Alma Mater Studiorum · Università di Bologna

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

Analisi di log files prodotti dal sistema StoRM usato dall'esperimento ATLAS attraverso algoritmi di clustering

Relatore:
Prof. Lorenzo Rinaldi

Presentata da:
Elisabetta Ferri

Anno Accademico 2021/2022

Alla mia famiglia che mi è stata accanto e mi ha spronato in questo percorso e alla professoressa Giacconi che è riuscita a trasmettermi il suo amore per la Fisica

Abstract

L'esperimento ATLAS, come gli altri esperimenti che operano al Large Hadron Collider producono Petabytes di dati ogni anno, che devono poi essere archiviati ed elaborati. Inoltre gli esperimenti si sono proposti di rendere accessibili questi dati in tutto il mondo. In risposta a questi bisogni è stato progettato il Worldwide LHC Computing Grid che combina la potenza di calcolo e le capacità di archiviazione di più di 170 siti sparsi in tutto il mondo.

Nella maggior parte dei siti del WLCG sono state sviluppate tecnologie per la gestione dello storage, che si occupano anche della gestione delle richieste da parte degli utenti e del trasferimento dei dati. Questi sistemi registrano le proprie attività in logfiles, ricchi di informazioni utili agli operatori per individuare un problema in caso di malfunzionamento del sistema.

In previsione di un maggiore flusso di dati nei prossimi anni si sta lavorando per rendere questi siti ancora più affidabili e uno dei possibili modi per farlo è lo sviluppo di un sistema in grado di analizzare i file di log autonomamente e individuare le anomalie che preannunciano un malfunzionamento. Per arrivare a realizzare questo sistema si deve prima individuare il metodo più adatto per l'analisi dei file di log.

In questa tesi viene studiato un approccio al problema che utilizza l'intelligenza artificiale per analizzare i logfiles, più nello specifico viene studiato l'approccio che utilizza dell'algoritmo di clustering K-means.

Indice

Introduzione

I moderni esperimenti di fisica e in particolare quelli di fisica delle particelle sono costituiti da apparati di misura molto complessi, il che si traduce in un flusso molto elevato di dati, che richiede delle infrastrutture informatiche con un elevata potenza di calcolo e di archiviazione. Questo lavoro si concentra su uno degli esperimenti più noti di questo ambito, ossia il Large Hadron Collider e in particolare sull'esperimento ATLAS. Per la gestione di questi dati è stata creata un'infrastruttura grid, che permette di condividere i dati degli esperimenti tra centri di calcolo presenti in più di 70 paesi e a disposizione di più di 12 200 ricercatori. Maggiori informazioni riguardo agli esperimenti e all'infrastruttura di grid sono presentate nel capitolo 1.

In questa tesi ci si concentra su una di queste strutture, il Tier-1 che si trova presso l'INFN-CNAF di Bologna in cui vengono gestiti un quantità di dati dell'ordine dei 10 Petabytes all'anno. Questa struttura oltre ad immagazzinare i dati provenienti dagli esperimenti deve anche condividerli con gli utenti che ne fanno richiesta. A tale scopo viene utilizzato il servizio StoRM che gestisce l'archiviazione e il trasferimento dei file. Questo servizio produce a sua volta dei file di log in cui monitora i processi gestiti dal servizio. Questi file vengono visionati da operatori umani, per individuare le anomalie a seguito di un malfunzionamento del servizio; questa analisi però non è efficiente sopratutto in previsione dell'aumento dei dati prodotti dagli esperimenti e di conseguenza dei trasferimenti. Per ovviare a questo problema si vogliono implementare degli algoritmi di machine learning per velocizzare il lavoro degli operatori umani e creare in futuro un sistema di monitoraggio autonomo.

Una breve introduzione al machine learning è riportata nel capitolo 2 insieme alla descrizione dell'algoritmo di clustering che si è scelto di utilizzare per questo lavoro di tesi. Nel capitolo 3 sono riportate le tecniche utilizzare per la creazione del data set a partire dai file di log e l'implementazione dell'algoritmo. Infine nel capitolo 4 vengono riportati i risultati prodotti dall'applicazione dell'algoritmo e la loro discussione.

Capitolo 1

Esperimenti di Fisica al Large Hadron Collider

Al giorno d'oggi la teoria principale nell'ambito della fisica delle particelle fondamentali è il Modello Standard. Questa teoria fu formulata negli anni '70 e da allora è stata modificata diverse volte, integrando per esempio il meccanismo di Higgs. Questo modello utilizza la teoria quantistica dei campi per descrive tre delle quattro interazioni fondamentali (l'interazione forte, elettromagnetica e debole) e i costituenti fondamentali della materia divisi in due famiglie, quarks e leptoni: secondo il modello sono presenti dodici campi di materia divisi fra sei leptoni e sei quarks, a cui si aggiungono tre campi di forza, che corrispondono alle tre interazioni, ciascuna mediata da bosoni; il bosone del campo elettromagnetico è il fotone, quelli del campo debole sono i bosoni massivi W e Z e quelli del campo forte sono i gluoni. A questi campi si aggiunge poi il campo di Higgs mediato dal suo bosone. Il grande valore di questa teoria risiede nelle importanti previsioni teoriche, poi verificate sperimentalmente come per esempio l'esistenza dei bosoni W e Z, mediatori dell'interazione debole, o l'esistenza dei quarks top e charm.

I laboratori del CERN (*Conseil Européen pour la Recherche Nucléaire*), situati a Ginevra in Svizzera, sono il punto di riferimento per la ricerca nell'ambito della fisica delle alte energie e lo studio del Modello Standard. Questo centro nacque dopo la seconda guerra mondiale dall'esigenza di riformare la comunità scientifica europea e creare una collaborazione fra gli stati che avesse come scopo lo studio delle particelle fondamentali. Con il passare del tempo gli esperimenti del CERN hanno richiesto lo sviluppo di tecnologie sempre più sofisticate e fra queste sono molto note le invenzioni del touchscreen e del World Wide Web, poi diffuse e impiegate comunemente in tutto il resto del mondo. Gli esperimenti del CERN hanno portato molti progressi alla fisica fondamentale, fra cui la ben nota scoperta del bosone di Higgs annunciata nel 2012.

1.1 Esperimenti del Large Hadron Collider (LHC)

Gli esperimenti del CERN si servono di un sistema di acceleratori in Fig.1.1, di cui l'ultima aggiunta è il Large Hadron Collider (LHC), inaugurato nel 2008, che consiste in un anello di 27 chilometri a circa 100 m di profondità e rimane ad oggi l'acceleratore più grande e potente costruito [1].

Il sistema di acceleratori funziona a cascata, per cui una macchina accelera un fascio di particelle fino a una certa energia per poi iniettarlo nella macchina successiva che porterà il fascio ad un'energia ancora maggiore e così via; l'ultimo acceleratore del sistema è LHC che è composto da strutture di accelerazione e magneti superconduttori disposti per tutta la lunghezza dell'anello. Le prime aumentano l'energia del fascio accelerandolo, mentre i magneti servono a curvare la traiettoria in modo che segua la forma dell'anello. Oltre a regolare la traiettoria i magneti sono anche impiegati per focalizzare i fasci e aumentare le probabilità di collisione. Questi magneti sono molto potenti e producono un campo di 8 T; sono costruiti impiegando bobine di cavi elettrici particolari che operano nello stato di superconduttori, ossia sono in grado di condurre con enorme efficienza perché la loro resistenza è praticamente nulla. Questo stato è ottenuto mantenendo i magneti a una temperatura di $2K$ (-271.3°C) grazie a un sistema di raffreddamento a elio liquido.

