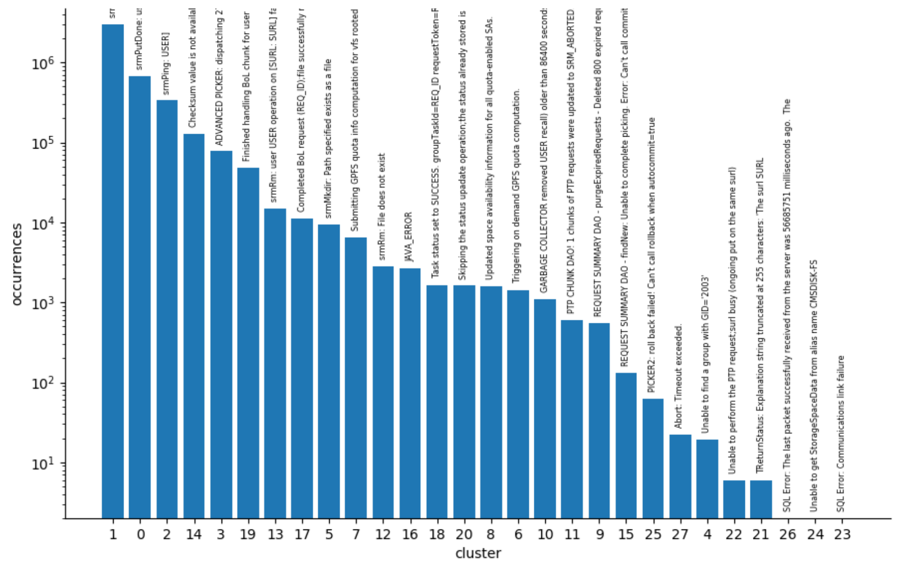


Figure B.6: Clusters population for BE log file (24/05/2019), with similarity threshold 0.6. Y-axis in logarithmic scale to highlight the least populated clusters.

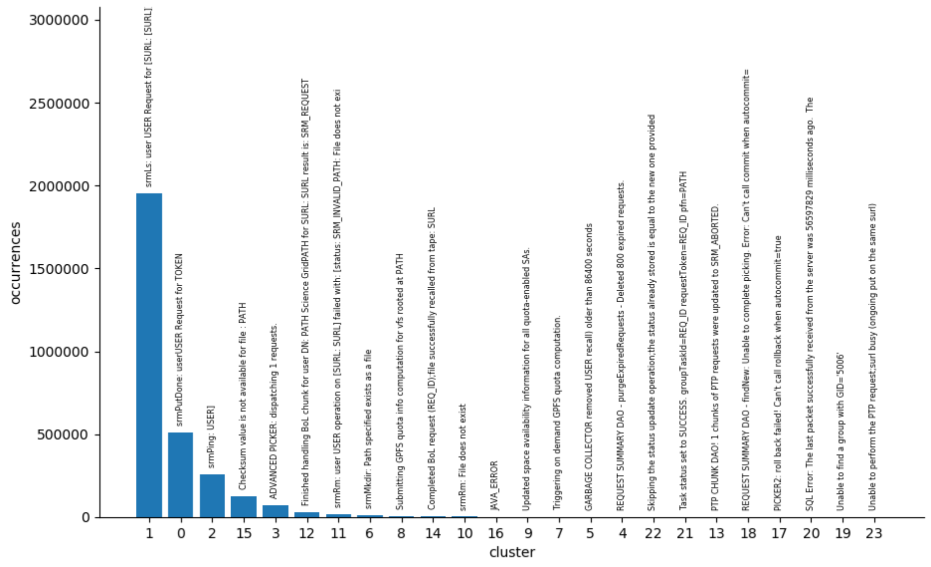


Figure B.7: Clusters population for BE log file (25/05/2019), with similarity threshold 0.6.

