Alma Mater Studiorum · Università di Bologna

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

Analysis of the log files of the StoRM storage system used by the ATLAS experiment, performed with Anomaly Detection through Deep Learning

Relatore:
Prof. Lorenzo Rinaldi

Presentata da:
Edoardo Corallo

Anno Accademico 2020/2021

Contents

Abstract

Ogni anno, l'esperimento ATLAS, come gli altri esperimenti operanti al Large Hadron Collider, produce Petabytes di dati grezzi ed elaborati da distribuitre attraverso la Worldwide LHC Computing Grid, che combina le risorse per il calcolo distribuito da più di 170 siti in 42 paesi, creando un'enorme infrastruttura per la computazione distribuita che garantisce ad oltre 12000 scienziati nel mondo un rapido acesso ai dati di LHC.
In gran parte dei siti della Worlwide LHC Computing Grid (WLCG) sono state sviluppate tecnologie basate sul Grid Computing, dedicate tra le altre cose al monitoraggio e all'organizzazione dello storage; quest'ultime, come la gran parte delle applicazioni informatiche, registrano la propria attività sotto forma di log di sistema.
Siccome ci si aspetta un notevole incremento dei dati da distribuire e processare sulle risorse della WLCG, queste strutture devono essere affidabili e l'Anomaly Detection sui log di sistema rappresenta una soluzione per migliorare l'efficienza di utilizzo dei sistemi, dal momento che le varie tipologie di log di sitema sono un'eccellente fonte di informazioni per il monitoraggio delle anomalie.

1 Introduction

In this thesis work is presented an analysis of the log files produced by StoRM, the storage system in use at CNAF INFN-T1. The log files produced by the StoRM services will be analysed using DeepLog, a software designed for the detection of system anomalies. The analysis has been performed on the log files collected during a working week, concerning the data transfer activities of the ATLAS experiment. In chapter 2 is presented the ATLAS experiment held at CERN's Large Hadron Collider. The flow of data coming from the experiments at LHC ranges in the Petabytes per year and needs a distributed infrastructure to be stored and analyzed: the Worldwide LHC's Computing Grid (WLCG), introduced in chapter 3, which is composed of multiple sites with different importance within the network. In these sites new grid technology has been developed for a multitude of tasks, including StoRM. These systems need to be as reliable as possible, since the number of accesses the WLCG resources is expected to grow in the future as the amount of data to be distributed and analyzed grows, and one way to obtain a more efficient system is through log analysis and anomaly detection on system logs, which was performed on StoRM's system logs collected in one week.
Chapter 4 contains a brief introduction on Neural Networks, focusing on Deep Learning, Recurrent Neural Networks and Long Short Term Memory finishing with the presentation of DeepLog. The 5^{th} chapter is the Log Analysis section where the methods used for the creation of the dataset are shown, along side the results of the work on the log data for StoRM's front-end and a discussion on the obtained results.

2 The ATLAS Experiment at CERN's Large Hadron Collider

The Large Hadron Collider (LHC) is the world's largest and most powerful particle accelerator. It first started up on September 10^{th} 2008, and remains the latest addition to CERN's accelerator complex.

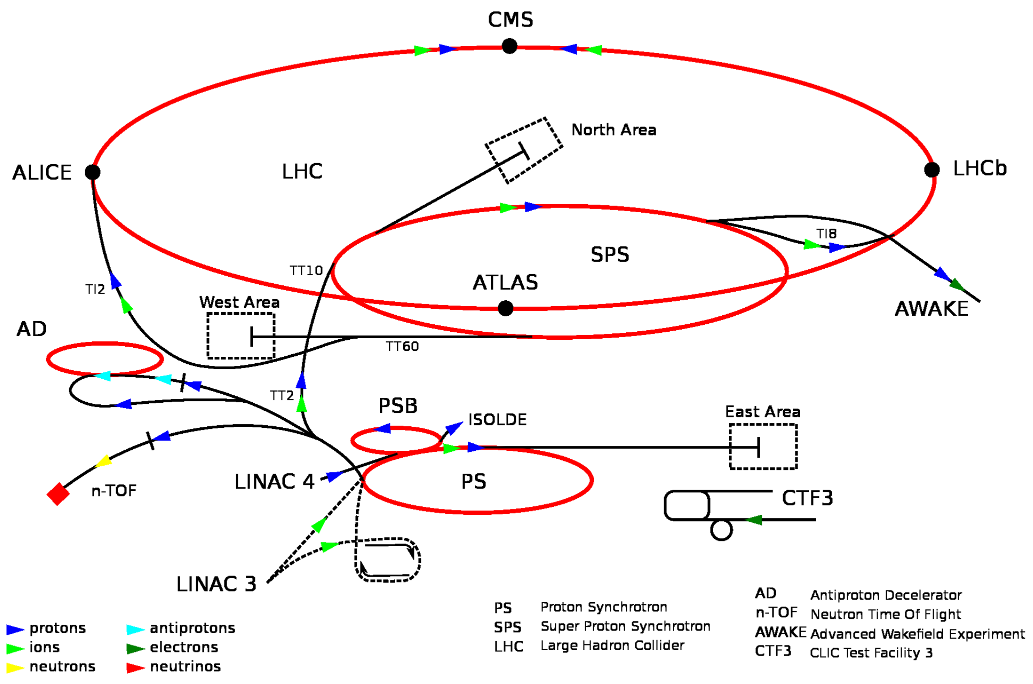


Figure 1: CERN's accelerator complex [1]

The LHC consists of a 27 kilometer ring of superconducting magnets with a number of accelerating structures to boost the energy of the particles along the way. Protons exiting Linac2, with an energy of 50 MeV, are then further accelerated up to 450 GeV in a series of Proton Synchrotron and divided in two beams and made to collide after one last boost at a center of mass energy of 13 TeV (design energy is 14 Tev). These proton beams are kept on track by strong magnetic fields generated by thousands of superconducting magnets along LHC.
There are seven experiments held at CERN's LHC and are all positioned around the four $p-p$ collision spots, where the following detectors are placed:
ALICE, A Large Ion Collider Experiment, is an experiment dedicated to the study of heavy ions collisions and is designed to study the physics of strongly interacting matter at extreme energy densities, where a phase of matter called quark-gluon plasma forms [2].
ATLAS, A Toroidal LHC ApparatuS, is one of two general-purpose detectors at LHC

and is the largest volume particle detector ever constructed. It investigates a wide range of physics, from the measurement of Higgs boson properties to extra dimensions and particles that could make up dark matter [3].
CMS, Compact Muon Solenoid, is the other multi-purpose experiment held at CERN, although it has the same research goals as ATLAS it employs different design and detectors [4].
LHCb, Large Hadron Collider beauty, is designed to investigating the differences between matter and antimatter by studying the properties of b quarks [5].
The smallest experiments on the LHC are TOTEM and LHCf, which focus on protons or heavy ions that scatter at very small angles when the beams collide and thus not detectable by the main experiments. TOTEM (TOTal cross section, Elastic scattering and diffraction dissociation Measurement at the LHC) [6] uses detectors positioned on either side of the CMS interaction point, while LHCf (Large Hadron Collider Forward) [7] is made up of two detectors which sit along the LHC beamline, at 140 meters either side of the ATLAS collision point. MoEDAL uses detectors deployed near LHCb to search for a hypothetical particle called the magnetic monopole [8].

2.1 Description of the ATLAS Detector

The ATLAS experiment [9] is designed to take advantage of the high energies achieved at LHC and to explore the physics beyond the Standard Model of Particles (BSM). The high luminosity of LHC, which makes possible to study processes with low cross section, dictates the use of fast detectors with high granularity, able to endure enormous flows of particles [10]. Within 2027 the luminosity of LHC will be upgraded with HL-LHC (High Luminosity-Large Hadron Collider) and will produce at least 15 million Higgs bosons per year, compared to around three million from the LHC in 2017 [11].
It's required to have good acceptance coverage to effectively study processes with an imbalance of the impulse of the final states due to particles that don't interact with the detector as those expected by Supersimmetry or for the study of the Dark Matter. For the latter reason a series of Calorimeters is needed to achieve precise measures of Jets and missing Energies in the plane perpendicular to the beam axis.

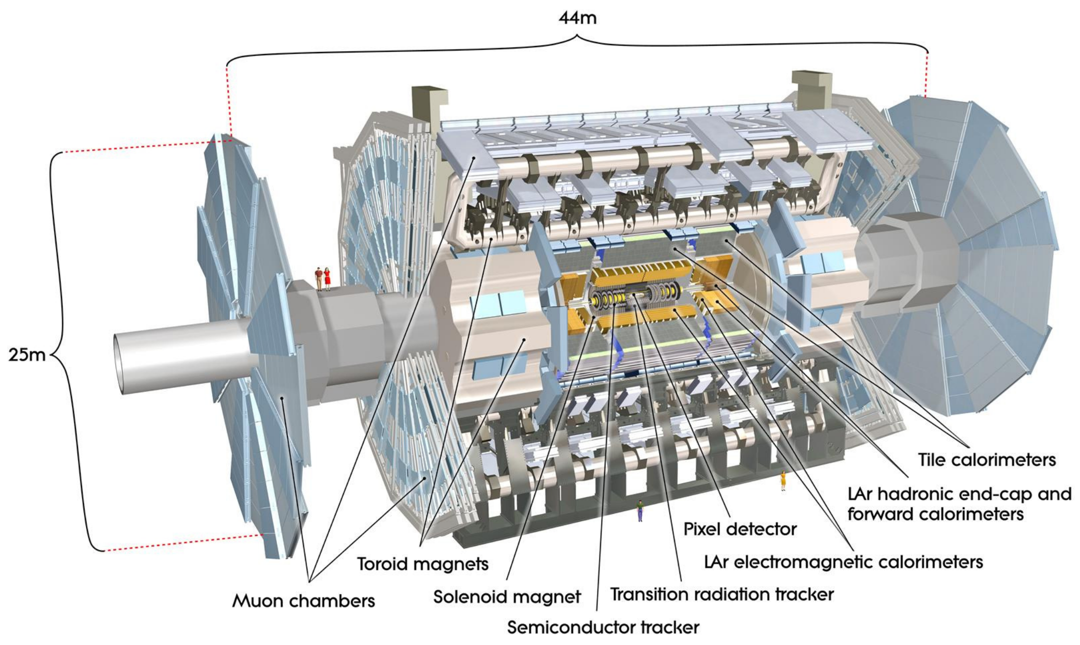

44m
25m
Tile calorimeters
LAr hadronic end-cap and forward calorimeters
Pixel detector
LAr electromagnetic calorimeters
Toroid magnets
Transition radiation tracker
Muon chambers
Solenoid magnet
Semiconductor tracker

Figure 2: An exploded image of the ATLAS detector [12]

The ATLAS detector is nominally forward-backward symmetric with respect to the interaction point. The magnet configuration comprises a thin superconducting solenoid surrounding the inner-detector cavity, and three large superconducting toroids (one barrel and two end-caps) arranged with an eight-fold azimuthal symmetry around the calorimeters. This fundamental choice has driven the design of the rest of the detector.
The inner detector is immersed in a 2 T solenoidal magnetic field. Pattern recognition, momentum and vertex measurements, and electron identification are achieved with a combination of discrete, high-resolution semiconductor pixel and strip detectors in the inner part of the tracking volume, and straw-tube tracking detectors with the capability to generate and detect transition radiation in its outer part.
High granularity liquid-argon (LAr) electromagnetic sampling calorimeters, with excellent performance in terms of energy and position resolution, cover a wide rapidity range. The hadronic calorimetry is provided by a scintillator-tile calorimeter, which is separated into a large barrel and two smaller extended barrel cylinders, one on either side of the central barrel. In the end-caps, LAr technology is also used for the hadronic calorimeters, matching the outer limits of end-cap electromagnetic calorimeters.
The LAr forward calorimeters provide both electromagnetic and hadronic energy measurements, and extend the pseudorapidity coverage. The calorimeter is surrounded by the muon spectrometer. The air-core toroid system, with a long barrel and two inserted

end-cap magnets, generates strong bending power in a large volume within a light and open structure. Multiple-scattering effects are thereby minimised, and excellent muon momentum resolution is achieved with three layers of high precision tracking chambers. [13]

2.2 Trigger System

The ATLAS detectors are designed to observe few billions $p-p$ collision per second [14], with a data volume of more than 60 TB/s. However only a small fraction of these events is interesting for the researchers and to reduce the flow of data to be saved on disk, ATLAS employs a two level Trigger system.
The first trigger level (L1) is composed of dedicated hardware operating a rapid and accurate selection of the events by analyzing the data from the calorimeters and the spectrometer's trigger chambers. The time to elaborate and distribute the trigger's decision is set to $2.5\mu s$ and the accepted event's rate can be at most 100kHz. During elaboration data are kept in pipelines made of highly integrated circuits (ASIC) positioned near the detector. Event data selected by the L1 are then collected trough a readout system and temporarily stored for the next level of trigger to access.
The High Level Triggers (or HLT) are instead composed of ordinary CPU farms accessing the data previously stored. HLTs have the job to reduce the event rate from 100kHz exiting the L1 to 0.5-1kHz,roughly corresponding to 1Gb/s: a manageable amount of data for successive analysis and are divided in two virtual levels themselves. L2 operating a first selection accessing only the data from the regions of interest (RoI) and the Event Filter (EF), which instead uses all data from the current event. Since the RoI correspond to a little fraction of the complete data ($\sim 5\%$), L2 allows a drastic reduction of the acquisition flow [15].
Events accepted by the L1 are distributed among the farm's porcessors, each of which executes L2 and then EF; if the event is again accepted by the HLT it's then transmitted to the archiving nodes (Data Logger) that make a temporary local copy and successively are transferred to the storage system at CERN's Tier 0 to be reconstructed.

2.3 Data types

The data arriving at Tier-0 from the ATLAS experiment is RAW data, a persistent representation of the event data produced at the ATLAS online cluster (involving HLT) in byte-stream format.
Reconstruction generates various data types the more relevant being Analysis Object Data (AOD), a C++ object containing a summary of the reconstructed event and sufficient information for common analyses, along side with Event Summary Data (ESD):

another C++ object that contains the detailed output of the detector reconstruction; it stores sufficient information to allow particle identification, track re-fitting and jet calibrations.
The majority of the Grid disk space is used by the AOD and DAOD formats in the orders of 60 and 80 PB, respectively [16].