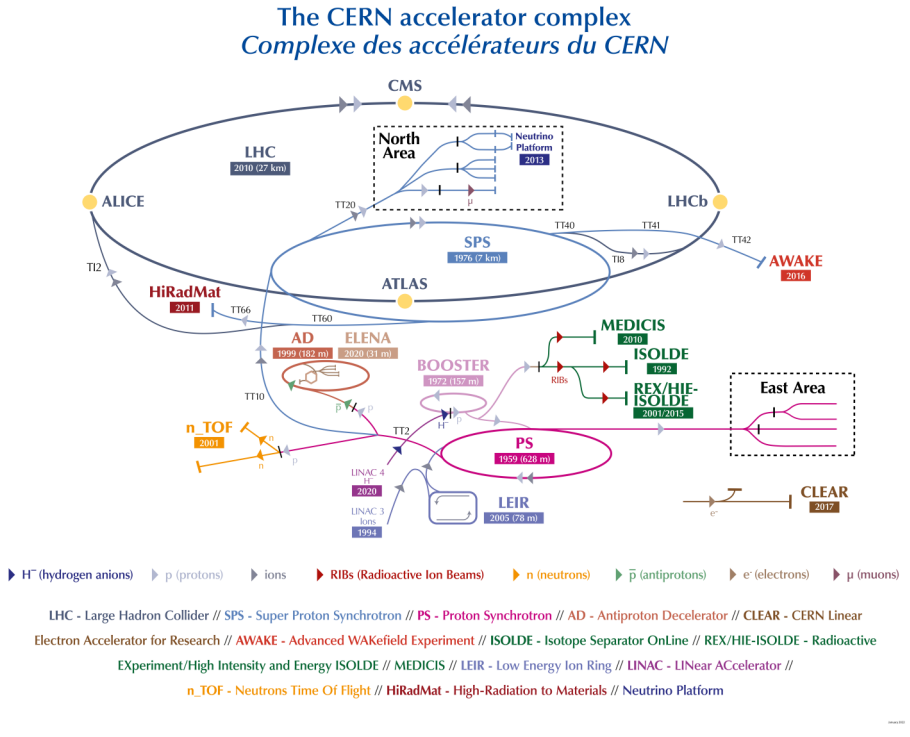

Figura 1.1: *Complesso di acceleratori situato ai laboratori del CERN*

Le collisioni avvengono perché il sistema di acceleratori produce due fasci opposti, con velocità prossime a quella della luce, che viaggiano in tubi mantenuti a vuoto ultra alto finché gli ioni di due fasci non collidono. Gli acceleratori attuali sono grado di portare ciascun fascio fino a un'energia di 6.8 TeV, da cui si ottengono collisioni da 13.6 TeV [2]. LHC è progettato in modo che queste collisioni avvengano in quattro punti particolari lungo l'anello, dove sono posizionati quattro rivelatori di particelle: ATLAS [3], CMS [4], ALICE [5] e LHCb [6]. Oltre a questi quattro a LHC sono presenti molti altri esperimenti, fra cui l'esperimento MoEDAL-MAPP [7], che utilizza dei rivelatori situati vicino a LHCb per cercare gli ipotetici monopoli magnetici, oppure FASER [8], l'ultimo esperimento aggiunto a LHC che è situato a 480 metri da ATLAS per la ricerca di nuove particelle leggere e lo studio dei neutrini.

Il terzo RUN di LHC è attualmente in corso e terminerà nel 2026, quando inizieranno i lavori per avviare nel 2029 l'High-Luminosity Large Hadron Collider (HL-LHC) [9], che si propone di aumentare la luminosità del fascio di un fattore 10. La luminosità è un fattore molto importante per un acceleratore perchè è proporzionale alla frequenza di collisioni, quindi al numero di dati che ogni esperimento può raccogliere. L'aumento delle collisioni comporta la possibilità di studiare in maggiore dettaglio fenomeni rari, per esempio HL-LHC permetterà di produrre 15 milioni di bosoni di Higgs all'anno, che sono cinque volte quelli prodotti nel 2017, inoltre si potrebbero anche osservare nuovi fenomeni rari.

1.1.1 Esperimento ATLAS

ATLAS (*A Toroidal LHC ApparatuS*) è uno dei due esperimenti più grandi ad operare su LHC, insieme a CMS; entrambi sono rivelatori general-purpose, cioè sono in grado di ricercare un'ampia gamma di fenomeni fisici, per esempio sono pensati sia per la ricerca del bosone di Higgs, sia per la ricerca della materia oscura. ATLAS condivide gli stessi scopi di ricerca di CMS, ma questi due utilizzano tecnologie diverse per potersi convalidare a vicenda i risultati.

ATLAS è il rivelatore più grande costruito per un collider con 46 metri di lunghezza e 25 metri di diametro [10] rappresentato in Fig.1.2. Il rivelatore è stato costruito in modo da essere simmetrico rispetto a traslazioni del punto di collisione al suo interno. I principali magneti che compongono la macchina sono un sottile strato di superconduttore con forma solenoidale, che circonda la cavità interna, e tre grandi superconduttori di forma toroidale, uno che circonda l'intera struttura e due che ne chiudono l'estremità.

Il magnete interno produce un campo di 2 T per curvare a sufficienza il moto delle particelle cariche nel rivelatore interno e misurarne la quantità di moto, il segno della carica e i vertici, grazie a una combinazione di tre tipi di rivelatori: una a pixel semiconduttori ad alta risoluzione, uno a strisce di semiconduttore e uno è la combinazione di rivelatori a cannucce e rivelatori a transizione di radiazione. Questa struttura è circondata da un calorimetro elettromagnetico ad argon liquido ad alta granularità (LAr) e da

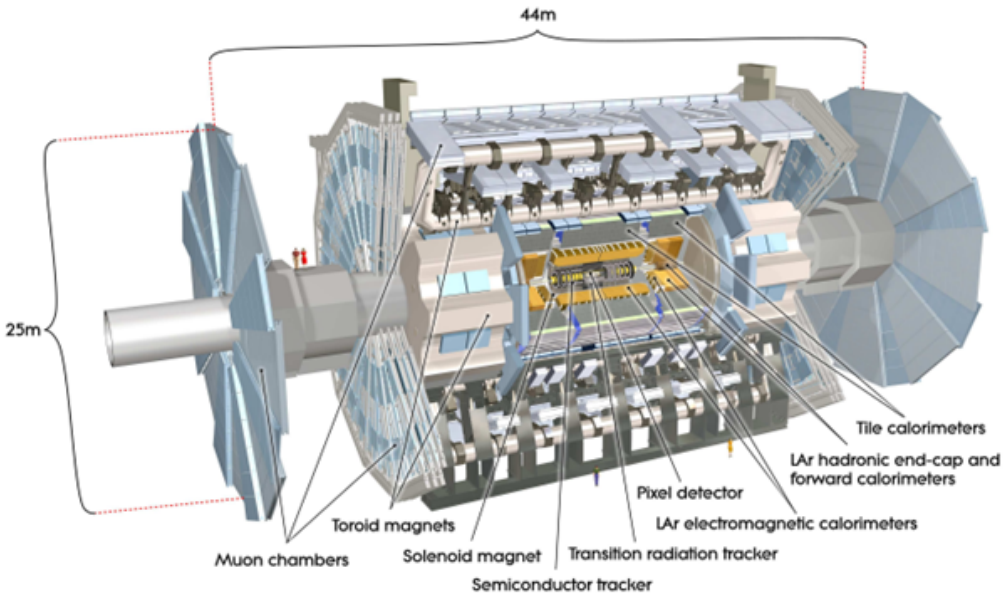

Figura 1.2: *Spaccato del rivelatore ATLAS in cui si può osservare la simmetria cilindrica e i vari strati che formano la macchina.*

un calorimetro adronico a scintillatore, che servono a misurare l'energia delle particelle mediante il loro assorbimento.