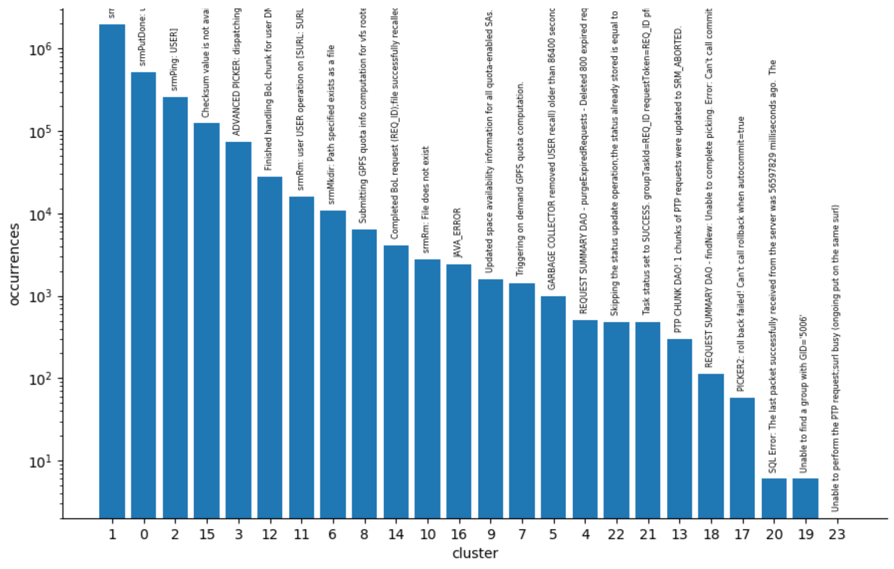


Figure B.8: Clusters population for BE log file (25/05/2019), with similarity threshold 0.6. Y-axis in logarithmic scale to highlight the least populated clusters.

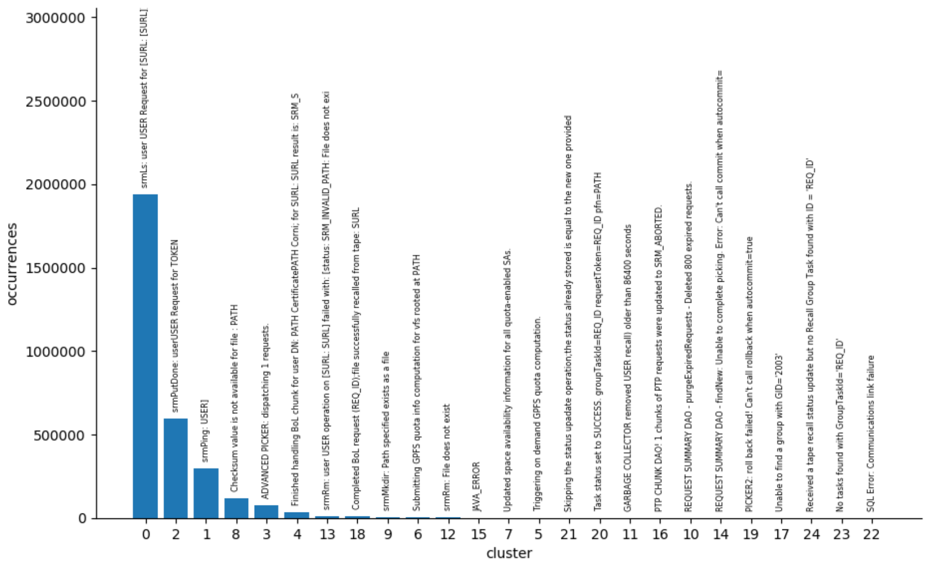


Figure B.9: Clusters population for BE log file (26/05/2019), with similarity threshold 0.6.