Monte Carlo (MC) simulations are run in order to test the detector performance and reconstruction efficiencies for validating data-driven methods as well as to model processes of interest and their backgrounds.
All Monte Carlo production is done on the Grid and is divided in steps: the Event Generation is the simulation of the interaction of quarks and gluons from the $p-p$ collisions, their hadronization and decays in stable particles. The C++ objects resulting after this first passage are Event Data (EVNT).
Then a Detector Simulation is run to calculate how the emerging particles from the generator interact with the detector material, how they shower in secondary particles and how much energy they deposit on each of the detector's elements; the resulting C++ objects being called HITS.
The final stage is Digitization that creates Raw Object Data (ROD) representing the byte-stream format analogous to that of the RAW data.

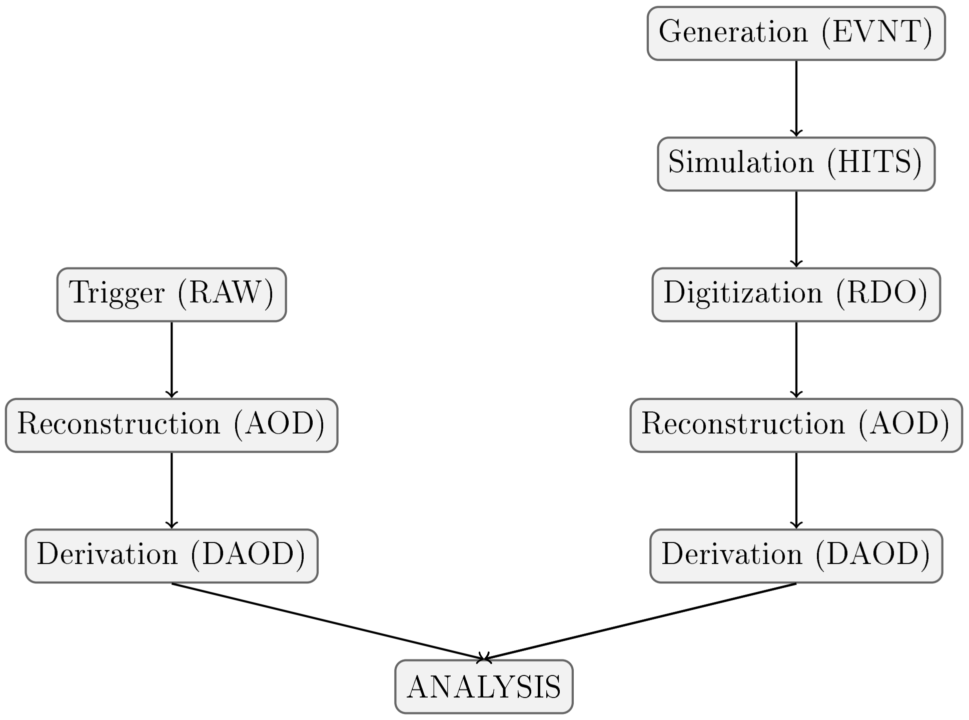


Figure 3: Workflow for the ATLAS Analysis model, with Data types

3 Data Flow and WLCG

The Worldwide LHC Computing Grid is a computer network that provides computing resources to store, distribute and analyse the data generated by the Large Hadron Collider (LHC) to research groups around the globe, all the machines connected to the WLCG are organized in a hierarchical Tier system, based on their function within the network.

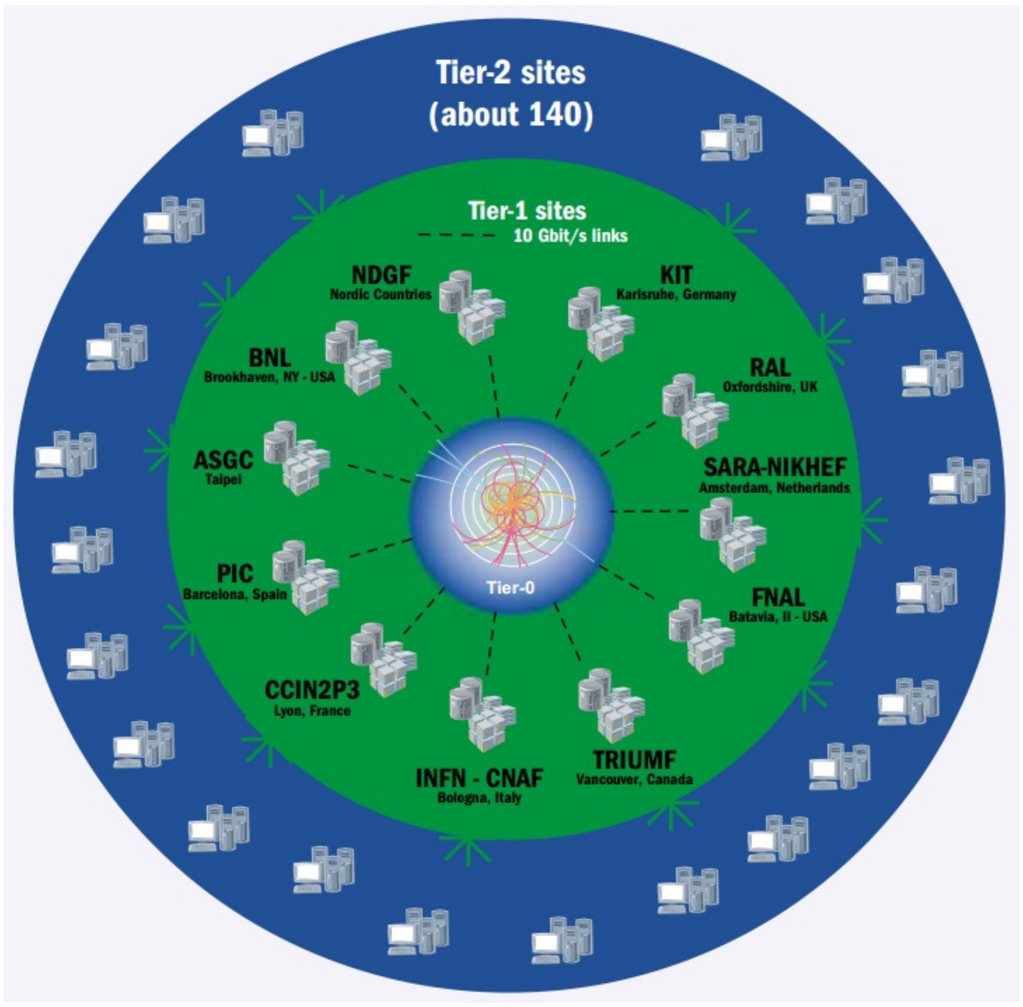


Figure 4: A visual representation of the WLCG tier system

The Tier-0 is situated at CERN and is responsible, among other things, for the first reconstruction of RAW data coming from all the Experiments held at CERN, operation which occupies a great part of the computing resources, as well as distributing a portion of the RAW data to the Tier-1s via the LCH Optical Private Network (LHCOPN) with

a stable 10 Gbit/s connection [17] up to 100 Gbit/s. It also manages the storage on tape of the RAW data and the storage of reconstructed Events to be shared among the Tier-1s in a second moment.
Tier-1s consist of 13 data centers with high computing performances and great storage capacity, both tape and disks, used to save a portion of the RAW data sent from CERN. Like the Tier-0, these sites are also responsible for reconstruction of RAW data and the distribution of elaborated data to subsequent Tiers to be further analyzed by local research groups. Tier-2s are smaller data centres that can store sufficient data and provide adequate computing power for specific analysis tasks. They handle a proportional share of the production and reconstruction of simulated events.
Tier-3s are access points to the grid, ranging from universities clusters to even normal PCs.

3.1 The Tier-1 at INFN-CNAF data centre

Since 2003 CNAF hosts the INFN Tier-1 data centre in Bologna, as well as a Tier-2 dedicated to the LHCb experiment, a cluster HPC (High Performance Computing) and a Tier-3 managed together with Bologna Division of INFN. More the three quarters of the total resources are dedicated to LHC's experiments which make a massive use of Grid technologies on which more refined and specific services have been developed for data management, monitoring and jobs [18].

3.2 The StoRM Storage System

StoRM [19] is a Storage Resource Manager (SRM) for disk based storage systems designed to support guaranteed space reservation and direct access (POSIX I/O calls) as well as high performance parallel file systems such as IBM's General Parallel File System (GPFS) [20].
StoRM has a multilayered architecture. The front-end component exposes a web service interface where the user requests land, manages users authentication and sends requests to the back-end. The Request Data Base (DB) that stores both SRM requests and status together with file and space information, to remark that losing the database content only affects the ongoing operations. The back-end is the main component: provides space tokens management, user redirection to the proper URL, and executes all SRM synchronous requests like the creation of directories.

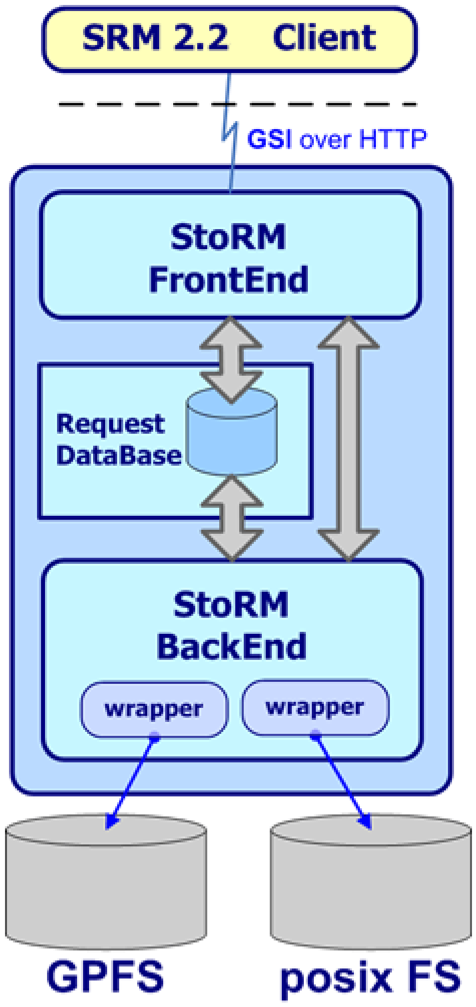


Figure 5: The front-end and back-end architecture of StoRM [19]

3.2.1 StoRM and GPFS

A cluster file system allows large numbers of disks attached to multiple storage servers to be configured as a single file system, providing transparent parallel access to storage devices while maintaining standard UNIX file system semantics, high speed file access to applications executing on multiple nodes of a cluster and high availability. StoRM takes advantage from aggregation functionalities provided by dedicated systems, such as parallel and cluster file systems. Such a file system allows to achieve complete redundancy without single point of failure increasing reliability and dynamic management of volumes (dynamic resize of file system, data migration between disks), all online, with a significant improvement of management flexibility.
Data access is performed through Network Shared Disks (NSD) for Local Area Network

(LAN) and GridFTP servers for Wide Area Network (WAN); it is possibile to have as many GridFTP servers as needed to provide the required transfer throughput. Moreover the GridFTP servers can be partitioned per space token (i.e. per logical subset of the storage), so that the real traffic load can always be turned on different machines.
The GPFS file system allows direct access from the clients using the file protocol avoiding the need of any external protocol, such as RFIO or Xrootd [20].
A fundamental feature offered by GPSF is the redundancy of the system: the unavailability of one (or even several) server only decreases the performance of the overall system. A disadvantage of GPFS is the cache amount limited by the operating system. For frequent access to a large amount of files this can slow very much the I/O operations. Directories suffering for this issue (e.g. shared software areas) can be exported using the Cluster Network File System (CNFS), an highly scalable and clustered version of NFS leverages on GPFS. [19]

4 Neural Networks

Neural Networks, also known as Artificial Neural Networks (ANNs) are a subset of machine learning and are at the heart of deep learning algorithms. Their name and structure are loosely inspired by the human brain, mimicking the connections between neurons and their activation.
Artificial Neural Networks (ANNs) are composed of nodes (also called neurons or artificial neurons) usually arranged in layers: an input layer, one or more hidden layers, and an output layer. The network consists of connections, each connection providing the output of one neuron as an input to the next node. Each connection is assigned a weight that represents its importance within the network. If the output of any individual node is above the specified threshold value, that node is activated, sending data to the next layer of the network. Otherwise, no data is passed along to the next layer of the network.

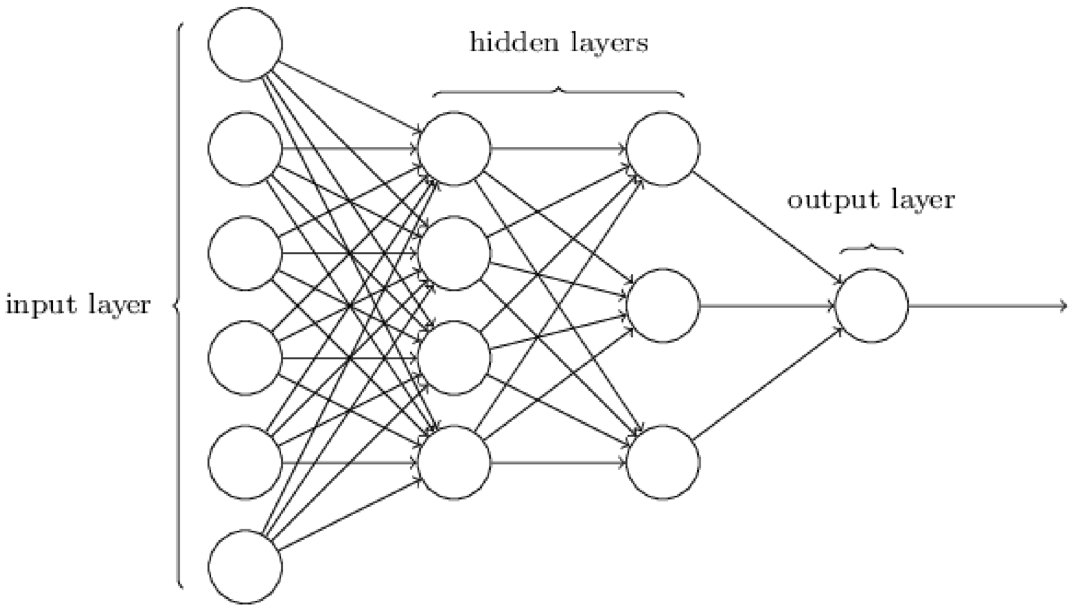


Figure 6: An example of feed forward neural network

To find the output of the neuron, first the sum of all the inputs, weighted by the weights of the connections from the inputs to the neuron, is taken then a bias term is added to this sum. That is $\Sigma_j w_j \cdot x_j + b$. This weighted sum is then passed through an activation function f_W to produce the output.
The simplest case of activation function is the one used in the perceptron, the step function:

$$f_W(x) = \begin{cases} 0 & \text{if } \mathbf{w} \cdot \mathbf{x} + b \leq 0 \\ 1 & \text{if } \mathbf{w} \cdot \mathbf{x} + b > 0 \end{cases}$$

A more refined version of which can be the sigmoid function:

$$f_W(x) = \sigma(x, w, b) = \frac{1}{1 + e^{-(\Sigma_j w_j \cdot x_j + b)}}$$

Among other frequently used activation functions such as hyperbolic tangent $tanh$ and ReLu, which is also called Rectifier: $max(0, \mathbf{w} \cdot \mathbf{x} + b)$.
Learning is the adaptation of the network to better handle a task by adjusting the weights of the network to improve the result's accuracy. This is done by minimizing the observed errors and is complete when examining additional observations does not usefully reduce the error rate. Practically this is done by defining a cost (or Loss) function that is evaluated periodically during learning; as long as its output continues to decline, learning continues.
Back propagation is a method used to adjust the connection weights to compensate for each error found during learning, it calculates the gradient of the cost function associated with a given state with respect to the weights. The weight updates can be done via stochastic gradient descent or other methods.
The three major learning paradigms are supervised learning, unsupervised learning and reinforcement learning. They each correspond to a particular learning task:
Supervised learning uses a set of paired inputs and desired outputs and the learning task is to produce the desired output for each input. In this case the cost function is related to eliminating incorrect deductions, a commonly used cost is the mean-squared error (MSE), which tries to minimize the average squared error between the network's output and the desired output. Tasks suited for supervised learning are pattern recognition (also known as classification) and regression (also known as function approximation), but is also applicable to sequential data.
For the Unsupervised Learning, no labels are given to the algorithm, leaving it on its own to find structure in its input. These algorithms discover hidden patterns or data groupings without the need for human intervention. Its ability to discover similarities and differences in information make it the ideal solution for exploratory data analysis, cross-selling strategies, customer segmentation, and image recognition.
Reinforcement learning is an area of machine learning concerned with how intelligent agents ought to take actions in an environment in order to maximize the notion of cumulative reward. A system interacts with a dynamic environment in which it must perform a certain goal (such as driving a vehicle or playing a game against an opponent). The system is provided feedback in terms of rewards and punishments as it navigates its problem space.