I calorimetri riescono a fermare la maggior parte delle particelle eccetto neutrini e muoni: i primi vengono rivelati ricostruendo la quantità di moto mancante mentre i secondi vengono individuati dallo spettrometro muonico che circonda i calorimetri. Lo spettrometro utilizza tre strati di camere di rivelazione per misurare la carica e quantità di moto dei muoni, grazie anche ai tre magneti toroidali che circondano l'intera struttura, questi curvano le traiettorie dei muoni e riducono gli effetti del multiple-scattering.[11]

Trigger

L'apparato di ATLAS può osservare più di un miliardo di interazioni protone-protone per secondo, ma solo una piccola parte contengono informazioni interessanti, inoltre i macchinari sono in grado di trasferire e salvare solo i dati relativi a una piccola parte delle collisioni osservate. Per filtrare gli eventi registrati è stato sviluppato un sistema di trigger suddiviso in tre livelli: Level-1 (L1), Level-2 (L2) e Event Filter [11]. Il primo livello (L1) è un trigger di tipo hardware che utilizza elettronica appositamente progettata e montata direttamente sul rivelatore per ridurre la frequenza degli eventi registrati da 1 GHz iniziale fino a 75-100 kHz. I successivi due livelli vengono chiamati high-level trigger (HLT) e sono di tipo software, questi conducono analisi dettagliate

di ogni collisione, concentrandosi in particolare su certe regioni di interesse (Region-of-Interest) individuate da L1 riducendo così la frequenza degli eventi registrati fino a 400 Hz [11].

I dati relativi alle collisioni, che hanno superato tutti i livelli di trigger, vengono salvati come dati grezzi (RAW) all'interno del sistema di storage del Tier 0 situato al CERN.

1.1.2 Esperimento CMS

CMS (*Compact Muon Solenoid*) è un rivelatore general-purpose che opera su LHC. Questo esperimento è stato progettato sia per lo studio del Modello Standard, per esempio per la ricerca del bosone di Higgs, sia per la ricerca di dimensioni spazio-temporali aggiuntive e per la ricerca di possibili particelle che costituiscono la materia oscura. Questo esperimento opera in parallelo ad ATLAS, ma utilizza soluzioni tecniche differenti e un diverso sistema di magneti così da ottenere un'ulteriore conferma della validità dei risultati sperimentali.

Questo rivelatore, come quello di ATLAS, ha una struttura cilindrica formata strati concentrici: la struttura è alta 15 metri per 25 metri di lunghezza, il che lo rende abbastanza compatto visto i materiali che lo compongono, ed è stato progettato per ottenere una rivelazione molto accurata dei muoni, grazie all'impiego di un magnete solenoidale, il più potente mai costruito [12].

CMS impiega una combinazione di sensori calorimetri e rivelatori di muoni, per ricostruire le tracce delle particelle emesse nelle collisioni e misurarne carica, momento ed energia.

Scoperta del bosone di Higgs

Gli esperimenti di ATLAS e CMS hanno raggiunto un grande obbiettivo nel 2012 con la scoperta del bosone di Higgs. La rivelazione di questa particelle è resa molto complicata per via della sua grande massa, 120 volte la massa di un protone, e della sua vita media molto breve, all'incirca 10^{-22} secondi [13]. Una vita media così breve rende impossibile osservare direttamente il bosone, ma si possono osservare i prodotti del suo decadimento; questi prodotti sono però generati anche da altri processi che avvengono in grande quantità durante le collisioni. Per distinguere i decadimenti del bosone da questo fondo si utilizzano le proprietà dei decadimenti (momento, energia, ...), che sono state misurate dai rivelatori, per calcolare la massa invariante del processo: nel caso in cui il decadimento sia proprio quello di un bosone di Higgs la massa invariante calcolata corrisponde alla massa del bosone, mentre nel caso in cui appartenga al fondo la massa invariante assumerà un valore aleatorio nel range di masse possibili.

Misurando un numero sufficientemente elevato di processi è possibile compiere un'analisi statistica da cui si dovrebbe ottenere un picco in corrispondenza della massa del

bosone di Higgs, perché questo è l'unica massa invariante con un valore definito (entro un certo errore) in un insieme di masse invarianti con valori aleatori. Questo picco è stato appunto osservato nel 2012 sul valore di 125 GeV[14] con una significatività di 4.9 sigma.

Ad eccezione della massa le proprietà del bosone di Higgs erano state previste teoricamente e successivamente verificate sperimentalmente, per esempio si verificò che il bosone avesse spin nullo, infatti i decadimenti da cui era stato scoperto erano quelli in fotoni o in bosoni Z, che limitavano i valori possibili dello spin a zero e due.

I possibili decadimenti del bosone di Higgs furono trovati sperimentalmente nel corso del tempo: i primi decadimenti trovati furono quelli in fotoni, bosoni W e Z, successivamente nel 2016 si trovò il decadimento in una coppia di leptoni tau [15] seguita nel 2018 dall'osservazione del decadimento in quarks b [15]. Il decadimento in una coppia di quarks top non è possibile per via della massa maggiore dei due quarks, ma nel 2018 si osservò il processo inverso, di fusioni di due quarks top in un bosone di Higgs [15].

1.1.3 Esperimento ALICE

ALICE (*A Large Ion Collider Experiment*) è un rivelatore che si concentra sulla fisica degli ioni pesanti, per cui è stato costruito per studiare la materia fortemente interagente in condizioni di densità di energia estreme, dove la materia assume la fase chiamata plasma di quarks e gluoni.

Per parte dell'anno LHC produce collisioni fra ioni di piombo, queste producono temperature così elevate da andare a 'scogliere' i nucleoni e formare il plasma. Lo studio di questo plasma e delle sue proprietà è fondamentale per lo studio della cromodinamica quantistica (QCD) e per capire il fenomeno del confinamento.

1.1.4 Esperimento LHCb

LHCb (*Large Hadron Collider beauty*) è un esperimento sviluppato per studiare la sottile differenza fra materia e antimateria, concentrandosi sull'osservazione del quark beauty o quark b.

A differenza degli altri grandi esperimenti di LHC, LHCb non circonda il punto di collisione con i suoi detector ma si concentra sulla rivelazione di particelle che si muovono in avanti lungo l'asse di collisione, utilizzando a tale scopo una serie di subdetector.