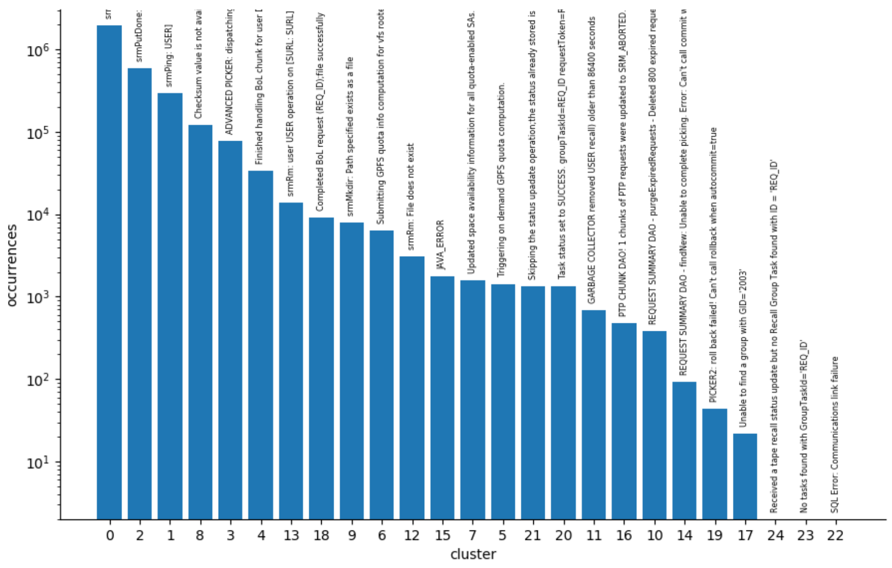


Figure B.10: Clusters population for BE log file (26/05/2019), with similarity threshold 0.6. Y-axis in logarithmic scale to highlight the least populated clusters.

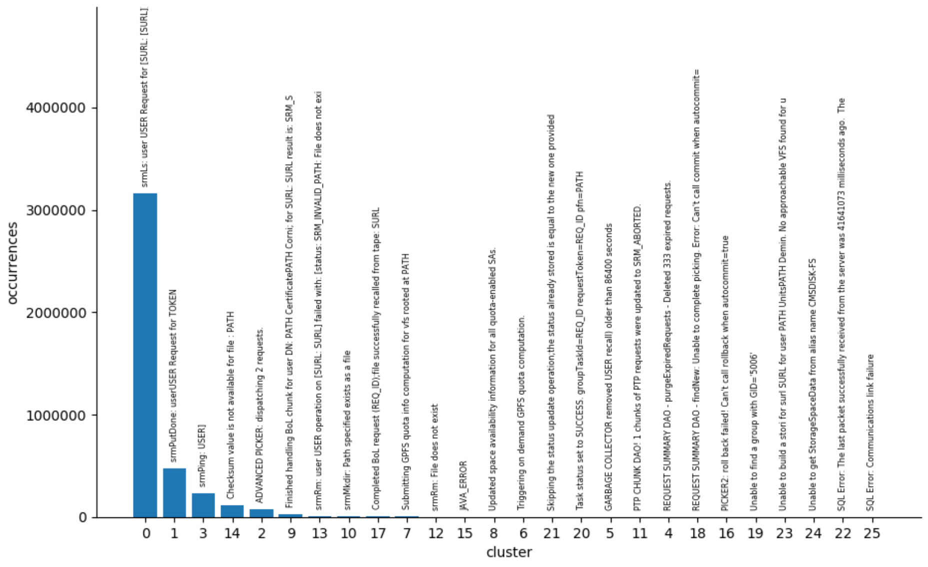


Figure B.11: Clusters population for BE log file (27/05/2019), with similarity threshold 0.6.