4.1 Recurrent Neural Networks and Long Short Term Memory

A Recurrent Neural Network (RNN) is a type of artificial neural network which uses sequential data or time series data. These deep learning algorithms are commonly used for

ordinal or temporal problems, such as language translation, natural language processing (NLP), speech recognition and image captioning.
The main difference from deep neural networks is that they take information from prior inputs to influence the current input and output. While traditional deep neural networks assume that inputs and outputs are independent of each other, the output of recurrent neural networks depend on the prior elements within the sequence.

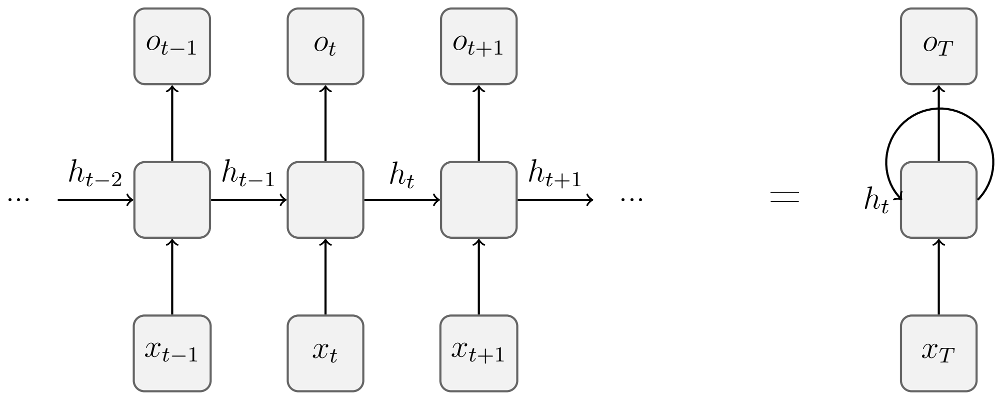


Figure 7: Unrolled RNN Layer

At each time step, the hidden state and output can be written as:

$$\begin{aligned} h^{(t)} &= f(W_i \cdot x^{(t)} + W_r \cdot h^{(t-1)} + b_h); \\ o^{(t)} &= g(W_o \cdot h^{(t)} + b_y). \end{aligned}$$

where W_i, W_r and W_y are weight matrices and b_h, b_y are biases. f and g are activation functions, exactly those discussed in the previous section.
The input and the hidden state can be concatenated to be multiplied by one weight variable W_h in the hidden layer, so that $h_t = f(x_t, h_{t-1}, W_h)$ and $o_t = g(h_t, W_o)$.
The discrepancy between output o_t and the desired label y_t is then evaluated by an objective function (loss) across all the T time steps as:

$$L(x_1, .., x_T, y_1, .., y_T, o_T, W_h, W_o) = \frac{1}{T} \sum_{t=1}^{T} l(o_t, y_t)$$

Also the Backpropagation gets calculated for each time step by computing the gradients of the loss function with respect to W_h. That is:

$$\begin{aligned}\frac{\partial L}{\partial W_h} &= \frac{1}{T}\sum_{t=1}^{T}\frac{\partial l(y_t, o_t)}{\partial W_h} \\ &= \frac{1}{T}\sum_{t=1}^{T}\frac{\partial l(y_t, o_t)}{\partial o_t}\frac{\partial g(h_t, W_h)}{\partial h_t}\frac{\partial h_t}{\partial W_h}\end{aligned}$$

Since an unrolled RNN can be viewed as a very deep feed farward network, it suffers from the same problems for the training stage, that is the tendency of gradients to vanish or diverge as they pass though so many time steps.
Since the new weights for the subsequent epoch is updated with a sort of gradient descent, like $W \leftarrow W - \alpha\frac{\partial L}{\partial W}$ as the gradient $\frac{\partial L}{\partial W} \rightarrow 0$, the new weights are almost identical to the previous ones, making it difficult to learn patterns over long distance in the sequence.
Long Short Term Memory (LSTM) was introduced by Sepp Hochreiter and Juergen Schmidhuber as a solution to vanishing gradient problem. In their paper [21], they work to address the problem of long-term dependencies. That is, if the previous state that is influencing the current prediction is not in the recent past, the RNN model may not be able to accurately predict the current state.
To remedy this, LSTMs have “cells” in the hidden layers of the neural network, which have three gates: an input gate, an output gate, and a forget gate. These gates control the flow of information which is needed to predict the output in the network.

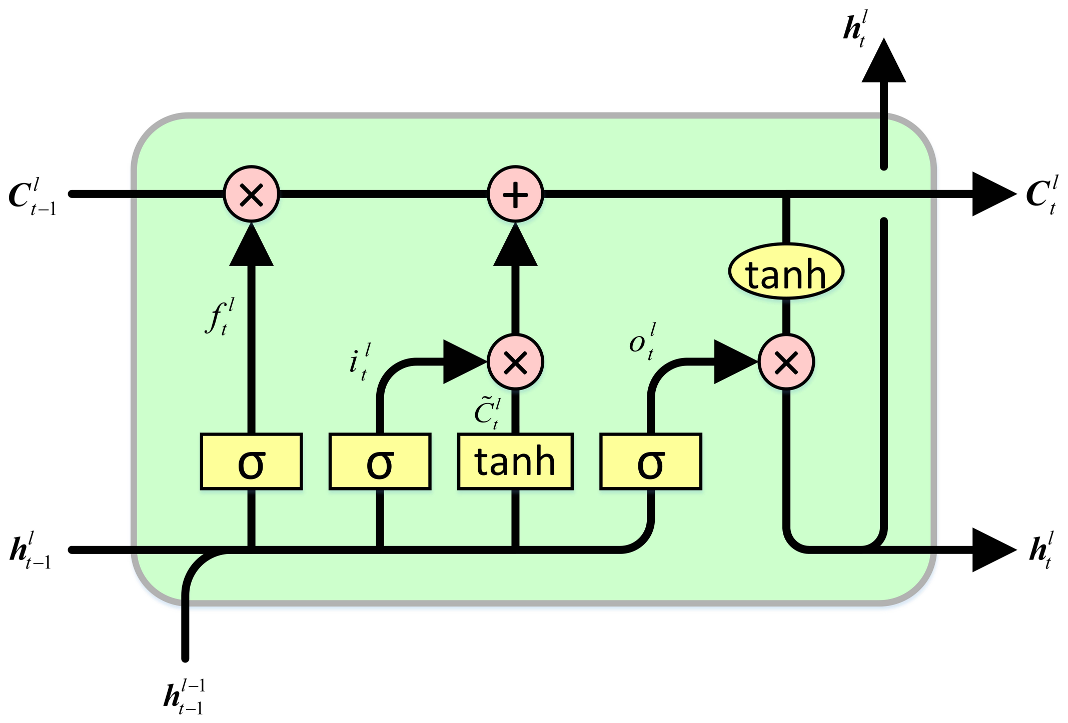


Figure 8: Long Short Term Memory architecture

The key to LSTMs is the cell state, the horizontal line running through the top of the diagram straight down the entire chain, conveying the information about sequence history. LSTMs have the ability to remove or add information to the cell state, carefully regulated by structures called gates.
The first step in the LSTM is to decide what information is going to be deleted from the cell state. This decision is made by a sigmoid layer called the "forget gate layer." It looks at h_{t-1} and x_t, and outputs a number between 0 and 1 for each number in the cell state C_{t-1}. A 1 represents "completely keep this" while a 0 represents "completely get rid of this."

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$$

The next step is to decide what new information is going to be stored in the cell state. This has two parts: first, a sigmoid layer called the "input gate layer" decides which values will be updated. Next, a tanh layer creates a vector of new candidate values, $\tilde{C}_t$, that could be added to the state. In the next step, we'll combine these two to create an update to the state.

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i).$$
$$\tilde{C_t} = tanh(W_C \cdot [h_{t-1}, x_t].b_C)$$

It's now time to update the old cell state, C_{t-1}, into the new cell state C_t by multiplying the old state by the forget state f_t. Then we add $i_t * \tilde{C_t}$.

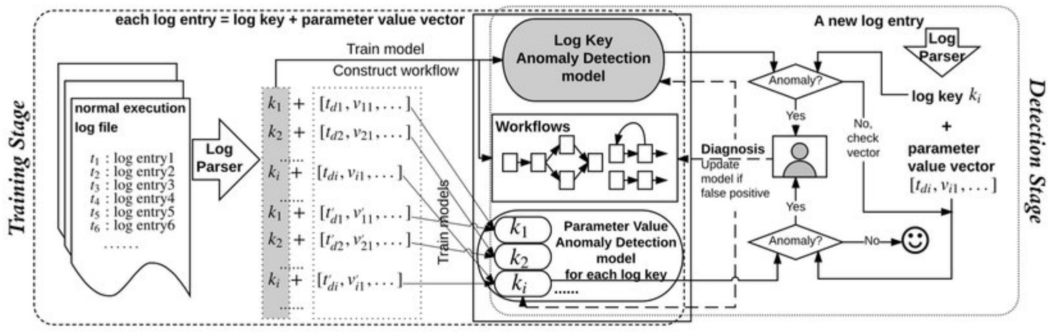
$$C_t = f_t * c_{t-1} + i_t * \tilde{C_t}$$

This is the new candidate values, scaled by how much we decided to update each state value.
Finally the output and hidden state are calculated based on the current cell state using a sigmoid and a tanh layer:

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o)$$
$$h_t = o_t * tanh(C_t).$$

4.2 DeepLog

DeepLog [22], developed by Min Du, Feifei Li, Guineng Zheng, Vivek Srikumar at the University of Utah, is a data-driven approach for anomaly detection that leverages the large volumes of system logs.
The key idea of DeepLog comes from Natural Language Processing (NLP) by interpreting each log line as belonging to a sequence that follows certain patterns and rules; indeed a system log is produced by a program with defined logic and control flow, much like a natural language although more structured and restricted in dictionary.
DeepLog is a deep neural network that model this sequence of log entries using a Long Short-Term Memory (LSTM), allowing it to automatically learn a model of patterns from normal system execution and to flag deviations from these patterns as anomalies. Furthermore, since is a learning-driven approach, it's possible to incrementally update DeepLog's model in order to adapt to new normal patterns.
The complete architecture of DeepLog is composed of three parts: LogKey anomaly detection model, Parameter value anomaly detection Model and Workflow Construction the latter being useless for this particular case as the sequences can be directly extracted from the log files by means of grouping log entries with the same token (which is assigned by the back end and identifies the process). Also the Parameter value anomaly detection wasn't used in this work, leaving only the log Key anomaly detection to study.



each log entry = log key + parameter value vector
Training Stage
normal execution log file
Log Parser
Train model
Construct workflow
Log Key Anomaly Detection model
Workflows
Train models
Parameter Value Anomaly Detection model for each log key
Diagnosis
Update model if false positive
Anomaly?
Yes
No, check vector
No
A new log entry
Log Parser
log key k_i
+
parameter value vector
Detection Stage

Figure 9: The complete architecture of DeepLog as presented in [22]

The training data are a small portion of the complete log entries from normal system execution paths usually generated in a controlled environment such as a virtual machine. Each log message (only the content) is parsed to a log key represented by an integer number and a sequence is constructed from the token identifier. The input of the model is a part of these sequences extracted by means of a window of fixed size sliding on it and outputs a probability distribution for the next log key in the sequence as explained below.

4.3 Log Key Model

Since the total number of distinct print statements (that print log entries) in a source code is constant, so is the total number of distinct log keys. Let $K = \{k_1, k_2, ..., k_n\}$ be the set of distinct log keys from a log-producing system source code. Once log entries are parsed into log keys, the log key sequence reflects an execution path that leads to that particular execution order of the log print statements. Let m_i denote the value of the key at position i in a log key sequence. Clearly, m_i may take one of the n possible keys from K, and is strongly dependent on the most recent keys that appeared prior to m_i.
Anomaly detection in a log key sequence can be modeled as a multi-class classification problem, where each distinct log key defines a class. The input is a history of recent log keys, and the output is a probability distribution over the n log keys from K, representing the probability that the next log key in the sequence is a key $k_i \in K$. Suppose t is the sequence id of the next log key to appear. The input for classification is a window w of the h most recent log keys. That is, $w = \{m_{t-h}, ..., m_{t-2}, m_{t-1}\}$, where each m_i is in K and is the log key from the log entry e_i. Note that the same log key value may appear several times in w. The output of the training phase is a model of the conditional probability distribution $Pr[m_t = k_i|w]$ for each $k_i \in K(i = 1, ..., n)$. The detection phase uses this model to make a prediction and compare the predicted output against the observed

log key value that actually appears.