1.2 Worldwide LHC Computing Grid (WLCG)

Il CERN e le collaborazioni che lavorano ai suoi esperimenti (ATLAS, CMS, ALICE e LHCb) devono gestire un enorme mole di dati, per esempio per il RUN3 di LHC le collaborazioni si sono preparate ad archiviare e analizzare più di 600 PB di dati e persino

durante il RUN2 ogni giorno venivano processati in media 1 PB di dati [16]. Tutti questi dati vengono salvati su dei nastri magnetici conservati nel CERN storage system (EOS), costruito appositamente per soddisfare le esigenze degli esperimenti di LHC. I nastri magnetici vengono utilizzati come memoria a lungo termine ma le collaborazioni richiedono mezzi di condivisione di questi dati che siano molto più veloci, per questo motivo si servono del Worldwide LHC Computing Grid (WLCG), un'infrastruttura informatica globale, che conserva, distribuisce e analizza i dati prodotti dagli esperimenti rendendoli disponibili a tutti i membri della collaborazione, in qualsiasi luogo essi si trovino.

Il WLCG è ordinato in quattro livelli di tiers, schematizzati in Fig.1.3, ossia strati, numerati da 0 a 3, e ognuno di questi mette a disposizioni un certo insieme di servizi [17]:

- Il Tier-0 è il CERN Data Centre che si trova a Ginevra, Svizzera. Tutti i dati prodotti da LHC passano da questo hub, che però offre solo il 20% circa della capacità di calcolo dell'intera infrastruttura. Gli scopi del Tier 0 sono mantenere al sicuro la prima copia dei dati (raw data), farne le prime ricostruzioni e distribuire sia i dati raw sia le ricostruzioni ai Tier 1;

- I Tier-1 sono tredici grandi centri di calcolo con una potenza di calcolo e capacità di archiviazione sufficientemente grandi, che mettono a disposizione un supporto permanente alla Grid. Il loro scopo è quello di mantenere al sicuro una parte dei dati raw e i dati ricostruiti, riprocessare su larga scala questi dati, conservare i risultati di questa operazione e infine condividere i dati con i Tier 2, conservando anche una parte dei dati simulati da essi prodotti;

- I Tier-2 sono principalmente università o altri istituti scientifici, con una capacità di calcolo e di archiviazione adatta a specifiche analisi. Si occupano di fare analisi e conservare una parte dei dati inerenti alla produzione e ricostruzione di eventi simulati. Attualmente sono presenti 160 siti intorno al mondo che ospitano un Tier 2;

- I Tier-3 sono i singoli ricercatori che accedono al resto dell'infrastruttura attraverso le macchine di un dipartimento universitario o anche semplicemente usando un singolo PC.

1.2.1 Modello di calcolo di ATLAS

Il modello di calcolo di ATLAS sfrutta a pieno il paradigma della Grid, cercando di decentralizzare e condividere il più possibile le sue risorse di calcolo. Secondo questo modello i dati raw e processati vengono distribuiti ai Tier 1 e Tier 2, per sfruttare a pieno le risorse computazionali messe a disposizione dalla collaborazione. Successivamente vengono impiegate le risorse dei Tier 3 e di altre strutture a cui ATLAS ha accesso per

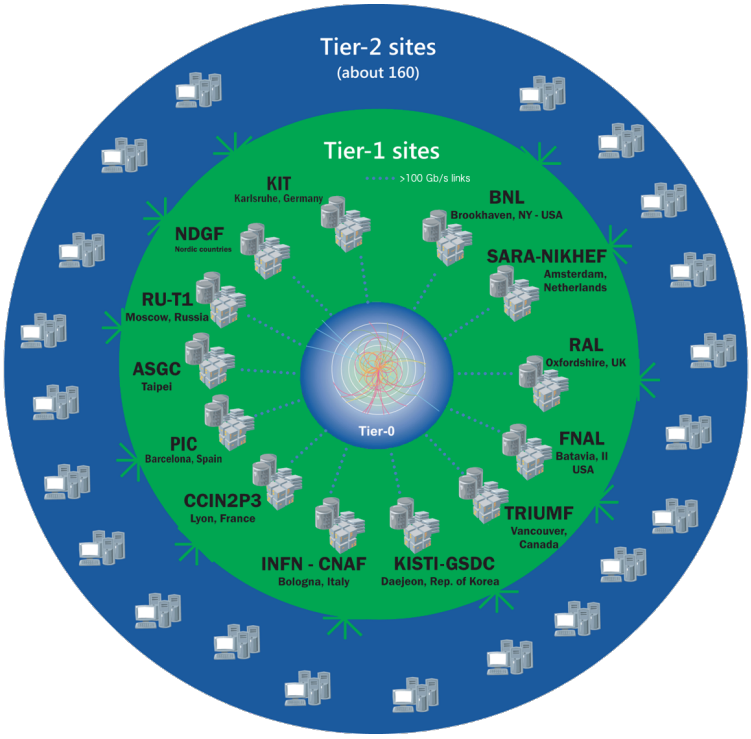

Figura 1.3: *Schema della struttura del WLCG suddiviso in tiers.*

svolgere le analisi. L'accesso ai dati è gestito da ATLAS Virtual Organization e permette agli utenti di eseguire i jobs, che gli danno accesso da remoto ai dati conservati nelle varie infrastrutture della Grid.

Infrastrutture del CNAF

In Italia sono presenti strutture del Tier 1, del Tier 2 e del Tier 3. Il Tier 1 si trova presso il CNAF di Bologna, mentre i quattro Tier 2 sono nelle città di Frascati, Napoli, Milano and Roma [18]. Infine i Tier 3 sono sparsi in varie aree del paese. Il CNAF è il centro nazionale per la ricerca e lo sviluppo nelle tecnologie informatiche e telematiche dell'INFN (Istituto Nazionale di Fisica Nucleare). Questo centro si occupa di fornire risorse e supporto per le attività di storage, distribuzione, processamento e analisi dei dati degli esperimenti di LHC, ma funge da computing facility anche per altri esperimenti.

Per prepararsi all'enorme quantità di dati che verranno prodotti da High-Luminosity LHC, ma anche da altri grandi esperimenti come per esempio Virgo/Ligo, il CNAF ha progettato la costruzione di un nuovo data center attualmente in costruzione al Tecnopolo di Bologna. Il nuovo centro verrà posizionato affianco a Leonardo, il supercomputer del CINECA (centro di supercomputing italiano) nella prospettiva di utilizzare l'enorme

potere di calcolo di Leonardo per l'analisi dei dati provenienti dagli esperimenti di fisica [18].

1.2.2 Servizio StoRM

Il servizio StoRM è un'implementazione di storage resource manager (SRM) sviluppata all'INFN-CNAF per la gestione del suo Tier-1 e di altri 30 siti [18]. In generale un SRM si deve occupare della gestione di diverse storage resources come sistemi di archiviazioni su disco e/o su nastro, allocando dinamicamente spazio per i nuovi file e condividendo i dati già in suo possesso. StoRM si appoggia sul General Parallel File Systems (GPFS [19]) di IBM e utilizza il sistema IBM Tivoli Storage Manager (TSM [20]) per accedere ai dati registrati su nastro. Per il trasferimento dei dati, invece, StoRM si appoggia sui servizi di GridFTP, HTTP e XRootD.

Il servizio StoRM ha un'architettura a multi-layer schematizzata in Fig.1.4, questa si divide fra due componenti principali, il Frontend (FE) and Backend (BE) [21]; le due componenti comunicano fra loro in due modi: con richieste sincrone che mettono in diretta comunicazione FE e BE attraverso un XML-RPC api, oppure con richieste asincrone mediate da un database. Il Frontend fornisce l'interfaccia del SRM con gli utenti autorizzati, fornendo anche il sistema di autenticazione, e gestisce le richieste operando direttamente con il BE per quelle sincrone o salvandole nel database nel caso di asincrone. Il Backend invece contiene la logica per la gestione dello storage e interagisce direttamente con il file system.