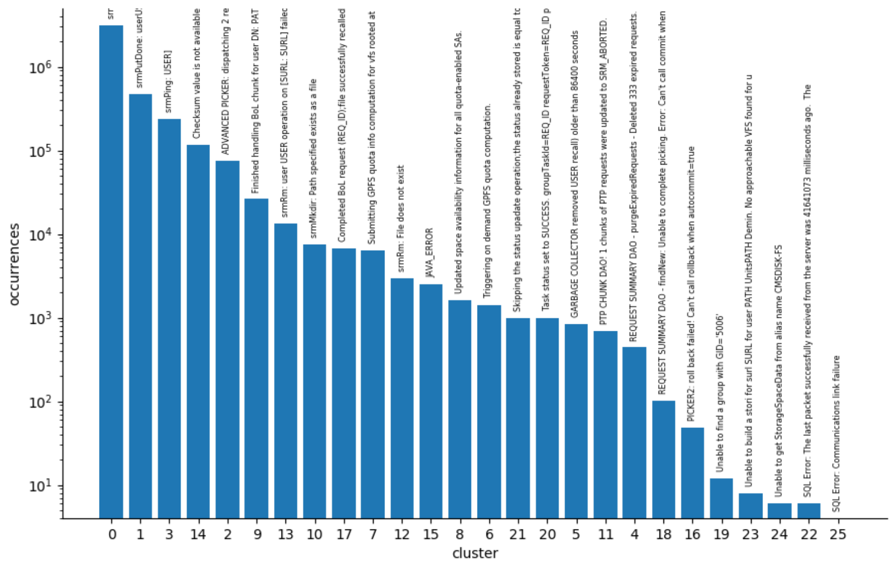


Figure B.12: Clusters population for BE log file (27/05/2019), with similarity threshold 0.6. Y-axis in logarithmic scale to highlight the least populated clusters.

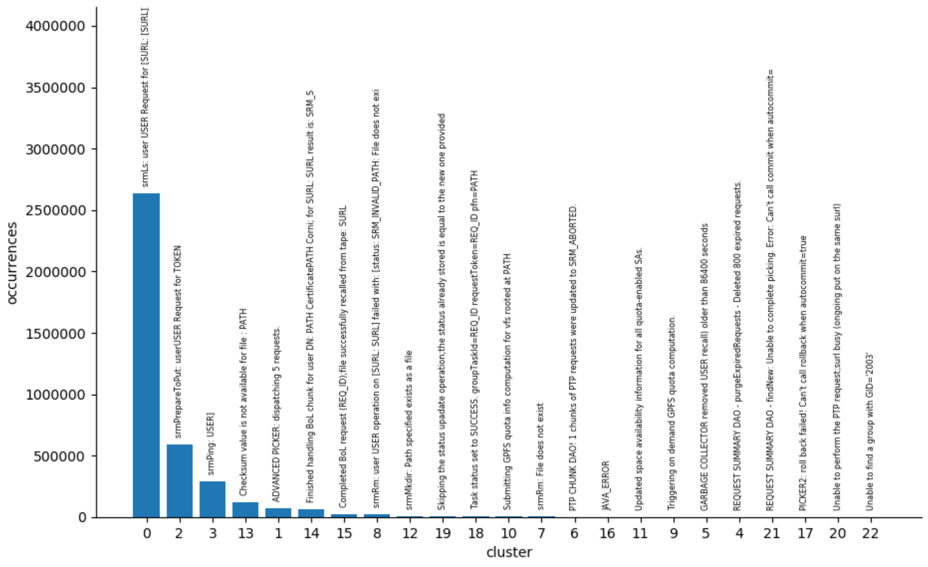


Figure B.13: Clusters population for BE log file (28/05/2019), with similarity threshold 0.6.