The training stage relies on a small fraction of log entries produced by normal execution of the underlying system. For each log sequence of length h in the training data, DeepLog updates its model for the probability distribution of having $k_i \in K$ as the next log key value. For example, suppose a small log file resulted from normal execution is parsed into a sequence of log keys: $\{1, 9, 3, 10, 3, 4\}$. Given a window size $h = 3$, the input sequence and the output label pairs to train DeepLog will be: $\{1, 9, 3 \to 10\}$, $\{9, 3, 10 \to 3\}$, $\{3, 10, 3 \to 4\}$.
To test if an incoming log key m_t is to be considered normal or abnormal, we send $w = m_{t-h}, ..., m_{t-1}$ to DeepLog as its input. The output is a probability distribution $Pr[m_t|w] = \{k_1 : p_1, k_2 : p_2, ..., k_n : p_n\}$ describing the probability for each log key from K to appear as the next log key value given the history.
Since the model is a multi-class classifier Cross Entropy loss function is used:

$$Loss = -\ln\left(\frac{e^{x[class]}}{\sum_{j=0}^{N} e^{x[j]}}\right) = -x[class] + \ln\left(\sum_{j=0}^{N} e^{x[j]}\right)$$

The function expects a "class" index, $class \in [0, C-1]$, with C number of classes, as the target for each value of a 1D tensor of size N (the batch size). $x[class]$ is the output of the model for the label class. Cross Entropy increases as the predicted value diverges from the label and decreases as the prediction approaches the ground truth.
In practice, multiple log key values may appear as m_t. For instance, if the system is attempting to connect to a host, then m_t could either be 'Waiting for * to respond' or 'Connected to *'; both are normal system behavior. DeepLog must be able to learn such patterns during training. Our strategy is to sort the possible log keys K based on their probabilities $Pr[m_t|w]$, and treat a key value as normal if it's among the top g candidates. A log key is flagged as being from an abnormal execution otherwise.

5 Log Analysis

A wide range of programmable technologies, from network devices to applications and operating systems, produce records about the users' and their own activity called logs. These messages are generated in a time ordered sequence from possibly concurrent processes and provide a great amount of information on the user's behavior and system performance encapsulated in a text string. Logs are implemented by developers to make debugging and maintenance easier by announcing security-relevant or operations-relevant events like a user login or systems errors and can be either saved on disk or directed as a network stream to a log collector.
The analysis of such records is a useful tool to make diagnosis about the system performance and failures, as well as monitoring users activity or recognizing security threats in a timely manner.

5.1 Dataset preparation

The Complete dataset consists of the log files generated by one week of activity of StoRM and is composed by 5 types of messages: front-end, back-end, back-end-metrics, monitoring, heartbeat. For the following work were used the system logs from two different instances of StoRM front-end server we used.

Date	Instance 1	Instance 2
07/03/2020	3410401	3382210
08/03/2020	1559369	1525579
09/03/2020	1777349	1793120
10/03/2020	1732644	1808492
11/03/2020	594199	612507
12/03/2020	1147129	1156481
13/03/2020	1261513	1323814

Table 1: Numerosity of log lines for each instance of StoRM front-end

The training dataset was created from the first instance of StoRM front-end and kept separated from the test data, coming from the other front-end instance, in order to avoid overfitting. The test dataset was divided in normal and abnormal depending on the appearance of the words "ERROR" and "FAILURE" at any point in the sequences.
Even thought this division keeps track of the day in which the log itself got generated, this information is discarded as the actual train and test files are obtained by randomly extracting sequences coming from their respective dataset, since the objective is to value different models upon their efficiency on finding anomalies.

5.1.1 Parsing

Parsing unstructured log entries is known to be effective to get faster and more precise results on several algorithms [23] and is almost always employed in data mining and in particular in log mining; therefore only a part of each line is meaningful for the Anomaly Detection stage and clustering purposes, namely "Component" and "Content". Each line has been parsed, filling a comma separated values (csv) file created accordingly to the format of the log file. A log format in this work is a string containing the fields for the log entry to be divided in isolated with angular parenthesis and any character that appear as delimiter in the lines. For the front-end server the log format utilized is:

```
log_format = '<Date> <Time> <Pid> - <Level> <Component>: <Content>'
```

From this string regular expressions are generated to split the log lines and extract the headers for the csv file. For example, from the following raw log entries:

```
03/07 03:12:03.268 Thread 7  - INFO [17ea2e4e-e5c2-42d5-8fdc-6c76ce64dbd3]:
 Result for request 'Put done' is 'SRM\_SUCCESS'
03/07 03:12:03.318 Thread 42 - INFO [ac57e4dd-46a6-4219-99ec-38c9c7c0d809]:
 process_request : Connection from 2001:6b0:17:180::2:2
03/07 03:12:03.382 Thread 55 - INFO [9a632c05-9787-4cb8-9ab4-4e836aec64f1]:
 process_request : Connection from 2001:1458:d00:a::100:304
03/07 03:12:03.419 Thread 7  - INFO [17ea2e4e-e5c2-42d5-8fdc-6c76ce64dbd3]:
 ns1__srmLs : Request: Ls. IP: 2001:1458:201:e4::100:574. Client DN: ...
03/07 03:12:03.428 Thread 47 - INFO [f2bf8eb2-b0cf-4896-8676-7358fbff7a64]:
 Result for request 'Rm' is 'SRM\_FAILURE'
```

a structured file is derived, with the log template fields as headers and the corresponding contents for each log line:

Date	Time	PID	Level	Component (TKN)	Content
03/07	03:12:03.268	Thread 7	INFO	[17ea2e4e-e5c2-42d5-8fdc-6c76ce64dbd3]	Result for request ...
03/07	03:12:03.318	Thread 42	INFO	[ac57e4dd-46a6-4219-99ec-38c9c7c0d809]	process_request : ...
03/07	03:12:03.382	Thread 55	INFO	[9a632c05-9787-4cb8-9ab4-4e836aec64f1]	process_request : ...
03/07	03:12:03.419	Thread 7	INFO	[17ea2e4e-e5c2-42d5-8fdc-6c76ce64dbd3]	ns1__srmLs : Request: ...
03/07	03:12:03.428	Thread 47	INFO	[f2bf8eb2-b0cf-4896-8676-7358fbff7a64]	Result for request ...

The column labeled "Content" is then used for subsequent processing.

5.1.2 Masking

In order to improve on clustering efficiency, parameters such as IP, Token or ClientDN are masked using regular expressions which are compiled at masking. Masking instructions, composed by the regular expression and its corresponding mask, are are collected in an array in the configuration file of Drain, which will be exposed in the following chapter. For example, the masking instruction for $' < TKN >'$is:

```
(\\w{8}-\\w{4}-\\w{4}-\\w{4}-\\w{12})
```

Since the parameters that are to be masked are usually wrapped by special characters, the employed regular expressions also use positive lookaheads `(?<=[^A-Za-z0-9])|^)` and lookbehinds `((?=[^A-Za-z0-9])|$)` for non-alphanumerical characters to be excluded from the match when at the beginning or at the end of the string. Lookahead and lookbehind do not consume characters in the string, but only assert whether a match is possible or not. Lookaround allows to create regular expressions that are impossible to create without them, or that would get very longwinded without them.
After masking is done the log lines look like this:

```
 Request <TSK> from Client IP='<IP>' Client DN=<ID># Requested token '<TKN>'
on <NUM> SURL(s): '<URL>'
```

5.1.3 Drain

Drain, the rather imaginative acronym for "a fixed **d**epth t**r**ee b**a**sed onl**i**ne log parsi**n**g method", is a parsing algorithm for unstructured log messages developed by the Hong Kong and Sun Yat-sen Universities [24]. When a new raw message arrives, Drain preprocesses it by domain knowledge as explained in the Parsing and Masking sections, then a log group (i.e. leaf node of the tree) by following the rules of the internal nodes of the tree. If a suitable group is found the log message will be matched with the corresponding event template stored in that node, otherwise a new group is created.

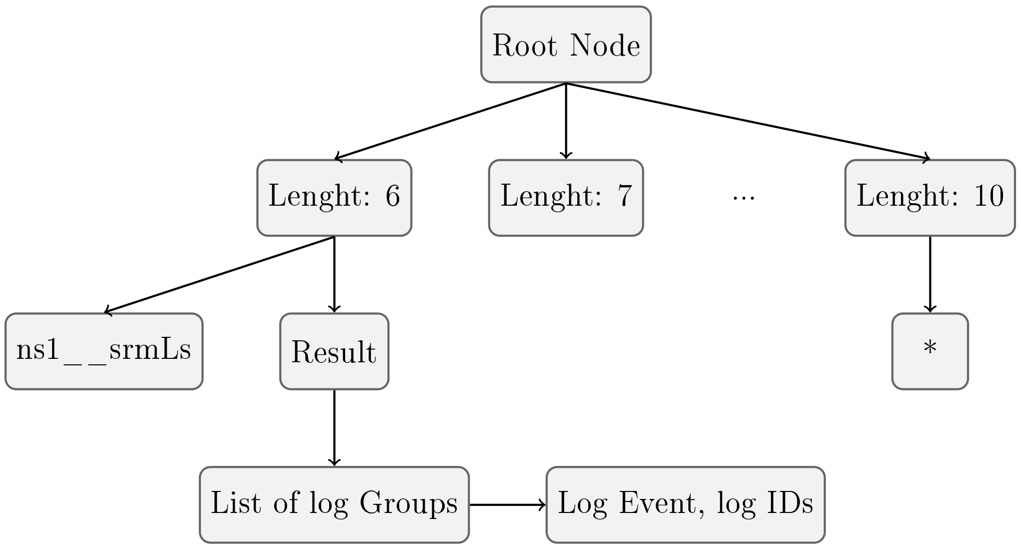


Figure 10: Structure of Parse Tree in Drain (depth=3)

Drain starts from the root node of the parse tree with the preprocessed log message. The 1-st layer nodes in the parse tree represent log groups whose log messages are of different log message lengths. In this step, Drain selects a path to a 1-st layer node based on the log message length of the preprocessed log message. For example, for log message

```
Result for request 'Ls' is 'SRM_SUCCESS'
```

Drain traverse to the internal node “Length: 6” in Figure 10. This is based on the assumption that log messages with the same log event will probably have the same log message length. Although it is possible that log messages with the same log event have different log message lengths, it can be handled by simple postprocessing.
Then Drain traverses from a 1-st layer node, which is searched in the previous step, to a leaf node. Specifically, Drain selects the next internal node by the tokens in the beginning positions of the log message based on the assumption that the first token of a log message is more likely to be constants. For example, for log message: "Result for request 'Ls' is 'SRM_SUCCESS'".
Drain traverses from 1-st layer node “Length: 6” to 2-nd layer node “Result” because the token in the first position of the log message is “Result”. Then Drain will traverse to the leaf node linked with internal node “Result”, and go to step 4.
The number of internal nodes that Drain traverses in this step is $(depth - 2)$, where depth is the parse tree parameter restricting the depth of all leaf nodes. Thus, there are $(depth - 2)$ layers that encode the first $(depth - 2)$ tokens in the log messages as search rules. In the example above, we use the parse tree in Figure 2 for simplicity, whose depth is 3, so we search by only the token in the first position. In practice, Drain can consider more preceding tokens with larger depth settings. Note that if depth is 2, Drain only

considers the first layer used by step 2.
In some cases, a log message may start with a parameter, these kinds of log messages can lead to branch explosion in the parse tree because each parameter will be encoded in an internal node. To avoid branch explosion, only the tokens with no digits are considered for this step. If a token contains digits, it will match a special internal node “*”. For example, for the log message:

```
__process_file_request<> : Received - 4 - protocols,...
```

Drain will traverse to the internal node “*” instead of “4”. Besides, a parameter maxChild is also defined, which restricts the maximum number of children of a node. If a node already has maxChild children, any non-matched tokens will match the special internal node “*” among all its children.
Before this step, Drain has traversed to a leaf node, which contains a list of log groups. The log messages in these log groups comply with the rules encoded in the internal nodes along the path. For example, the log group in Figure 6 has log event "Result for request ’Ls’ is ’SRM_SUCCESS’" where the log messages contain 6 tokens and start with token “Result”. In this step, Drain selects the most suitable log group from the log group list. We calculate the similarity *simSeq* between the log message and the log event of each log group. *simSeq* is defined as following:

$$simSeq = \frac{\Sigma_{i=1}^{n} \delta(seq_1(i), seq_2(i))}{n}$$

where $seq1$ and $seq2$ represent the log message and the log event respectively; $seq(i)$ is the i-th token of the sequence; n is the log message length of the sequences; function δ is defined as following:

$$\delta(t_1, t_2) = \begin{cases} 1 & \text{if } t_1 = t_2, \\ 0 & \text{otherwise.} \end{cases}$$

where $t1$ and $t2$ are two tokens. After finding the log group with the largest *simSeq*, we compare it with a predefined similarity threshold *st*. If $simSeq \geq st$, Drain returns the group as the most suitable log group. Otherwise, Drain returns a flag (None in Python) to indicate no suitable log group.
If a suitable log group is returned in step 4, Drain will add the log ID of the current log message to the log IDs in the returned log group. Besides, the log event in the returned log group will be updated. Specifically, Drain scans the tokens in the same position of the log message and the log event. If the two tokens are the same, we do not modify the token in that token position. Otherwise, we update the token in that token position by wildcard (i.e., *) in the log event.
If Drain cannot find a suitable log group, it creates a new log group based on the current log message, where log IDs contains only the ID of the log message and log event is exactly the log message. Then, Drain will update the parse tree with the new log group.

Intuitively, Drain traverses from the root node to a leaf node that should contain the new log group, and adds the missing internal nodes and leaf node accordingly along the path.
For the extraction of log templates was employed Drain3, an implementation of Drain by IBM written in Python3 and compatible with later versions [25]. It features a persistence handler that makes possible to manage the save sates of the parsing tree by means of an Apache Kafka topic, Redis or a file as well as the opportunity to run in a streaming fashion by feeding log lines one by one, or as they are produced.
Drain's parameters and masking instructions are configured using configparser which implements a basic configuration language for end users to customize the program by editing a configuration file, in this case drain3.ini in the working directory.
Drain3 can be installed with pip and was used to implement a parser able to structure the unstructured log entry using pandas, a fast and easy to use open source data analysis and manipulation tool [26], and regular expressions to separate different parts of the complete log entry. Once the entries are structured the "Content" part of each log line is passed as argument to template_miner.add_log_message() to be parsed.