Il Frontend e il Backend trascrivono nei rispettivi logfile informazioni riguardanti il funzionamento dei servizi, le richieste degli utenti e le conseguenti operazioni eseguite, per tenere traccia del funzionamento del servizio StoRM. Questi file possono arrivare anche alle dimensioni di qualche GB, per cui un analisi dei files da parte di un operatore risulta lunga e poco efficiente e deve quindi essere sostituita da un'analisi più automatica. I file però non sono scritti con un linguaggio standard ma variano molto da macchina a macchina, motivo per cui è conveniente scegliere come approccio al problema l'utilizzo del Machine Learning, per via della sua forte flessibilità.

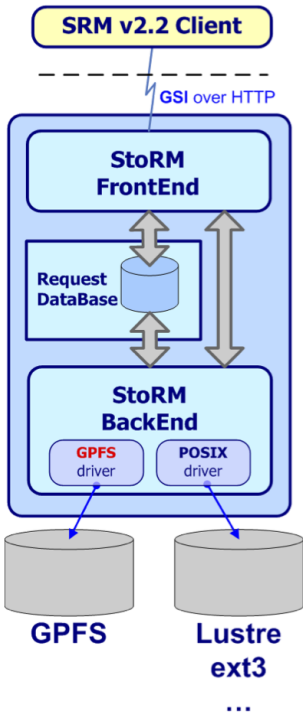

Figura 1.4: *Schema semplificato dell'architettura del servizio StoRM*

Capitolo 2

Machine Learning

L'intelligenza artificiale, comunemente abbreviata come AI (*Artificial Intelligence*) è una branca dell'informatica che si occupa di sviluppare sistemi, sia hardware sia software, in grado di emulare caratteristiche tipicamente umane, quali le percezioni spazio-temporale o decisionali. Si cerca, quindi, di dotare le macchine non solo di una capacità di calcolo ma anche di differenti forme d'intelligenza, come l'intelligenza spaziale o quella sociale.

L'intelligenza artificiale nasce per convenzione nel 1956, anno in cui si tenne un importante convegno al Dartmouth College ad Hanover, New Hampshire, organizzato dal matematico McCarthy, che attribuì a questa branca dell'informatica il nome di intelligenza artificiale [22]. Il convegno riuniva scienziati che provenivano da ambiti molto diversi, dalla psicologia all'informatica, ma tutti interessati allo sviluppo di macchine 'pensanti'. A seguito di questa conferenza ci fu un grande lavoro di ricerca teorica e sperimentale che coinvolse sia università sia aziende, in particolare IBM. L'intelligenza artificiale estese le sue applicazioni dalla sola matematica e informatica anche alla biologia, per lo studio delle molecole, e alla psicologia; oggi le sue applicazioni sono molto vaste: vanno dall'apprendimento autonomo utilizzato quotidianamente nei sistemi di riconoscimento vocale e nei sistemi di sicurezza, fino allo sviluppo di veicoli a guida autonoma, ma anche nella programmazione di giochi, come per esempio gli scacchi. Le sue applicazioni spaziano anche in ambito finanziario, robotico e medico, dove si utilizza l'intelligenza artificiale per le analisi del battito cardiaco e le diagnosi di alcune forme tumorali.

Di particolare interesse per questo lavoro sono le applicazioni nell'ambito della fisica delle particelle, come ad esempio negli esperimenti di LHC, dove l'intelligenza artificiale e in particolare il machine learning vengono utilizzati per diversi scopi [23]. Il machine learning è utilizzato dai sistemi di trigger per filtrare le ingenti quantità di dati prodotti, identificando i dati da mantenere e quelli da scartare. Viene anche utilizzato dagli apparati di misura per migliorare la risoluzione degli strumenti, come ad esempio i calorimetri [23], oppure può essere utilizzato come supporto alle risorse di calcolo, ad esempio CMS utilizza tecniche di machine learning per monitorare il trasferimento dei dati e riconoscere in anticipo nodi con possibili congestioni [23].

2.1 Machine learning

Una delle branche più importanti dell'intelligenza artificiale è il machine learning (apprendimento automatico) che impiega algoritmi specifici per far migliorare le capacità della macchina di agire e prendere decisioni attraverso l'esperienza. Lo sviluppo di questi algoritmi è fondamentale per operare in ambiti in cui non è possibile conoscere a priori tutti i possibili sviluppi del sistema, quindi la macchina si trova anche a compiere azioni che non erano state programmate.

L'idea alla base del machine learning, ossia creare una macchina in grado di apprendere, era già condivisa da diversi scienziati che si erano approcciati all'intelligenza artificiale come per esempio A. Turing [24], ma il termine machine learning fu coniato solo successivamente da A. Samuel nel 1959 [25] per descrivere il metodo con cui una macchina aveva imparato il gioco della dama. Samuel distingueva due tipi di machine learning, l'apprendimento supervisionato e quello per rinforzo: questi vengono descritti attraverso reti neurali, che nel primo caso sono altamente organizzate e predisposte per imparare un'attività specifica, mentre nel secondo caso sono reti con collegamenti casuali che evolvono durante l'apprendimento attraverso un meccanismo di ricompensa e punizione. A seconda dell'algoritmo utilizzato per implementare l'apprendimento della macchina, si possono distinguere tre modelli, l'apprendimento supervisionato, quello non supervisionato e quello per rinforzo:

- l'apprendimento supervisionato richiede di fornire al computer una serie di nozioni iniziali già codificate, come modelli ed esempi da cui estrarre un database di informazioni ed 'esperienze'. Quando il computer si trova davanti a un problema formula una risposta in base all'analisi delle informazioni che le sono state fornite in precedenza, questo però significa che l'algoritmo è pensato per risolvere solo un ristretto numero di problemi. L'apprendimento supervisionato viene utilizzato in diversi settori come quello medico o quello del riconoscimento vocale;

- l'apprendimento non supervisionato richiede di fornire alla macchina delle informazioni iniziali non codificate, in questo modo può estrarre informazioni senza avere nessun esempio del modo in cui farlo o dei risultati che devono essere ottenuti. Sarà la macchina stessa a organizzare e comprendere le informazioni, per poi trovare il risultato a cui portano. Questo tipo di apprendimento permette di lasciare più libertà alla macchina ed è applicabile a problemi più generici;

- l'apprendimento per rinforzo è il sistema più complicato fra i tre, perché richiede oltre alla capacità di calcolo della macchina anche degli strumenti aggiuntivi per analizzare l'ambiente circostante. Se si prende l'esempio delle macchine a guida autonoma questi strumenti possono essere telecamere, GPS o sensori, che il computer usa per prendere la decisione che si adatta meglio all'ambiente in cui si trova.

2.1.1 Clustering

Tra gli algoritmi di machine learning non supervisionato, il clustering assume una notevole rilevanza per la sua capacità di trovare un eventuale pattern all'interno dei dati presi in esame. Il clustering prevede di passare all'algoritmo molti dati in input e lasciare che questo li divida in gruppi, *cluster*, formati da punti 'simili' fra loro. Questo metodo è utile quando si devono analizzare dati di cui non si ha una conoscenza sufficiente, per esempio per ricercare al loro interno dei patterns o estrarne delle features.