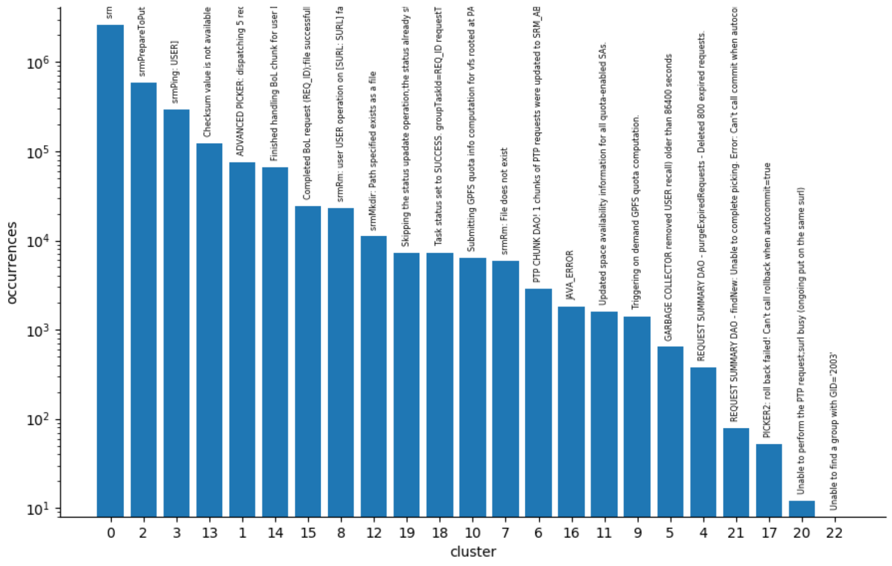


Figure B.14: Clusters population for BE log file (28/05/2019), with similarity threshold 0.6. Y-axis in logarithmic scale to highlight the least populated clusters.

Bibliography

[1] *The HL-LHC Project.* Apr. 2019. URL: http://hilumilhc.web.cern.ch/ (cit. on p. 8).

[2] *HEP Software Foundation.* URL: https://hepsoftwarefoundation.org/ (cit. on p. 8).

[3] The HEP Software Foundation et al. "A Roadmap for HEP Software and Computing R&D for the 2020s". In: *Computing and Software for Big Science* 3.1 (Mar. 2019), p. 7. ISSN: 2510-2044. DOI: 10.1007/s41781-018-0018-8. URL: https://doi.org/10.1007/s41781-018-0018-8 (cit. on p. 8).

[4] Amir Gandomi and Murtaza Haider. "Beyond the hype: Big data concepts, methods, and analytics". In: *International Journal of Information Management* 35.2 (Apr. 2015), pp. 137–144. DOI: 10.1016/j.ijinfomgt.2014.10.007. URL: https://doi.org/10.1016/j.ijinfomgt.2014.10.007 (cit. on pp. 9, 12).

[5] D Laney. *3-D Data Management: Controlling Data Volume, Velocity and Variety. META Gr. Res.* Tech. rep. Note 6, 70, 2001 (cit. on p. 9).

[6] *The Four V's of Big Data.* URL: https://www.ibmbigdatahub.com/infographic/four-vs-big-data (cit. on pp. 10, 11).

[7] *Oracle Big Data.* URL: https://www.oracle.com/big-data/guide/what-is-big-data.html (cit. on p. 10).

[8] *CERN's IT gears up to face the challenges of LHC Run 2.* URL: https://cerncourier.com/cerns-it-gears-up-to-face-the-challenges-of-lhc-run-2/ (cit. on p. 12).

[9] Alexander Radovic et al. "Machine learning at the energy and intensity frontiers of particle physics". In: *Nature* 560.7716 (2018), pp. 41–48. ISSN: 1476-4687. DOI: 10.1038/s41586-018-0361-2. URL: https://doi.org/10.1038/s41586-018-0361-2 (cit. on p. 14).

[10] F. Pedregosa et al. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830 (cit. on pp. 14, 62).

[11] Martń Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: https://www.tensorflow.org/ (cit. on p. 14).

[12] François Chollet et al. *Keras*. https://keras.io. 2015 (cit. on p. 14).

[13] Adam Paszke et al. "Automatic differentiation in PyTorch". In: *NIPS-W*. 2017 (cit. on p. 14).

[14] *ELK Stack*. URL: https://www.elastic.co/elk-stack (cit. on pp. 15, 54).

[15] K. Shvachko et al. "The Hadoop Distributed File System". In: *2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)*. Apr. 2010, pp. 1–10. DOI: 10.1109/MSST.2010.5496972 (cit. on pp. 15, 48).

[16] *CERN - Accellerating Science*. URL: http://wlcg.web.cern.ch/ (cit. on pp. 15, 17).

[17] *HEP Software Foundation*. URL: https://hepsoftwarefoundation.org/ (cit. on p. 16).

[18] *EGI Advanced Computing Services for Research*. URL: https://www.egi.eu/ (cit. on p. 17).

[19] *Open Science Grid*. URL: https://opensciencegrid.org/ (cit. on p. 17).