5.1.4 Post Processing

Since it is possible that log messages belonging to the same log event have different lengths and would not be included in the same cluster by drain, a post processing work may be done to obtain more readable log templates, effectively reducing the number of cluster for further analysis.
The post process work mostly consisted of manually merging clusters whose log lines were truncated at different lengths. For example:

```
Request <TSK> from Client IP='<IP>' Client DN=<ID>#
Requested <NUM> SURL(s): '<URL> srm TRUNCATED
Request <TSK> from Client IP='<IP>' Client DN=<ID>#
Requested <NUM> SURL(s): '<URL> s TRUNCATED
Request <TSK> from Client IP='<IP>' Client DN=<ID>#
Requested <NUM> SURL(s): '<URL> srm:/ TRUNCATED
```

Obviously belong to the same cluster, nominally:

```
Request <TSK> from Client IP='<IP>' Client DN=<ID>#
Requested <NUM> SURL(s): '<URL>'
```

These ill truncated messages have little occurrences in the present dataset and should not impact the Log Analysis in a great way.

5.2 Log Analysis Results

Here follows the results of the analysis on the log dataset; for the clustering part the parameter sim_th (similarity threshold) was changed from the default value of 0.4 to 1 in order to avoid the use of the wildcard "*" by Drain, obtaining more distinct log messages after pre-processing with regular expressions. The post-processing was conducted on the classes obtained with $sim\_th = 1$. The clustering process was also reproduced with $sim\_th = 0.875$ to obtain a third different number of classes to confront with subsequent analysis.
After the clustering is done, the raw log file has been converted in sequences of integers, with each number (log key) corresponding to a log message type. The three datasets obtained so far have then been separated in two distinct sets each for the training and prediction stage respectively and the prediction ones got further divided in normal and abnormal obtaining the test sets.

5.2.1 Clustering Results

The following are the cluster extracted using $sim\_th = 1$ (from left: the cluster's identifier, cluster size and log template associated) followed by the 44 post processed from the 81 templates first extracted.
Using these templates mined, three data sets were created encoding the same system behaviors but with different representation as some cluster changes his log key when changing enumeration.

```
A0001 (size 1383831): process_request : Connection from <IP>
A0002 (size 196048): ns1__srmPutDone : Request: Put done. IP: <IP>. Client DN: <ID>(s): <URL> token: <TKN>
A0003 (size 2997244): Result for request <TSK> is 'SRM_SUCCESS'
A0004 (size 1593720): ns1__srmLs : Request: Ls. IP: <IP>. Client DN: <ID>
A0005 (size 45165): Request <TSK> from Client IP='<IP>' Client DN=<ID># Requested token '<TKN>' on <NUM> SURL(s): '<URL> TRUNCATED
A0006 (size 468714): Result for request <TSK> is 'SRM_REQUEST_INPROGRESS'
A0007 (size 449697): ns1__srmPing : Request: Ping. IP: <IP>. Client DN: <ID>
A0008 (size 451590): Request <TSK> from Client IP='<IP>' Client DN=<ID># Requested <NUM> SURL(s): '<URL>
A0009 (size 452003): Result for request <TSK> is 'SRM_REQUEST_QUEUED'. # Produced request token: '<TKN>'
A0010 (size 1406293): Request <TSK> from Client IP='<IP>' Client DN=<ID># Requested token '<TKN>'
A0011 (size 107104): ns1__srmRm : Request: Rm. IP: <IP>. Client DN: <ID>(s): <URL>
A0012 (size 200056): ns1__srmGetSpaceTokens : Request: Get space tokens. IP: <IP>. Client DN: <ID>
A0013 (size 8809): process_request : Connection from ::1
A0014 (size 557459): Result for request <TSK> is 'SRM_REQUEST_QUEUED'
A0015 (size 253023): ns1__srmReleaseFiles : Request: Release files. IP: <IP>. Client DN: <ID>(s): <URL> token: <TKN>
A0016 (size 2327): __process_file_request<> : Protocol check failed, received some unsupported protocols
A0017 (size 1646): __process_file_request<> : Received - 4 - protocols, 2 are supported, 2 are not supported
A0018 (size 2327): __process_file_request<> : Some of the provided protocols are supported, proceeding
A0019 (size 26309): Request <TSK> from Client IP='<IP>' Client DN=<ID># Requested token '<TKN>' on <NUM> SURL(s): '<URL>
A0020 (size 327364): Result for request <TSK> is 'SRM_FAILURE'
A0021 (size 60564): ns1__srmMv : Request: Mv. IP: <IP>. Client DN: <ID>_surl: <URL> to_surl: <URL>
A0022 (size 11676): ns1__srmMkdir : Request: Mkdir. IP: <IP>. Client DN: <ID>(s): <URL>
A0023 (size 1362): ns1__srmAbortRequest : Request: Abort request. IP: <IP>. Client DN: <ID>
A0024 (size 903): Result for request <TSK> is 'SRM_INTERNAL_ERROR'
A0025 (size 680): __process_file_request<> : Received - 5 - protocols, 3 are supported, 2 are not supported
A0026 (size 395): Result for request <TSK> is 'SRM_INVALID_PATH'
A0027 (size 1007): ns1__srmGetSpaceMetaData : Request: Get space metadata. IP: <IP>. Client DN: <ID>
A0028 (size 575): Result for request <TSK> is 'SRM_INVALID_REQUEST'
A0029 (size 406): ns1__srmReleaseFiles : Request: Release files. IP: <IP>. Client DN: <ID>
A0030 (size 16): Result for request <TSK> is 'SRM_DUPLICATION_ERROR'
A0031 (size 194): ns1__srmAbortFiles : Request: Abort files. IP: <IP>. Client DN: <ID>(s): <URL> token: <TKN>
A0032 (size 232): rpcResponseHandler_AbortFiles : arrayOfFileStatuses not specified by BE.
A0033 (size 375): Request <TSK> from Client IP='<IP>' Client DN=<ID># Requested <NUM> SURL(s): '<URL> TRUNCATED
A0034 (size 21): ns1__srmCheckPermission : Request: Check permission. IP: <IP>. Client DN: <ID>(s): <URL>
A0035 (size 21): Result for request <TSK> is 'SRM_NOT_SUPPORTED'
A0036 (size 1): ns1__srmPing : Request: Ping. IP: <IP>. Client DN: <ID>'Keefe
A0037 (size 1): ns1__srmLs : Request: Ls. IP: <IP>. Client DN: <ID>'Keefe
A0038 (size 1): Request <TSK> from Client IP='<IP>' Client DN='<ID><TSK># Requested <NUM> SURL(s): '<URL>
A0039 (size 3): Request <TSK> from Client IP='<IP>' Client DN='<ID><TSK># Requested token '<TKN>'
A0040 (size 1): ns1__srmReleaseFiles : Request: Release files. IP: <IP>. Client DN: <ID>'Keefe. surl(s): <URL> token: <TKN>
A0041 (size 1): rpcResponseHandler_ReleaseFiles : xml_arrayOfFileStatuses is empty
A0042 (size 1): __process_file_request<> : Received - 5 - protocols, 2 are supported, 3 are not supported
A0043 (size 1): ns1__srmRmdir : Request: Rmdir. IP: <IP>. Client DN: <ID>(s): <URL>
A0044 (size 1): Result for request <TSK> is 'SRM_NON_EMPTY_DIRECTORY'
A0045 (size 4): Request <TSK> from Client IP='<IP>' Client DN=<ID># Requested <NUM> SURL(s): '<URL> srm TRUNCATED
A0046 (size 24): Request <TSK> from Client IP='<IP>' Client DN=<ID># Requested <NUM> SURL(s): '<URL> s TRUNCATED
A0047 (size 1): Request <TSK> from Client IP='<IP>' Client DN=<ID># Requested <NUM> SURL(s): '<URL> srm:/ TRUNCATED
A0048 (size 8): ns1__srmAbortFiles : Request: Abort files. IP: <IP>. Client DN: <ID>(s): <URL> t TRUNCATED
A0049 (size 2): Request <TSK> from Client IP='<IP>' Client DN=<ID># Requested <NUM> SURL(s): '<URL> srm:// TRUNCATED
A0050 (size 3): ns1__srmAbortFiles : Request: Abort files. IP: <IP>. Client DN: <ID>(s): <URL> to TRUNCATED
A0051 (size 11): ns1__srmAbortFiles : Request: Abort files. IP: <IP>. Client DN: <ID>(s): <URL> TRUNCATED
A0052 (size 3): ns1__srmAbortFiles : Request: Abort files. IP: <IP>. Client DN: <ID>(s): <URL> s TRUNCATED
A0053 (size 2): ns1__srmAbortFiles : Request: Abort files. IP: <IP>. Client DN: <ID>(s): <URL> srm TRUNCATED
A0054 (size 9): ns1__srmAbortFiles : Request: Abort files. IP: <IP>. Client DN: <ID>(s): <URL> sr TRUNCATED
A0055 (size 252): rpcResponseHandler_Ls : ERROR: XML-RPC Fault: RPC failed at server. Failed to invoke method ls in class ... Invalid argument (code: 0)
A0056 (size 204): rpcResponseHandler_Ls : ERROR: XML-RPC Fault: RPC failed at server. Failed to invoke method ls in class ... Invalid argument (code: 0)
A0057 (size 2): rpcResponseHandler_Rm : ERROR: XML-RPC Fault: libcurl failed to execute the HTTP POST transaction, explaining:
                Failed connect to storm-atlas.cr.cnaf.infn.it:8080; Operation now in progress (code: -504)
A0058 (size 1): ns1__srmAbortFiles : Request: Abort files. IP: <IP>. Client DN: <ID>(s): <URL> token: TRUNCATED
A0059 (size 8): storm::BolStatusRequest::loadFromDB() : No tokens found for token <TKN> and the requested SURLs
A0060 (size 1): Request <TSK> from Client IP='<IP>' Client DN=<ID># Requested token '<TKN>' on <NUM> SURL(s): '<URL> srm:/ TRUNCATED
A0061 (size 1): Request <TSK> from Client IP='<IP>' Client DN=<ID># Requested token '<TKN>' on <NUM> SURL(s): '<URL> s TRUNCATED
A0062 (size 2): Request <TSK> from Client IP='<IP>' Client DN=<ID># Requested token '<TKN>' on <NUM> SURL(s): '<URL> srm TRUNCATED
A0063 (size 1): Request <TSK> from Client IP='<IP>' Client DN=<ID># Requested token '<TKN>' on <NUM> SURL(s): '<URL> srm: TRUNCATED
A0064 (size 7): Request <TSK> from Client IP='<IP>' Client DN=<ID># Requested <NUM> SURL(s): '<URL> sr TRUNCATED
A0065 (size 1): Request <TSK> from Client IP='<IP>' Client DN=<ID># Requested <NUM> SURL(s): '<URL> srm: TRUNCATED
A0066 (size 1): rpcResponseHandler_PutDone : xml_arrayOfFileStatuses is empty
A0067 (size 1): ns1__srmAbortFiles : Request: Abort files. IP: <IP>. Client DN: <ID>(s): <URL> token TRUNCATED
A0068 (size 1): ns1__srmAbortFiles : Request: Abort files. IP: <IP>. Client DN: <ID>(s): <URL> srm:/ TRUNCATED
A0069 (size 2): rpcResponseHandler_Ls : ERROR: XML-RPC Fault: RPC failed at server. Failed to invoke method ls in class ... Communications link failure
A0070 (size 2): rpcResponseHandler_Ls : ERROR: XML-RPC Fault: libcurl failed to execute the HTTP POST transaction, explaining:
                            couldn't connect to host (code: -504)
A0071 (size 1): logConfiguration : Starting StoRM frontend as user: storm
A0072 (size 1): logConfiguration : --------------------- Configuration -----------------
A0073 (size 18): logConfiguration : <ID>
A0074 (size 1): logConfiguration : fe.gsoap.send_<ID>
A0075 (size 1): logConfiguration : fe.gsoap.recv_<ID>
A0076 (size 1): logConfiguration : argus-pepd-endpoint=
A0077 (size 1): logConfiguration : xmlrpc <ID>
A0078 (size 1): logConfiguration : -------------------------------------------------------
A0079 (size 1): initSoap : Mapping disabled
A0080 (size 1): main : StoRM frontend successfully started...
A0081 (size 1): Request <TSK> from Client IP='<IP>' Client DN=<ID># Requested token '<TKN>' on <NUM> SURL(s): '<URL> sr TRUNCATED
```