Sono presenti diversi tipi di algoritmi di clustering, ognuno adatto a situazioni differenti, ma quello d'interesse per questa tesi ed anche il più diffuso è il tipo *centroid-based*, per cui i punti, rappresentanti i dati vengono divisi nei diversi centroidi in base alla distanza quadratica dai centri dei centroidi. Per questo lavoro è stato appunto utilizzato un algoritmo *centroid-based* chiamato K-means.

Algoritmo K-means

L'algoritmo K-means è il più usato per il clustering per via della sua semplicità. Questo algoritmo è una variante dell'algoritmo di aspettativa-massimizzazione (EM), ossia cerca di raggruppare i dati, dividendoli in gruppi con uguale varianza e per fare ciò cerca di minimizzare l'inerzia del sistema ossia la somma dei quadrati delle distanze dei punti dal centro del loro *cluster*.

L'algoritmo parte da un campione di n punti e deve dividerli in k *clusters* differenti, dove ogni *cluster* è individuato dalla posizione del suo centro, anche detto centroide, μ_j che è la media dei punti appartenenti a quel *cluster*. L'algoritmo assegna i punti al *cluster* più vicino e modifica i centroidi fino a minimizzare l'inerzia del sistema

$$\sum_{i=0}^{n} \min_{\mu_j \in C} \left(||x_i - \mu_j||^2 \right)$$

che può anche essere pensata come una misura della coerenza nella scelta dei *cluster*.

L'algoritmo K-means viene normalmente risolto in forma approssimata utilizzando l'algoritmo di Lloyd che può essere suddiviso in tre passaggi: per il primo passaggio si scelgono i valori iniziali per i centroidi e il modo più facile di farlo è scegliere k punti dal dataset; per il secondo passaggio si aggiorna la posizione dei centroidi calcolando la media dei punti appartenenti ai vari *cluster*; per ultimo si calcolala la distanza fra i vecchi cenroidi e i nuovi e si ripetono gli ultimi due passaggi fino a che le variazioni sui centroidi non diventano trascurabili. Con abbastanza tempo questo algoritmo converge sempre a un minimo, ma questo minimo potrebbe essere locale e non globale, quindi per ovviare a questo problema l'algoritmo viene processato più volte con diversi valori di inizializzazione dei centroidi. Per ottenere una precisione maggiore i valori di inizializzazione vengono ottenuti utilizzando l'algoritmo K-means++ che inizializza i centroidi in

modo che siano distanti fra loro e al termine di tutte le iterazioni viene scelto il risultato migliore.

Il problema di questo algoritmo è che parte dall'assunzione che i *clusters* siano isotropi e convessi, ma questo in generale non è vero, quindi l'algoritmo funziona male per *cluster* allungati o con forme particolari. Un altro problema di questo algoritmo è che necessita di avere il numero di *cluster* k come parametro di input, ma si possono utilizzare diversi metodi per trovare il numero ottimale di *cluster*. Per questo lavoro sono stati utilizzati tre metodi:

- l'elbow method è molto comune e consiste nel calcolare l'inerzia del dataset per vari valori di k, questi vengono poi rappresentati su un grafico prendono la forma di una curva a gomito e il valore ottimale di *clusters* corrisponde al 'gomito' di questa curva ossia il punto in cui si osserva un forte cambio di pendenza;

- il coefficiente di silhouette è una misura di quanto ogni *cluster* sia definito quindi al numero ottimale di *clusters* corrisponde il coefficiente massimo. Per ottenere questo coefficiente si deve calcolare a la distanza media fra i punti del dataset e b la distanza media fra un punto e tutti gli altri punti del *cluster* più vicino dopo quello a cui appartiene, che combinati danno il coefficiente

$$s = \frac{b - a}{\max(a, b)};$$

- l'indice di Davies-Bouldin è una misura della somiglianza fra i *cluster* per cui il numero ottimale di *clusters* corrisponde a un minimo di questo indice. La somiglianza fra due *cluster* R_{ij} è calcolata usando la distanza fra due *cluster* M_{ij} e la dispersione dei singoli *clusters* S_i da cui

$$R_{ij} = \frac{S_i + S_j}{M_{ij}}$$

  dove per ogni *cluster* si prende il valore massimo $R_i = \max(R_{ij})$ e infine si ottiene l'indice semplicemente come la media delle misure di somiglianza.

Capitolo 3

Descrizione del dataset e dei programmi utilizzati

Per questo lavoro di tesi sono stati analizzati i logfiles prodotti dal Frontend del sistema StoRM del CNAF di Bologna e sono riferiti alla settimana che va dal 07/03/2020 al 13/03/2020. I file sono stati analizzati utilizzando programmi scritti in Python (versione 3) e sviluppati sfruttando la piattaforma Jupyter Noteboook [26]. L'analisi può essere divisa in due fasi: una prima fase in cui i file sono stati preprocessati utilizzando la libreria Pandas [27] in modo da ottenere un database e una seconda fase in cui sono stati effettivamente analizzati utilizzando l'algoritmo K-means della libreria Scikit-learn [28].

Pandas

Pandas è una libreria Python sviluppata per la creazione, manipolazione e analisi di dataset con lo scopo di diventare lo strumento open-source più performante e flessibile in questo campo. La libreria è stata sviluppata a partire dal 2008 da un impiegato presso AQR Capital Management per poi diventare open-source nel 2009, in seguito è stata migliorata anche grazie al supporto di una community di contributori da tutto il mondo. Dal 2015 Pandas è anche stata sponsorizzata da NumFOCUS [29] per garantire il suo sviluppo open-source.

Pandas fornisce due strutture per l'organizzazione dei dati: Series e DataFrame. Quest'ultima è più utilizzata per via della sua maggiore complessità, infatti oltre alla sua rappresentazione in forma di tabella, con indici integrati, permette anche un efficiente e veloce accesso e manipolazione dei dati, il che lo rende lo strumento migliore per lavorare con grandi dataset. Infine Pandas contiene funzioni per la lettura e scrittura di file in diversi formati come ad esempio JSON, CSV o Microsoft EXCEL.

Scikit-learn

Scikit-learn è una libreria Python di apprendimento automatico; contiene diversi algoritmi di machine learning come algoritmi di classificazione, di regressione o di clustering, fra cui anche K-means. Lo sviluppo di questa libreria partì nel 2007 come progetto del Google Summer of Code, per poi essere migliorata e resa open-source nel 2010; da allora la libreria è stata aggiornata periodicamente grazie ai contributi di una community internazionale. Lo scopo di questa libreria è rendere accessibili gli algoritmi di machine learning a chi non è specializzato in questo campo.

Di questa libreria è stato utilizzato l'algoritmo di clustering K-means [30] per l'analisi del dataset e le funzioni del modulo sklearn.metrics [31] per valutare il numero di cluster ottimale per ogni file.