[20] Les Robertson. "Computing Services for LHC: From Clusters to Grids". In: *From the Web to the Grid and Beyond: Computing Paradigms Driven by High-Energy Physics*. Ed. by René Brun, Federico Carminati, and Giuliana Galli Carminati. Springer Berlin Heidelberg, 2012, pp. 69–89. DOI: 10.1007/978-3-642-23157-5_3 (cit. on p. 18).

[21] *The MONARC Project.* URL: https://monarc.web.cern.ch/MONARC/ (cit. on pp. 18, 19).

[22] em9228. URL: http://lhcopn.web.cern.ch/lhcopn/ (cit. on p. 19).

[23] Epifani. *Istituto Nazionale di Fisica Nucleare.* URL: http://home.infn.it/it/ (cit. on p. 20).

[24] *CNAF.* URL: https://www.cnaf.infn.it/ (cit. on pp. 20–22).

[25] *StoRM.* URL: https://italiangrid.github.io/storm/index.html (cit. on pp. 21, 41).

[26] *GARR.* URL: https://www.garr.it/en/ (cit. on p. 22).

[27] J. Moubray. *Reliability-centered Maintenance.* Industrial Press, 1997. ISBN: 9780831131463. URL: https://books.google.it/books?id=bNCVF0B7vpIC (cit. on pp. 25, 28).

[28] Flavio Trojan and Rui F M Marçal. "Proposal of Maintenance-types Classification to Clarify Maintenance Concepts in Production and Operations Management". In: *JBE* 8 (July 2017). DOI: 10.15341/jbe(2155-7950)/07.08.2017/005 (cit. on pp. 26, 27).

[29] Kevin F Gallimore and Richard J Penlesky. "A framework for developing maintenance strategies". In: *Production and Inventory Management Journal* 29.1 (1988), p. 16. URL: https://search.proquest.com/docview/199902491?accountid=9652 (cit. on pp. 27–29).

[30] Alessio Bonifetti. "Predictive and Prescriptive Maintenance". In: *REM.* 2018. URL: http://www.remenergy.it/admin/wp-content/uploads/08-REM2018_Bonfietti_MindIT.pdf (cit. on p. 27).

[31] Alina Sirbu and Ozalp Babaoglu. "Towards operator-less data centers through data-driven, predictive, proactive autonomics". In: *Cluster Computing* 19.2 (Apr. 2016), pp. 865–878. DOI: 10.1007/s10586-016-0564-y (cit. on p. 27).

[32] Su Chuan-Jun and Jorge A. Quan Yon. "Big Data Preventive Maintenance for Hard Disk Failure Detection". In: *IJIET* 8.7 (2018), pp. 471–481. DOI: 10.18178/ijiet.2018.8.7.1085 (cit. on pp. 29, 30).

[33] Mirela Madalina Botezatu et al. “Predicting disk replacement towards reliable data centers”. In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM. 2016, pp. 39–48 (cit. on p. 30).

[34] Somnath Mazumdar and Anoop S Kumar. “Forecasting Data Center Resource Usage: An Experimental Comparison with Time-Series Methods”. In: *International Conference on Soft Computing and Pattern Recognition*. Springer. 2016, pp. 151–165 (cit. on p. 30).

[35] You-Luen Lee et al. “DC-Prophet: Predicting Catastrophic Machine Failures in DataCenters”. In: *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer. 2017, pp. 64–76 (cit. on p. 30).

[36] Ruben Sipos et al. “Log-based predictive maintenance”. In: *Proceedings of the 20th ACM SIGKDD international conference on knowledge discovery and data mining*. ACM. 2014, pp. 1867–1876 (cit. on pp. 30, 31).

[37] Felix Salfner and Steffen Tschirpke. “Error Log Processing for Accurate Failure Prediction”. In: *Proceedings of the First USENIX Conference on Analysis of System Logs*. WASL'08. San Diego, California: USENIX Association, 2008, pp. 4–4. URL: http://dl.acm.org/citation.cfm?id=1855886.1855890 (cit. on pp. 31, 63).