Then the results for $sim\_th = 0.875$:

```
A0001 (size 1207895): process_request : Connection from <IP>
A0002 (size 162647): ns1__srmPutDone : Request: Put done. IP: <IP>. Client DN: <ID>(s): <URL> token: <TKN>
A0003 (size 2572483): Result for request <TSK> is 'SRM_SUCCESS'
A0004 (size 1365436): ns1__srmLs : Request: Ls. IP: <IP>. Client DN: <ID>
A0005 (size 41196): Request <TSK> from Client <ID>=<ID> # Requested token '<TKN>' on <NUM> SURL(s): '<URL> <URL> <URL> <*> TRUNCATED
A0006 (size 452114): Result for request <TSK> is 'SRM_REQUEST_INPROGRESS'
A0007 (size 387908): ns1__srmPing : Request: Ping. IP: <IP>. Client DN: <ID>
A0008 (size 388659): Request <TSK> from Client <ID>=<ID> # Requested <NUM> SURL(s): '<URL>
A0009 (size 389256): Result for request <TSK> is 'SRM_REQUEST_QUEUED'. # Produced request token: '<TKN>'
A0010 (size 1255140): Request <TSK> from Client <ID>=<ID> # Requested token '<TKN>'
A0011 (size 90330): ns1__srmRm : Request: Rm. IP: <IP>. Client DN: <ID>(s): <URL>
A0012 (size 166301): ns1__srmGetSpaceTokens : Request: Get space tokens. IP: <IP>. Client DN: <ID>
A0013 (size 7832): process_request : Connection from ::1
A0014 (size 481784): Result for request <TSK> is 'SRM_REQUEST_QUEUED'
A0015 (size 224258): ns1__srmReleaseFiles : Request: Release files. IP: <IP>. Client DN: <ID>(s): <URL> token: <TKN>
A0016 (size 203): Request <TSK> from Client <ID>=<ID> # Requested <NUM> SURL(s): '<URL> <URL>
A0017 (size 1927): __process_file_request<> : Protocol check failed, received some unsupported protocols
A0018 (size 1233): __process_file_request<> : Received - 4 - protocols, 2 are supported, 2 are not supported
A0019 (size 1927): __process_file_request<> : Some of the provided protocols are supported, proceeding
A0020 (size 6001): Request <TSK> from Client <ID>=<ID> # Requested token '<TKN>' on <NUM> SURL(s): '<URL> <URL>
A0021 (size 274952): Result for request <TSK> is 'SRM_FAILURE'
A0022 (size 51228): ns1__srmMv : Request: Mv. IP: <IP>. Client DN: <ID>_surl: <URL> to_surl: <URL>
A0023 (size 16334): Request <TSK> from Client <ID>=<ID> # Requested token '<TKN>' on <NUM> SURL(s): '<URL>
A0024 (size 9955): ns1__srmMkdir : Request: Mkdir. IP: <IP>. Client DN: <ID>(s): <URL>
A0025 (size 1091): ns1__srmAbortRequest : Request: Abort request. IP: <IP>. Client DN: <ID>
A0026 (size 2106): Request <TSK> from Client <ID>=<ID> # Requested token '<TKN>' on <NUM> SURL(s): '<URL> <URL> <URL>
A0027 (size 903): Result for request <TSK> is 'SRM_INTERNAL_ERROR'
A0028 (size 693): __process_file_request<> : Received - 5 - protocols, 3 are supported, 2 are not supported
A0029 (size 281): Result for request <TSK> is 'SRM_INVALID_PATH'
A0030 (size 831): ns1__srmGetSpaceMetaData : Request: Get space metadata. IP: <IP>. Client DN: <ID>
A0031 (size 823): Result for request <TSK> is 'SRM_INVALID_REQUEST'
A0032 (size 459): ns1__srmReleaseFiles : Request: Release files. IP: <IP>. Client DN: <ID>
A0033 (size 87): Request <TSK> from Client <ID>=<ID> # Requested <NUM> SURL(s): '<URL> <URL> <URL>
A0034 (size 9): Result for request <TSK> is 'SRM_DUPLICATION_ERROR'
A0035 (size 123): ns1__srmAbortFiles : Request: Abort files. IP: <IP>. Client DN: <ID>(s): <URL> token: <TKN>
A0036 (size 198): rpcResponseHandler_AbortFiles : arrayOfFileStatuses not specified by BE.
A0037 (size 202): Request <TSK> from Client <ID>=<ID> # Requested <NUM> SURL(s): '<URL> <URL> <URL> <URL> <*> TRUNCATED
A0038 (size 15): ns1__srmCheckPermission : Request: Check permission. IP: <IP>. Client DN: <ID>(s): <URL>
A0039 (size 15): Result for request <TSK> is 'SRM_NOT_SUPPORTED'
A0040 (size 44): Request <TSK> from Client <ID>=<ID> # Requested <NUM> SURL(s): '<URL> <URL> <URL> <*>
A0041 (size 1933): Request <TSK> from Client <ID>=<ID> # Requested token '<TKN>' on <NUM> SURL(s): '<URL> <URL> <URL> TRUNCATED
A0042 (size 37): ns1__srmAbortFiles : Request: Abort files. IP: <IP>. Client DN: <ID>(s): <URL> <URL> token: <TKN>
A0043 (size 1): Request <TSK> from Client <ID># Requested <NUM> SURL(s): '<URL>
A0044 (size 1): Request <TSK> from Client <ID># Requested token '<TKN>'
A0045 (size 3): rpcResponseHandler_ReleaseFiles : xml_arrayOfFileStatuses is empty
A0046 (size 61): Request <TSK> from Client <ID>=<ID> # Requested <NUM> SURL(s): '<URL> <URL> <URL> <*> TRUNCATED
A0047 (size 1): __process_file_request<> : Received - 5 - protocols, 2 are supported, 3 are not supported
A0048 (size 1): ns1__srmRmdir : Request: Rmdir. IP: <IP>. Client DN: <ID>(s): <URL>
A0049 (size 1): Result for request <TSK> is 'SRM_NON_EMPTY_DIRECTORY'
A0050 (size 27): ns1__srmAbortFiles : Request: Abort files. IP: <IP>. Client DN: <ID>(s): <URL> <URL> <URL> <*> TRUNCATED
A0051 (size 11): ns1__srmAbortFiles : Request: Abort files. IP: <IP>. Client DN: <ID>(s): <URL> <URL> <URL> TRUNCATED
A0052 (size 456): rpcResponseHandler_Ls : ERROR: XML-RPC Fault: RPC failed at server. Failed to invoke method ls in ...
                Invalid filesystem entry <*> Invalid argument (code: 0)
A0053 (size 2): rpcResponseHandler_Rm : ERROR: XML-RPC Fault: libcurl failed to execute the HTTP POST transaction, explaining:
                Failed connect to storm-atlas.cr.cnaf.infn.it:8080; Operation now in progress (code: -504)
A0054 (size 2): ns1__srmAbortFiles : Request: Abort files. IP: <IP>. Client DN: <ID>(s): <URL> <URL> <URL> <URL> <*> TRUNCATED
A0055 (size 1): ns1__srmAbortFiles : Request: Abort files. IP: <IP>. Client DN: <ID>(s): <URL> <URL> <URL> token: <TKN>
A0056 (size 2): storm::BolStatusRequest::loadFromDB() : No tokens found for token <TKN> and the requested SURLs
A0057 (size 1): rpcResponseHandler_PutDone : xml_arrayOfFileStatuses is empty
A0058 (size 2): rpcResponseHandler_Ls : ERROR: XML-RPC Fault: RPC failed at server. Failed to invoke method ls ... Communications link failure
A0059 (size 2): rpcResponseHandler_Ls : ERROR: XML-RPC Fault: libcurl failed to execute the HTTP POST transaction, explaining:
                                 couldn't connect to host (code: -504)
A0060 (size 5): logConfiguration : Starting StoRM frontend as user: storm
A0061 (size 5): logConfiguration : ---------------------- Configuration ------------------
A0062 (size 90): logConfiguration : <ID>
A0063 (size 5): logConfiguration : fe.gsoap.send_<ID>
A0064 (size 5): logConfiguration : fe.gsoap.recv_<ID>
A0065 (size 5): logConfiguration : argus-pepd-endpoint=
A0066 (size 5): logConfiguration : xmlrpc <ID>
A0067 (size 5): logConfiguration : -------------------------------------------------------
A0068 (size 5): initSoap : Mapping disabled
A0069 (size 5): main : StoRM frontend successfully started...
```

And the 44 classes obtained from the post processing of the first 81 templates extracted:

```
A0001 (size 1392640): process_request : Connection from <IP>
A0002 (size 196048): ns1__srmPutDone : Request: Put done. IP: <IP>. Client DN: <ID>(s): <URL> token: <TKN>
A0003 (size 2997244): Result for request <TSK> is 'SRM_SUCCESS'
A0004 (size 1593721): ns1__srmLs : Request: Ls. IP: <IP>. Client DN: <ID>
A0005 (size 1929781): Request <TSK> from Client IP='<IP>' Client DN=<ID># Requested token '<TKN>' on <NUM> SURL(s): '<URL>
A0006 (size 468714): Result for request <TSK> is 'SRM_REQUEST_INPROGRESS'
A0007 (size 449698): ns1__srmPing : Request: Ping. IP: <IP>. Client DN: <ID>
A0008 (size 1009462): Result for request <TSK> is 'SRM_REQUEST_QUEUED'. # Produced request token: '<TKN>'
A0009 (size 107104): ns1__srmRm : Request: Rm. IP: <IP>. Client DN: <ID>(s): <URL>
A0010 (size 200056): ns1__srmGetSpaceTokens : Request: Get space tokens. IP: <IP>. Client DN: <ID>
A0011 (size 253430): ns1__srmReleaseFiles : Request: Release files. IP: <IP>. Client DN: <ID>(s): <URL> token: <TKN>
A0012 (size 2327): __process_file_request<> : Protocol check failed, received some unsupported protocols
A0013 (size 2327): __process_file_request<> : Received - 4 - protocols, 2 are supported, 2 are not supported
A0014 (size 2327): __process_file_request<> : Some of the provided protocols are supported, proceeding
A0015 (size 327364): Result for request <TSK> is 'SRM_FAILURE'
A0016 (size 60564): ns1__srmMv : Request: Mv. IP: <IP>. Client DN: <ID>_surl: <URL> to_surl: <URL>
A0017 (size 11676): ns1__srmMkdir : Request: Mkdir. IP: <IP>. Client DN: <ID>(s): <URL>
A0028 (size 1362): ns1__srmAbortRequest : Request: Abort request. IP: <IP>. Client DN: <ID>
A0019 (size 903): Result for request <TSK> is 'SRM_INTERNAL_ERROR'
A0020(size 395): Result for request <TSK> is 'SRM_INVALID_PATH'
A0021 (size 1007): ns1__srmGetSpaceMetaData : Request: Get space metadata. IP: <IP>. Client DN: <ID>
A0022 (size 575): Result for request <TSK> is 'SRM_INVALID_REQUEST'
A0023 (size 16): Result for request <TSK> is 'SRM_DUPLICATION_ERROR'
A0024 (size 233): ns1__srmAbortFiles : Request: Abort files. IP: <IP>. Client DN: <ID>(s): <URL> token: <TKN>
A0025 (size 232): rpcResponseHandler_AbortFiles : arrayOfFileStatuses not specified by BE.
A0026 (size 21): ns1__srmCheckPermission : Request: Check permission. IP: <IP>. Client DN: <ID>(s): <URL>
A0027 (size 21): Result for request <TSK> is 'SRM_NOT_SUPPORTED'
A0028 (size 1): rpcResponseHandler_ReleaseFiles : xml_arrayOfFileStatuses is empty
A0029 (size 1): ns1__srmRmdir : Request: Rmdir. IP: <IP>. Client DN: <ID>(s): <URL>
A0030 (size 1): Result for request <TSK> is 'SRM_NON_EMPTY_DIRECTORY'
A0031 (size 252): rpcResponseHandler_Ls : ERROR: XML-RPC Fault: RPC failed at server. Failed to invoke method ls in class ... Invalid argument (code: 0)
A0032 (size 2): rpcResponseHandler_Rm : ERROR: XML-RPC Fault: libcurl failed to execute the HTTP POST transaction, explaining:
                Failed connect to storm-atlas.cr.cnaf.infn.it:8080; Operation now in progress (code: -504)
A0033 (size 8): storm::BolStatusRequest::loadFromDB() : No tokens found for token <TKN> and the requested SURLs
A0034 (size 1): rpcResponseHandler_PutDone : xml_arrayOfFileStatuses is empty
A0035 (size 1): logConfiguration : Starting StoRM frontend as user: storm
A0036 (size 1): logConfiguration : ---------------------- Configuration ------------------
A0037 (size 18): logConfiguration : <ID>
A0038 (size 1): logConfiguration : fe.gsoap.send_<ID>
A0039 (size 1): logConfiguration : fe.gsoap.recv_<ID>
A0040 (size 1): logConfiguration : argus-pepd-endpoint=
A0041 (size 1): logConfiguration : xmlrpc <ID>
A0042 (size 1): logConfiguration : -------------------------------------------------------
A0043 (size 1): initSoap : Mapping disabled
A0044 (size 1): main : StoRM frontend successfully started...
```

5.2.2 Model Loss and Prediction

In order to check wether the LogKey model is capable of effectively find anomalies in a log sequence, the test run consist of two files: one containing normal sequences, the other containing abnormal sequences as defined in the "Dataset Preparation" section. The model is tested on both files, asking it to find anomalies in a perfectly normal sequence gives a measure of the precision of the network: if few anomalies get signaled in a normal execution sequence we have a measure of how many true anomaly are found when used on a unlabeled dataset (not divided in normal and abnormal). The other test is more straight forward: we request the model to confront the pattern it learned, which is supposed to be normal execution patterns, with a collection of anomalous patterns; if the count of anomalies found in this abnormal test is high it means that the model is able to find anomalies in an unlabeled dataset.
Using this [27] implementation of DeepLog's LogKey model which make use of torch, an open source machine learning framework, three models have been studied, all consisting a two-layer LSTM with hidden sizes 32, 64, 96; for each model were used the data sets obtained using the three different results of the clustering (44, 69 and 81 classes). Each of the resulting nine model has then been trained using different window sizes.
To evaluate the effectiveness of the LogKey models three statistics are used:

$$Precision = \frac{TruePositives}{TruePositives + FalsePositives}$$

measures the percentage of true anomalies among all anomalies detected;

$$Recall = \frac{TruePositives}{TruePositives + FalseNegatives}$$

that measures the percentage of anomalies in the data set being detected, assuming that the ground truth in known.

$$F - Measure = \frac{2 * Precision * Recall}{Precision + Recall}$$

The harmonic mean of Precision and Recall gives an overall accuracy score.
The sequence from the normal test set is paired with it's successive log key as label and fed through the model that outputs a list of candidates for the next log key in the sequence, if the label is not among the top candidates it's then counted as False Positive: inside a sequence that is known to be completely normal, an anomaly has been detected. Reversely for the abnormal test set if a label is not among the top candidates is counted as True Positive being an anomaly detected in a sequence that is abnormal.
False Negatives are then calculated as the length of the abnormal test set, seen as tuples of sequences of fixed window size and corresponding labels, minus the True Positives.
Here follows the train loss graphs for different models obtained by varying hidden size and window size followed the Precision, Recall and F-Measure that each model obtained on the same test set.