3.1 Preprocessamento

I files di log sono organizzati in sequenze di righe di testo scritte in un formato non standard e questo formato è differente anche fra file di Frontend e di Backend. Un esempio di alcune righe di un file di Frontend è riportato in seguito

```
03/07 03:49:02.040 Thread 13 -  INFO [2b8f0da1-e1ee-4f7b-85fd-b76984c851ad]:
 process_request : Connection from 2001:6b0:17:180::2:2
03/07 03:49:02.075 Thread 43 -  INFO [1a893ed2-2f31-48a7-b46e-cc71060bc210]:
 Request 'BOL status' from Client IP='2001:6b0:17:180::2:2'
 Client DN='/DC=ch/DC=cern/OU=Organic Units/OU=Users/CN=atlact1/CN=555105/CN=Robot:
 ATLAS aCT 1' # Requested token '324d95b6-c26f-46ab-b4d3-20a13c532e0f'
03/07 03:49:02.077 Thread 43 -  INFO [1a893ed2-2f31-48a7-b46e-cc71060bc210]:
 Result for request 'BOL status' is 'SRM_REQUEST_INPROGRESS'
```

dove si può vedere che per ogni riga lo StoRM-Frontend riporta data e ora in cui è stato scritto il messaggio, il numero di thread, l'identificativo (ID) univoco riportato fra parentesi quadre dopo l'indicatore INFO e il messaggio stesso. La differenza fra l'ID e il numero di thread sta nel fatto che l'ID non verrà mai utilizzato per due processi diversi, mentre uno stesso thread viene riutilizzato più volte nell'arco della stessa giornata.

I dati rappresentati in questo formato non sono adatti all'analisi ma richiedono un preprocessamento che li riorganizzi e produca dei file in formato CSV più facili da manipolare e analizzare. Innanzitutto le righe di testo devono essere suddivise in più colonne come nell'esempio riportato

date	time	PID	level	id	message
03/07	03:49:02.040	Thread 13	INFO	2b8f0da1-e1ee-4f7b-85fd-b76984c851ad	process_request ...
03/07	03:49:02.075	Thread 43	INFO	1a893ed2-2f31-48a7-b46e-cc71060bc210	Request 'BOL ...
03/07	03:49:02.077	Thread 43	INFO	1a893ed2-2f31-48a7-b46e-cc71060bc210	Result for ...

In seguito si può notare che uno stesso processo si articola in più messaggi, tutti identificati dallo stesso ID e lo stesso thread, quindi conviene raggrupparli in un'unica

riga del file indicizzata dall'ID e mantenere solo le informazioni temporali sul primo e ultimo messaggio. Questa operazione è stata compiuta utilizzando la libreria Pandas e ha prodotto un file come nell'esempio riportato

id	PID	Level	date_time_start	date_time_end	message
2b8f0da1-e1ee-4f7b-85fd-b76984c851ad	Thread 13	INFO	03/07 03:49:02.040	03/07 03:49:02.395	process_request ...
1a893ed2-2f31-48a7-b46e-cc71060bc210	Thread 43	INFO	03/07 03:49:01.737	03/07 03:49:02.077	process_request ...
1e64f1d9-e334-42a8-ad4a-6ff4ea6bda53	Thread 36	INFO	03/06 16:53:08.592	03/06 16:53:08.931	process_request ...

Il passaggio successivo al raggruppamento è il masking perché i messaggi contengono dati sensibili, come indirizzi IP degli utenti che inviano richieste, indirizzi URL o Token, tutte informazioni che potrebbero essere utili ai fini di un'analisi più completa ma che aumentano la complessità del problema, con ripercussioni in particolare sui tempi di esecuzione. Si è deciso allora in fase di preprocessamento di nascondere queste informazioni utilizzando le regular expression pre-compilate di Python. Per esempio, un token `1c3ad70f-9e1b-4903-98ef-1221255f907a` può essere riconosciuto dall'espressione `(\\w{8}-\\w{4}-\\w{4}-\\w{4}-\\w{12})` e sostituito dall'indicatore `<TOKEN>` così da ottenere messaggi come

```
Requested '1' SURL(s): '<URL>' Result for request 'PTP' is
 'SRM_REQUEST_QUEUED'. # Produced request token: '<TOKEN>' Request
 'PTP status' from Client IP='<IP>' Client DN=...
```

Infine si è deciso di eseguire due analisi separate sui processi contenenti errori e su quelli privi, quindi si sono separati i file di ogni giorno in due file diversi sempre utilizzando le regular expression di Python, uno che conteneva tutti i processi in cui comparivano le parole 'error' o 'failure' e uno con i restanti processi.

3.2 Applicazione dell'algoritmo

Dopo il preprocessamento i dati possono essere elaborati utilizzando l'algoritmo di clustering non supervisionato K-means. Per questa parte del lavoro è stato utilizzato il codice sviluppato dal Dott. Luca Clissa e messo a disposizione su GitHub [32]. Questo codice contiene, oltre all'algoritmo, anche le due operazioni preliminari a cui devono essere sottoposti i file prima di poter eseguire l'algoritmo: queste due operazioni sono l'estrazione di features e la riduzione di dimensioni del vettore delle feature, entrambi in grado di influire sul risultato di K-means.

3.2.1 Estrazione di feature

In generale per poter eseguire un algoritmo di machine learning è necessario trasformare i dati testuali in vettori numerici, che quantificano dei features del testo ed è importante

che questi vettori contengano le stesse informazioni del testo inziale. Per estrarre feature da un documento di testo si può identificare ogni parola che compare nel documento con una feature e quantificarla attraverso le sue occorrenze. Nel caso in esame per ogni processo si vanno a contare le occorrenze delle parole nei messaggi e successivamente si dividono per il numero totale di parole nello stesso messaggio, così da ottenere un risultato indipendente dalla lunghezza del messaggio: queste feature prendono il nome di Term Frequencies (TF). Un altro passaggio opzionale è l'estrazione per ogni feature di Inverse Document Frequencies (IDF), che in questo caso si rivela utile perché permette di dare meno importanza alle parole che compaiono più frequentemente nel documento e darne di più a quelle più rare. L'operazione complessiva prende il nome di TF-IDF (Term Frequency times Inverse Document Frequency).

Non è necessario compiere queste operazioni manualmente ma si può fare uso della funzione TfidVectorizer [33] della libreria Python Scikit-learn, come nel codice riportato

```
# Extract TF-IDF information
print("Extracting features from the training dataset")
vectorizer = TfidfVectorizer(max_df, min_df,
                             stop_words='english', use_idf=True)
X = vectorizer.fit_transform(logs.message)

print(vectorizer.stop_words_)
print("n_samples: %d, n_features: %d" % X.shape)
print()
```

La funzione TfidVectorizer possiede tre parametri interessati: innanzitutto i parametri `max_df` e `min_df` permettono di selezionare i limiti della document frequency, per poter escludere parole molto frequenti o molto rare. In questo caso si è scelto di non imporre dei limiti sulle frequenze ma lasciare che sia l'algoritmo ad assegnare il giusto peso alle feature, anche se questo può comportare un maggior tempo di esecuzione. Il terzo parametro interessante è `stop_words` impostato su 'english' in modo da escludere a priori parole prive di informazioni, come congiunzioni e articoli (per esempio 'and', 'the', etc.); il codice riportato sopra stampa in standard output la lista di parole esclude.