[38] Shenglin Zhang et al. “Syslog processing for switch failure diagnosis and prediction in datacenter networks”. In: *2017 IEEE/ACM 25th International Symposium on Quality of Service (IWQoS)*. IEEE. 2017, pp. 1–10 (cit. on p. 31).

[39] R. Keith Mobley. *An Introduction to Predictive Maintenance (Plant Engineering)*. Butterworth-Heinemann, 2002. URL: https://www.sciencedirect.com/book/9780750675314/an-introduction-to-predictive-maintenance (cit. on p. 31).

[40] Lauren Milechin et al. “D4M 3.0: Extended database and language capabilities”. In: *2017 IEEE High Performance Extreme Computing Conference*

(HPEC). IEEE, Sept. 2017. DOI: 10.1109/hpec.2017.8091083 (cit. on p. 32).

[41] Kurt Matyas et al. "A procedural approach for realizing prescriptive maintenance planning in manufacturing industries". In: *CIRP Annals* 66.1 (2017), pp. 461–464. ISSN: 0007-8506. DOI: https://doi.org/10.1016/j.cirp.2017.04.007 (cit. on p. 33).

[42] Fazel Ansari, Robert Glawar, and Wilfried Sihn. "Prescriptive Maintenance of CPPS by Integrating Multimodal Data with Dynamic Bayesian Networks". In: *Machine Learning for Cyber Physical Systems*. Ed. by Jurgen Beyerer, Alexander Maier, and Oliver Niggemann. Berlin, Heidelberg: Springer Berlin Heidelberg, 2019, pp. 1–8. ISBN: 978-3-662-59084-3 (cit. on p. 33).

[43] *The rocket-fast Syslog Server*. URL: https://www.rsyslog.com/ (cit. on p. 38).

[44] *InfluxDB 1.7*. URL: https://docs.influxdata.com/influxdb/v1.7/ (cit. on p. 38).

[45] *Grafana - The open platform for analytics and monitoring*. URL: https://grafana.com/ (cit. on p. 38).

[46] *Network File System (NFS) Version 4 Protocol*. Tech. rep. Mar. 2015. DOI: 10.17487/rfc7530. URL: https://doi.org/10.17487/rfc7530 (cit. on p. 39).

[47] Arne Wiebalck. *(R)Evolution in the CERN IT Department: A 5 year perspective on the A...* Nov. 2017. URL: https://www.slideshare.net/ArneWiebalck/revolution-in-the-cern-it-department-a-5-year-perspective-on-the-agile-infrastructure-project-bonn22nov2017 (cit. on p. 46).

[48] *Build the future of Open Infrastructure*. URL: https://www.openstack.org/ (cit. on p. 47).

[49] *DODAS*. URL: https://dodas-ts.github.io/dodas-doc/ (cit. on p. 48).

[50] *INDIGO DataCloud.* URL: https://www.indigo-datacloud.eu/ (cit. on p. 48).

[51] URL: https://www.eosc-hub.eu/catalogue/Dynamic%20On%20Demand%20Analysis%20Service (cit. on p. 48).

[52] Vinod Kumar Vavilapalli et al. "Apache hadoop yarn: Yet another resource negotiator". In: *Proceedings of the 4th annual Symposium on Cloud Computing*. ACM. 2013, p. 5 (cit. on p. 49).

[53] *Apache Flume.* URL: https://flume.apache.org/ (cit. on p. 49).

[54] *Apache Kafka.* URL: https://kafka.apache.org/ (cit. on p. 50).

[55] Gwen Shapira and Jeff Holoman. *Flafka: Apache Flume Meets Apache Kafka for Event Processing.* Sept. 2015. URL: https://blog.cloudera.com/blog/2014/11/flafka-apache-flume-meets-apache-kafka-for-event-processing/ (cit. on p. 50).

[56] Matei Zaharia et al. "Apache Spark: A Unified Engine for Big Data Processing". In: *Commun. ACM* 59.11 (Oct. 2016), pp. 56–65. ISSN: 0001-0782. DOI: 10.1145/2934664. URL: http://doi.acm.org/10.1145/2934664 (cit. on p. 50).