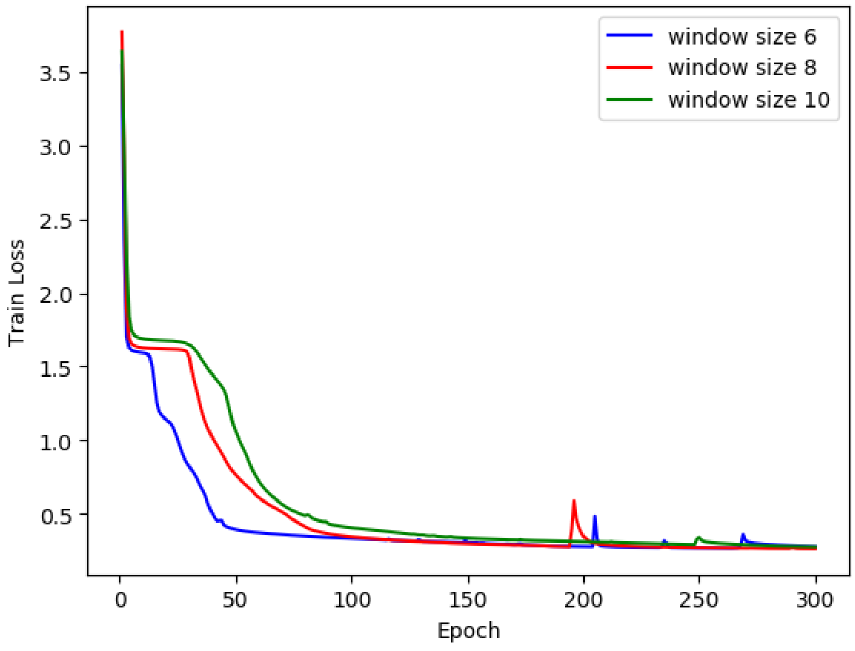


Figure 11: Train Loss for LogKey model using 44 classes with hidden size 32 at window size 6, 8 and 10

Precision	5 Candidates	3 Candidates	1 Candidate
Window size 6	97.77%	96.17%	69.54%
Window size 8	96.91%	96.82%	69.91%
Window size 10	96.79%	96.18%	70.43%
Recall	5 Candidates	3 Candidates	1 Candidate
Window size 6	76.89%	99.12%	100.0%
Window size 8	99.56%	99.89 %	100.0%
Window size 10	99.01%	99.34%	99.67%
F-measure	5 Candidates	3 Candidates	1 Candidate
Window size 6	86.08%	97.63%	82.03%
Window size 8	98.22%	98.33%	82,289%
Window size 10	97.89%	97.74%	82.54%

Table 2: Precision, Recall and F-measure for LogKey model using 44 classes with hidden size 32 at different window sizes and number of candidates

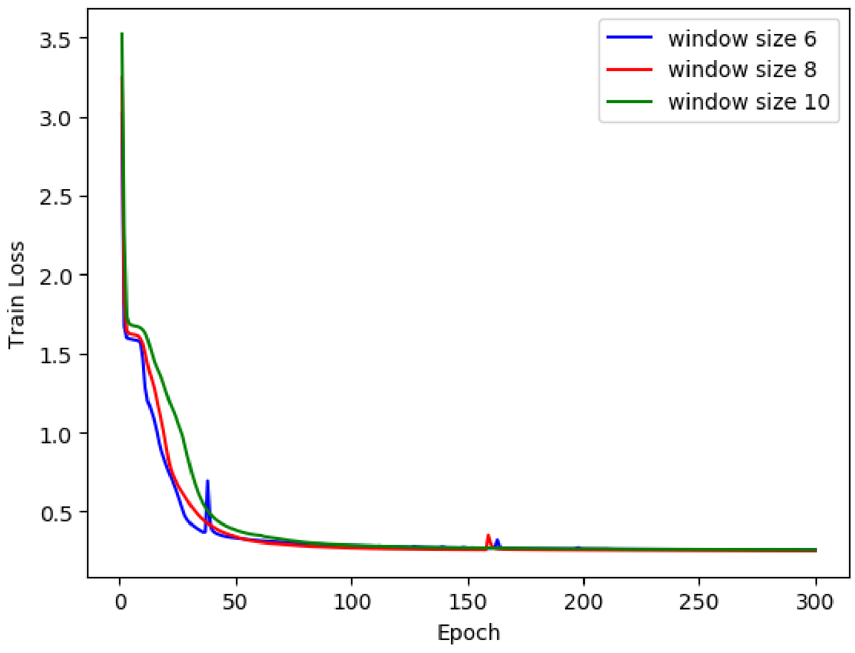


Figure 12: Train Loss for LogKey model using 44 classes with hidden size 64 at window size 6, 8 and 10

Precision	5 Candidates	3 Candidates	1 Candidate
Window size 6	99.26%	99.46%	69.59%
Window size 8	99.24%	98.70%	70.02%
Window size 10	98.81%	98.12%	70.57%
Recall	**5 Candidates**	**3 Candidates**	**1 Candidate**
Window size 6	73.82%	99.89%	100.0%
Window size 8	99.78%	100.0%	100.0%
Window size 10	90.58%	97.37%	99.78%
F-measure	**5 Candidates**	**3 Candidates**	**1 Candidate**
Window size 6	84.67%	99.67%	81.07%
Window size 8	99.51%	99.35%	82.36%
Window size 10	94.51%	97.76%	82.67%

Table 3: Precision, Recall and F-measure for LogKey model using 44 classes with hidden size 64 at different window sizes and number of candidates

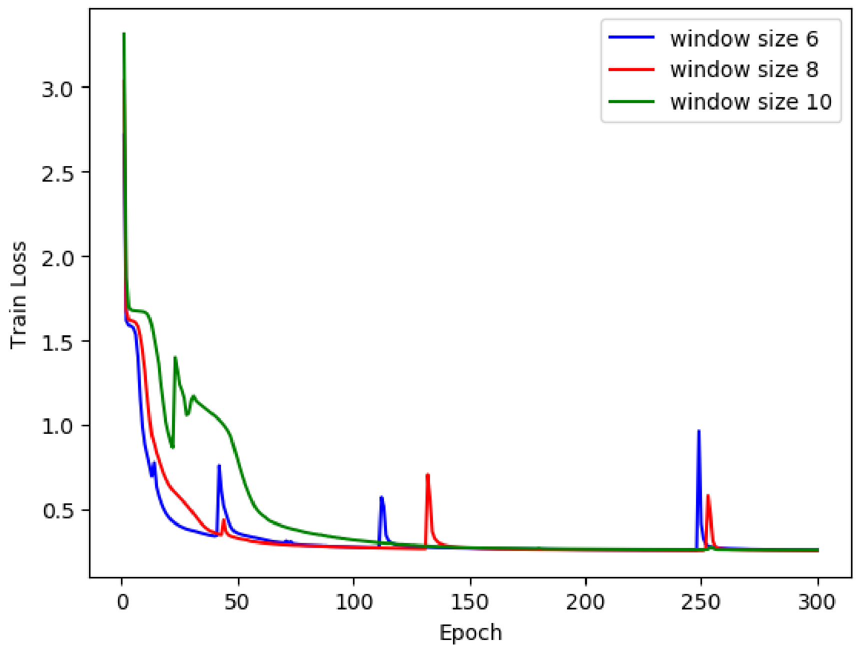


Figure 13: Train Loss for LogKey model using 44 classes with hidden size 96 at window size 6, 8 and 10

Precision	5 Candidates	3 Candidates	1 Candidate
Window size 6	99.25%	98.17%	69.59%
Window size 8	98.31%	96.31%	69.91%
Window size 10	97.61%	97.32%	70.56%
Recall	5 Candidates	3 Candidates	1 Candidate
Window size 6	86.86%	99.89%	100.0%
Window size 8	95.40%	100.0%	100.0%
Window size 10	98.58%	99.23%	100.0%
F-measure	5 Candidates	3 Candidates	1 Candidate
Window size 6	92.64%	99.02%	82.07%
Window size 8	96.83%	98.12 %	82.29%
Window size 10	98.09%	98.27%	82.74%

Table 4: Precision, Recall and F-measure for LogKey model using 44 classes with hidden size 96 at different window sizes and number of candidates

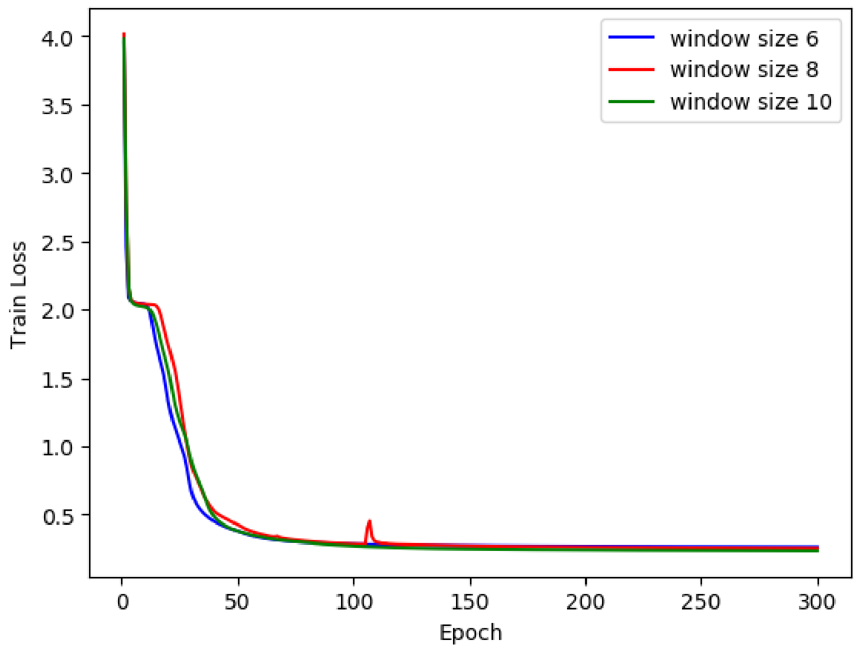


Figure 14: Train Loss for LogKey model using 69 classes with hidden size 32 at window size 6, 8 and 10

Precision	5 Candidates	3 Candidates	1 Candidate
Window size 6	99.77%	99.08%	66.33%
Window size 8	98.89%	97.72 %	66.95%
Window size 10	97.88%	96.68%	68.07%
Recall	5 Candidates	3 Candidates	1 Candidate
Window size 6	99.88%	99.88%	100.0%
Window size 8	82.31%	84.05%	100.0%
Window size 10	74.80 %	77.46 %	99.31%
F-measure	5 Candidates	3 Candidates	1 Candidate
Window size 6	99.83%	99.48%	79.76%
Window size 8	89.84%	90.37%	80.20%
Window size 10	84.80%	86.01%	80.77%

Table 5: Precision, Recall and F-measure for LogKey model using 69 classes with hidden size 32 at different window sizes and number of candidates

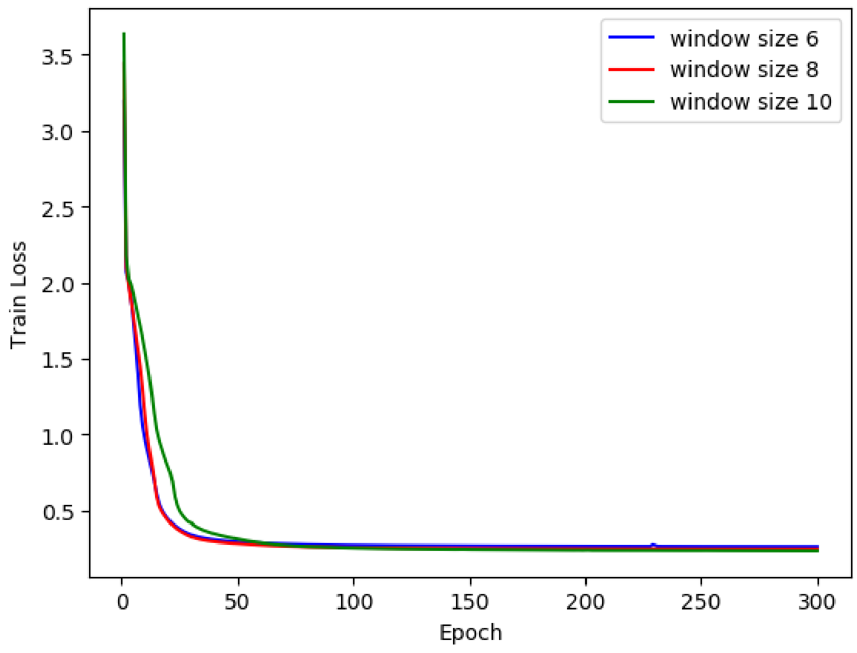


Figure 15: Train Loss for LogKey model using 69 classes with hidden size 64 at window size 6, 8 and 10

Precision	5 Candidates	3 Candidates	1 Candidate
Window size 6	99.87%	99.19%	66.49%
Window size 8	99.74%	99.14%	67.53%
Window size 10	98.01%	97.81%	68.25%
Recall	**5 Candidates**	**3 Candidates**	**1 Candidate**
Window size 6	86.24%	98.84%	100.0%
Window size 8	89.71%	93.64%	100.0%
Window size 10	73.87%	82.78%	99.65%
F-measure	**5 Candidates**	**3 Candidates**	**1 Candidate**
Window size 6	92.56%	99.02%	79.87%
Window size 8	94.46%	96.31%	80.62%
Window size 10	84.25%	89.67%	81.02%

Table 6: Precision, Recall and F-measure for LogKey model using 69 classes with hidden size 64 at different window sizes and number of candidates

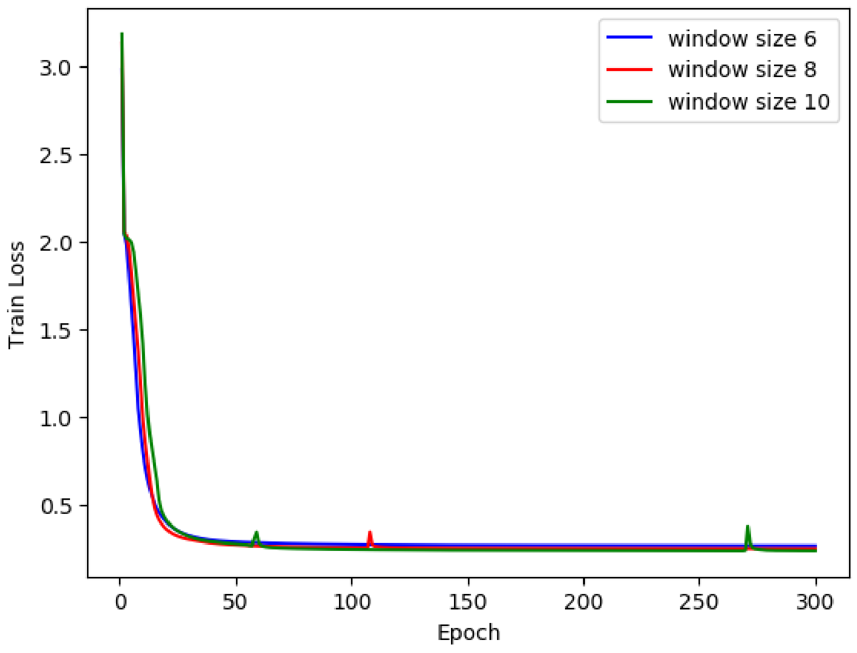


Figure 16: Train Loss for LogKey model using 69 classes with hidden size 96 at window size 6, 8 and 10

Precision	5 Candidates	3 Candidates	1 Candidate
Window size 6	99.88%	99.08%	66.49%
Window size 8	99.87%	99.24%	67.58%
Window size 10	97.80%	97.45%	68.15%
Recall	**5 Candidates**	**3 Candidates**	**1 Candidate**
Window size 6	99.88%	100.0%	100.0%
Window size 8	90.41%	90.41%	100.0%
Window size 10	66.82%	70.64%	99.42%
F-measure	**5 Candidates**	**3 Candidates**	**1 Candidate**
Window size 6	99.88%	99.54%	79.87%
Window size 8	94.90%	94.62%	80.65%
Window size 10	79.40%	81.90%	80.87%

Table 7: Precision, Recall and F-measure for LogKey model using 69 classes with hidden size 96 at different window sizes and number of candidates

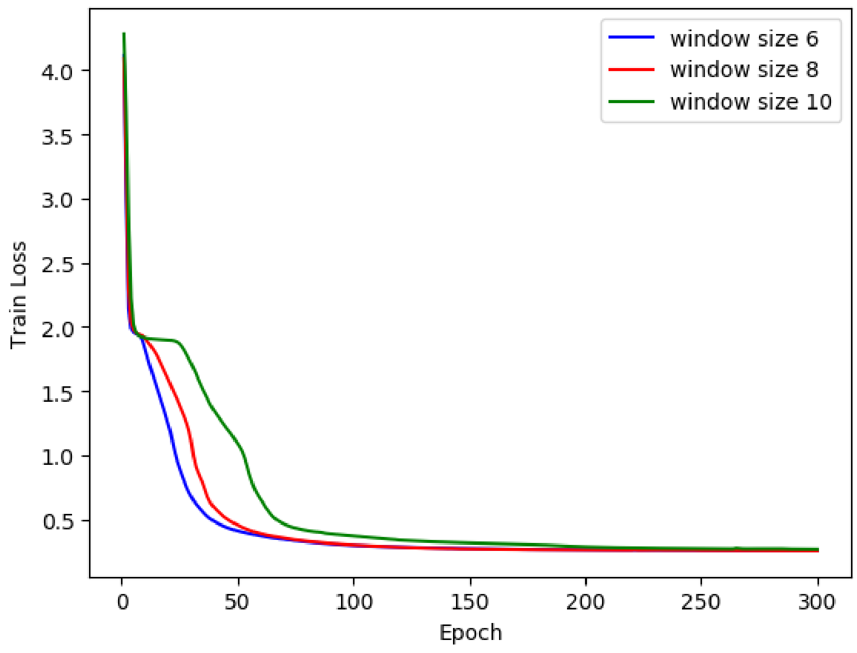


Figure 17: Train Loss for LogKey model using 81 classes with hidden size 32 at window size 6, 8 and 10

Precision	5 Candidates	3 Candidates	1 Candidate
Window size 6	99.43%	97.12%	69.01%
Window size 8	97.58%	96.48%	69.73%
Window size 10	96.73%	96.43%	70.34%
Recall	5 Candidates	3 Candidates	1 Candidate
Window size 6	93.99%	100.0%	100.0%
Window size 8	82.10%	100.0%	100.0%
Window size 10	76.10%	92.61%	99.89%
F-measure	5 Candidates	3 Candidates	1 Candidate
Window size 6	96.64%	98.57%	81.66%
Window size 8	89.17%	98.21%	82.17%
Window size 10	85.18%	94.48%	82.55%

Table 8: Precision, Recall and F-measure for LogKey model using 81 classes with hidden size 32 at different window sizes and number of candidates

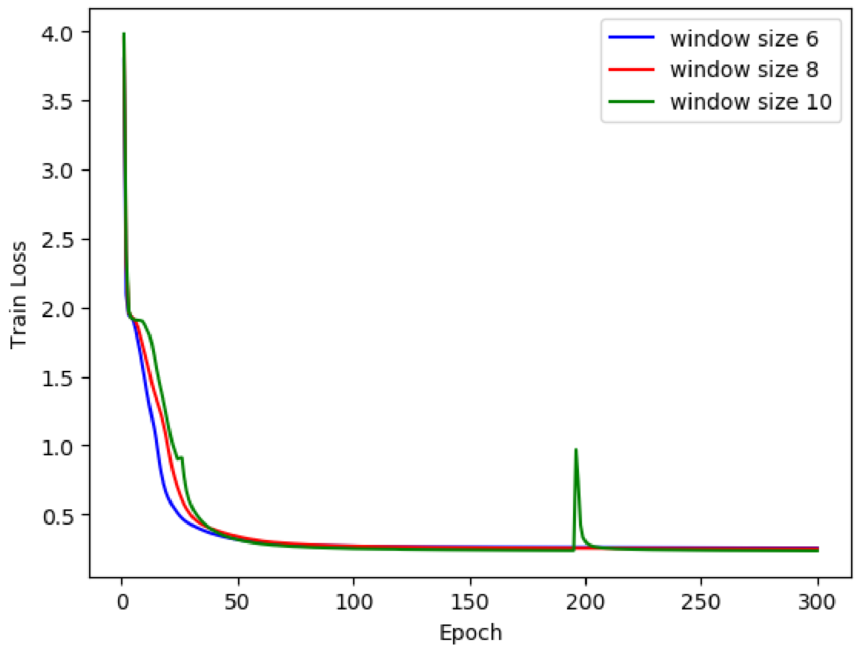


Figure 18: Train Loss for LogKey model using 81 classes with hidden size 64 at window size 6, 8 and 10

Precision	5 Candidates	3 Candidates	1 Candidate
Window size 6	98.77%	98.42%	69.16%
Window size 8	99.43%	98.91%	69.63%
Window size 10	71.33%	69.58%	69.55%
Recall	5 Candidates	3 Candidates	1 Candidate
Window size 6	94.64%	100.0%	100.0%
Window size 8	74.81%	87.87%	100.0%
Window size 10	97.86%	99.04%	100.0%
F-measure	5 Candidates	3 Candidates	1 Candidate
Window size 6	96.66%	99.20%	81.77%
Window size 8	85.38%	93.02%	82.09%
Window size 10	82.51%	91.73%	81.34%

Table 9: Precision, Recall and F-measure for LogKey model using 81 classes with hidden size 64 at different window sizes and number of candidates

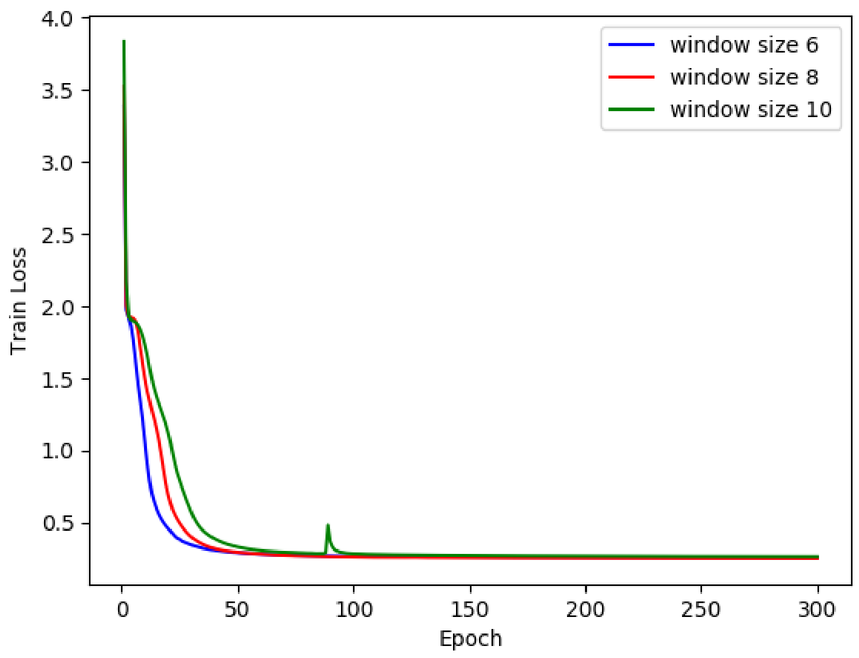


Figure 19: Train Loss for LogKey model using 81 classes with hidden size 96 at window size 6, 8 and 10

Precision	5 Candidates	3 Candidates	1 Candidate
Window size 6	99.43%	98.63%	69.21%
Window size 8	99.79%	99.25%	69.78%
Window size 10	98.04%	97.83%	70.36%
Recall	**5 Candidates**	**3 Candidates**	**1 Candidate**
Window size 6	93.99%	100.0%	100.0%
Window size 8	99.89%	100.0%	100.0%
Window size 10	69.56%	72.35%	99.47%
F-measure	**5 Candidates**	**3 Candidates**	**1 Candidate**
Window size 6	96.64%	99.31%	81.81%
Window size 8	99.84%	99.63%	82.20%
Window size 10	81.38%	83.18%	82.42%

Table 10: Precision, Recall and F-measure for LogKey model using 81 classes with hidden size 96 at different window sizes and number of candidates

The graphs, representing the train loss as the epochs go from 0 to 300, tend to converge to a value of ~ 0.2 somewhere between epoch 50 and 100, and to keep that value until the end of the training. The spikes seen along the curves are an unavoidable consequence of Mini-Batch Gradient Descent in Adam, the optimizer used in this model with $batch\_size = 2048$.
For some of the models the train loss presents a plateau around the value of $trainloss = 2$ which becomes more evident as the window size increases (see Figure 17). This zone of not learning is however surpassed in all the cases and the train loss converges close to his value in around epoch 200.
The results in all tables highlight the fact that asking the model to choose the next log key in a log sequence by drawing from a shorter candidate list boosts the Recall value, which approaches 100% for a single candidate to the expenses of Precision, which tends to reach the highest value with 5 Candidates for every model.
To find which model obtained the best overall results we pick those with the highest F-Measure, for example from the results in Table 2, representing the model with 44 classes and hidden size 32, we deduce that the best configuration among those considered in the table is the one with window size 8 and 3 candidates as it has the highest F-Measure (98.33%). By taking the F-Measure as reference for the overall accuracy of the model, we can sort out the best performing ones as those with the highest F:

F-Measure	Number of classes	Hidden size	Window size	Candidates
99.88%	69	96	8	5
99.84%	81	96	8	5
99.83%	69	32	6	5
99.68%	44	64	6	3
99.63%	81	96	8	3
99.54%	69	96	6	3
99.50%	44	64	8	5

Table 11: Recap of the best performing models with $F - Measure \geq 99.50\%$

6 Conclusions

We have seen the highly refined and complicated ATLAS' data acquisition system, the journey that the detector's signals undertake to the Tier-0 and trough the WLCG to the Tier-1 Tier-2 and Tier-3. This Grid approach to computation renders necessary for the main nodes of the network, the Tier 0 and 1s, to be as stable as possible since so many scientists and research groups rely on these infrastructures.
We introduced StoRM, the Storage Resource Manager at INFN-CNAF Tier-1 in Bologna; StoRM's front end server's log were used for the analysis. We investigated some Log Analysis tools such as Drain3 for mining log templates out of raw log entries and the LogKey model of DeepLog for the anomaly detection on log sequences. Drain3 has revealed to be highly customizable and relatively lightweight, compared to other parsing methods; its availability on PyPi makes it easy to install using pip and the configuration via configparser allows the users to adapt the research tree and regular expressions to their framework. Drain3 can also work in a streaming fashion, by feeding log lines one by one or as they are generated.
Since the expected number of log templates wasn't known first, different similarity thresholds were used to mine them, obtaining 3 data sets representing the same system execution, but differently enumerated in log keys.
As expected this difference in enumeration doesn't keep DeepLog's LogKey model from learning the underlying patterns in the sequences since the results were consistent in all three cases (44, 69, 81 classes). The promising results of this part of DeepLog make it a candidate tool for anomaly detection, once the parameter model gets implemented.

7 Aknowledgments

The Data for this work was kindly provided by INFN - CNAF who also granted access to the Tier-3 infrastructure, and was fundamental for this thesis.
All the software used is open source, I thank those who keep these programs freely hosted on github and under MIT License.

References

[1] CERN, Accelerator Complex at CERN http://public-archive.web.cern.ch/public-archive/en/research/AccelComplex-en.html.

[2] CERN. Alice. https://home.cern/science/experiments/alice.

[3] CERN. Atlas. https://home.cern/science/experiments/atlas.

[4] CERN. Cms. https://home.cern/science/experiments/cms.

[5] CERN. Lhcb. https://home.cern/science/experiments/lhcb.

[6] CERN. Totem. https://totem-experiment.web.cern.ch/.

[7] CERN. Lhcf. https://home.cern/science/experiments/lhcf.

[8] CERN. Experiments. https://home.cern/science/experiments.

[9] The ATLAS Collaboration. Atlas detector overview. https://jinst.sissa.it/LHC/ATLAS/ch01.pdf.

[10] Masetti Lucia. A high-granularity timing detector for the phase-ii upgrade of the atlas calorimeter system. Jan 2017.

[11] CERN. High luminosity lhc. https://home.cern/science/accelerators/high-luminosity-lhc.

[12] CERN, ATLAS Detector image http://opendata.atlas.cern/books/current/get-started/_book/GLOSSARY.html

[13] The ATLAS Collaboration. The atlas experiment at the cern large hadron collider. *Journal of Instrumentation*, 3(08):S08003–S08003, aug 2008.

[14] The ATLAS Collaboration. Trigger - daq. https://atlas.cern/discover/detector/trigger-daq.

[15] A Ruiz Martinez. The run-2 atlas trigger system. *Journal of Physics: Conference Series*, 762:012003, oct 2016.

[16] Johannes Elmsheuser et al. Evolution of the ATLAS analysis model for Run-3 and prospects for HL-LHC. *EPJ Web Conf.*, 245:06014, 2020.

[17] D Adams, D Barberis, C P Bee, R Hawkings, S Jarp, R Jones, D Malon, L Poggioli, G Poulard, D Quarrie, and T Wenaus. The ATLAS Computing Model. Technical Report ATL-SOFT-2004-007. ATL-COM-SOFT-2004-009.CERN-ATL-COM-SOFT-2004-009. CERN-LHCC-2004-037-G-085, CERN, Geneva, Dec 2004.

[18] INFN-CNAF. Tier-1 data center. https://www.cnaf.infn.it/wlcg-tier-1-data-center/.

[19] A. Brunengo et al. Commissioning of a StoRM based data management system for ATLAS at INFN sites. *J. Phys. Conf. Ser.*, 219:062042, 2010.

[20] IBM. Gpfs. http://publib.boulder.ibm.com/infocenter/clresctr/vxrx/index.jsp?topic= /com.ibm.cluster.gpfs.doc/gpfsbooks.html

[21] S. Hochreiter and J. Schmidhuber. Long Short Term Memory. Neural Computation 9(8):17351780, 1997

[22] Min Du, Feifei Li, Guineng Zheng, and Vivek Srikumar. Deeplog: Anomaly detection and diagnosis from system logs through deep learning. pages 1285–1298, 10 2017.

[23] P. He, J. Zhu, S. He, J. Li, and M. R. Lyu. An evaluation study on log parsing and its use in log mining. In *2016 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 654–661, 2016.

[24] P. He, J. Zhu, Z. Zheng, and M. R. Lyu. Drain: An online log parsing approach with fixed depth tree. pages 33–40, 2017.

[25] IBM. Drain3. https://github.com/IBM/Drain3.

[26] Pandas. Pandas. https://pandas.pydata.org/.

[27] Wu Yifan. DeepLog. https://github.com/wuyifan18/DeepLog