[57] Luca Giommi et al. "Towards Predictive Maintenance with Machine Learning at the INFN-CNAF computing centre". In: *Proceedings of the International Symposium on Grids & Clouds 2019.* ISGC2019 (cit. on pp. 54, 82).

[58] Tommaso Diotalevi et al. "Collection and harmonization of system logs and prototypal Analytics services with the Elastic (ELK) suite at the INFN-CNAF computing centre". In: *Proceedings of the International Symposium on Grids & Clouds 2019.* ISGC2019 (cit. on p. 54).

[59] Simone Rossi Tisbeni and Luca Giommi. *log-parsing repository.* URL: https://baltig.infn.it/summerstudentscnaf/log-parsing (cit. on p. 56).

[60] Travis Oliphant. *NumPy: A guide to NumPy.* USA: Trelgol Publishing. [Online; accessed ¡today¿]. 2006. URL: http://www.numpy.org/ (cit. on p. 59).

[61] Wes McKinney. “Data Structures for Statistical Computing in Python”. In: *Proceedings of the 9th Python in Science Conference*. Ed. by Stéfan van der Walt and Jarrod Millman. 2010, pp. 51–56 (cit. on p. 59).

[62] Simone Rossi Tisbeni. *log-clustering repository*. June 2019. URL: `https://github.com/rsreds/Clustering` (cit. on p. 59).

[63] Guillermo Navarro-Arribas et al. “User k-anonymity for privacy preserving data mining of query logs”. In: *Information Processing & Management* 48.3 (2012). Soft Approaches to IA on the Web, pp. 476–487. ISSN: 0306-4573. DOI: `https://doi.org/10.1016/j.ipm.2011.01.004` (cit. on pp. 61, 63).

[64] Suphakit Niwattanakul et al. “Using of Jaccard coefficient for keywords similarity”. In: *Proceedings of the international multiconference of engineers and computer scientists*. Vol. 1. 6. 2013, pp. 380–384 (cit. on p. 61).

[65] Claudio Lucchese et al. “Identifying task-based sessions in search engine query logs”. In: *Proceedings of the fourth ACM international conference on Web search and data mining*. ACM. 2011, pp. 277–286 (cit. on p. 61).

[66] Anna Huang. “Similarity measures for text document clustering”. In: *Proceedings of the sixth new zealand computer science research student conference (NZCSRSC2008), Christchurch, New Zealand*. Vol. 4. 2008, pp. 9–56 (cit. on p. 62).

[67] Rada Mihalcea, Courtney Corley, Carlo Strapparava, et al. “Corpus-based and knowledge-based measures of text semantic similarity”. In: *AAAI*. Vol. 6. 2006, pp. 775–780 (cit. on p. 62).

[68] Michal Aharon et al. “One Graph Is Worth a Thousand Logs: Uncovering Hidden Structures in Massive System Event Logs”. In: *Machine Learning and Knowledge Discovery in Databases*. Ed. by Wray Buntine et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 227–243. ISBN: 978-3-642-04180-8 (cit. on pp. 62, 82).

[69] Rosie Jones and Kristina Lisa Klinkner. “Beyond the session timeout: automatic hierarchical segmentation of search topics in query logs”. In: *Proceedings of the 17th ACM conference on Information and knowledge management*. ACM. 2008, pp. 699–708 (cit. on p. 63).

[70] *python-Levenshtein*. URL: `https://pypi.org/project/python-Levenshtein/` (cit. on p. 64).

[71] Purvil Bambharolia, Prajeet Bhavsar, and Vivek Prasad. “Failure Prediction And Detection In Cloud Datacenters”. In: *International Journal of Scientific & Technology Research* 6.9 (2017), pp. 122–127 (cit. on p. 82).

[72] Barbara Martelli et al. “Monitoring and Analytics at INFN Tier-1: the next step (accepted as poster)”. In: *24th International Conference on Computing in High Energy & Nuclear Physics*. Facilities, Clouds and Containers (cit. on p. 85).