3.2.2 Ridimensionamento delle features

Il ridimensionamento del vettore delle features serve in risposta al problema di ottenere vettori di dimensioni diverse da documenti diversi, per questo motivo è necessario ridurre queste dimensioni ad un'unica. A tale scopo è stata utilizzata la tecnica LSA (*Latent Semantic Analysis*), che utilizza *singular value decomposition* (SVD) per ridurre lo spazio delle feature a uno spazio a k dimensioni, costituito dalle k feature principali. La tecnica LSA è implementata nella libreria Scikit-learn dalla funzione TruncatedSVD [34], questa

però non normalizza automaticamente i vettori per cui si deve aggiungere la funzione Normalizer per farlo. Questo processo è implementato nel codice seguente con $k = 23$

```
# Apply LSA for dimensionality reduction to get
# a lower-dimensional embedding space.
# Vectorizer results are normalized
svd = TruncatedSVD(23)
normalizer = Normalizer(copy=False)
lsa = make_pipeline(svd, normalizer)

X = lsa.fit_transform(X)
```

3.2.3 Esecuzione dell'algoritmo K-mean

L'ultimo passaggio è applicare l'algoritmo, questo richiede di impostare diversi parametri in particolare il numero di cluster, che non è noto a priori ma può essere ottenuto utilizzando le tecniche esposte nella sezione 2.1.1. L'implementazione dell'algoritmo è riportata nel codice seguente

```
# run K-Means algorithm
 km = KMeans(n_clusters=n_clusters,
             init='k-means++',
             max_iter=500,
             n_init=100,
             verbose=1
            )

print("Clustering sparse data with %s" % km)
km.fit(X)
```

Il parametro `n_cluster` indica il numero di cluster e verrà spiegato in seguito il metodo utilizzato per determinarlo. Il parametro `init` indica come inizializzare i centroidi di partenza, in questo caso è stato selezionato l'algoritmo k-mean++, presentato nella sezione 2.1.1, che sceglie i centroidi iniziali in modo che siano distanti fra loro e non casuali. I parametri `max_iter` e `n_init` servono a ottenere una buona convergenza, imponendo però un limite sui tempi di esecuzione: `max_iter` indica il numero massimo di iterazioni per esecuzione, mentre `n_init` indica il numero di volte in cui viene eseguito l'algoritmo con diversi centroidi iniziali. Al termine l'algoritmo sceglie il risultato migliore fra le varie esecuzioni. La funzione fit applica l'algoritmo al dataset e restituisce una struttura contenete le informazioni sull'esito del processo fra cui un vettore con cui si associa ogni processo al cluster a cui appartiene attraverso un numero che identifica il cluster.

3.2.4 Ricerca del numero ottimale di cluster

Per trovare il numero ottimale di cluster utilizzando le tecniche elencate nel capitolo precedente si deve eseguire più volte l'algoritmo, con valori differenti del parametro `n_cluster`, e per ogni esecuzione estrarre l'inerzia dei cluster, il coefficiente di silhouette e l'indice di Davies-Bouldin. Questi valori possono essere ricavati facilmente perché l'inerzia viene calcolata automaticamente dall'algoritmo K-means e si ottenere dall'attributo inertia_ mentre i due coefficienti possono essere calcolati con le funzioni silhouette_score e davies_bouldin_score del modulo sklearn.metrics della libreria Scikit-learn. L'implementazione di questo passaggio è riportata nel codice seguente

```
# Inertia
Sum_of_squared_distances.append(km.inertia_)
# silhouette e davies-bouldin score
silhouette_avg.append(silhouette_score(X, cluster_labels,
                                       metric='euclidean'))
davies_bouldin_avg.append(davies_bouldin_score(X, cluster_labels))
```

Capitolo 4

Risultati

In questo capitolo vengono presentati i risultati dell'analisi effettuata sui logfiles prodotti dal Frontend del sistema StoRM nella settimana che va dal 07/03/2020 al 13/03/2020. L'analisi ha lo scopo di dimostrare l'efficacia e l'utilità degli algoritmi utilizzati, in particolare dell'algoritmo K-means, nell'organizzazione dei messaggi contenuti nei file in modo da consentire di individuare eventuali anomalie.

Per ogni file, diviso fra messaggi normali e messaggi d'errore, è stata fatta la ricerca del numero ottimale di cluster con i tre metodi discussi nel secondo capitolo e successivamente sono stati utilizzati i risultati ottenuti dalla ricerca per il clustering dei dati. In questa analisi sono stati usati sia i risultati della ricerca del numero ottimale di cluster, sia i risultati del clustering stesso per valutare il corretto funzionamento del sistema StoRM.

Una prima analisi dei dati può venire dal confronto giorno per giorno del numero di processi contenenti messaggi normali e contenenti messaggi d'errore. Nell'istogramma riportato in Fig. 4.1 si può notare che il giorno 11 il numero di messaggi normali è calato bruscamente, mentre nei giorni successivi è ricresciuto gradualmente. Se invece si osserva la percentuale dei processi, contenenti rispettivamente errori e messaggi normali, rispetto al numero totale di processi della giornata (Fig. 4.2), si vede che dal giorno 9 al giorno 12 la percentuale di messaggi d'errore è aumentata.

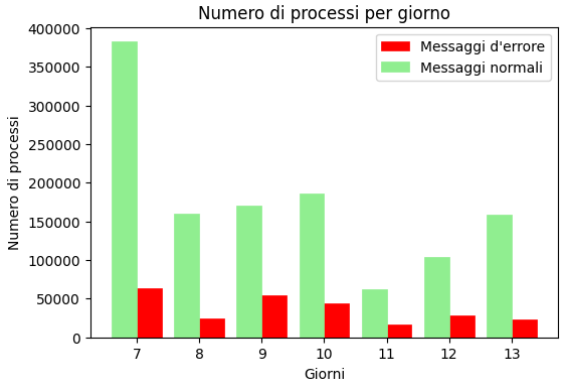

Figura 4.1: *Confronto fra il numero di processi per giorno, distinguendo fra i processi che contengono messaggi normali (in verde) e quelli che contengono messaggi d'errore (in rosso).*

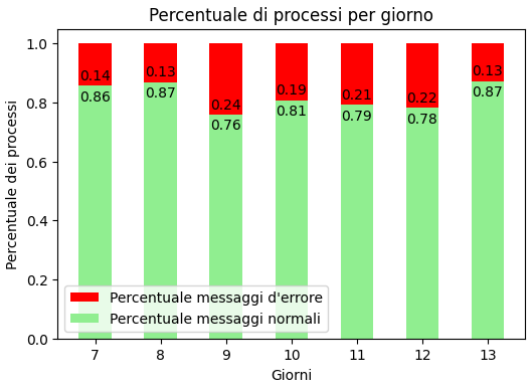

Figura 4.2: *Confronto giorno per giorno sulla percentuale di processi che contengono messaggi normali e di processi che contengono messaggi d'errore rispetto al numero totale di processi della giornata.*

4.1 Risultati della ricerca del numero ottimale di clusters

Per ogni giorno è stato determinato il numero ottimale di cluster, facendo un confronto fra le tre tecniche riportate: il metodo del gomito, il coefficiente di silhouette e l'indice di Davies-Bouldin. I risultati prodotti sono riportati nei grafici seguenti, realizzati mediante la libreria Python Matplotlib. La ricerca del numero ottimale di cluster è eseguita separatamente sui messaggi d'errore (Fig. 4.3-4.9) e su quelli normali (Fig. 4.10-4.16): per i messaggi d'errore la ricerca è stata fatta per un numero di cluster compreso fra 2 e 19, mentre per i messaggi normali fra 2 e 29; i tre metodi spesso non producono lo stesso risultato quindi si è scelto di utilizzare come numero ottimale quello prodotto dal coefficiente di silhouette.

Il metodo del gomito è stato escluso perché non restituisce un valore preciso per il numero ottimale di cluster, ma rimane comunque utile a identificare il range entro cui si può trovare. Nel confronto fra gli altri due metodi si è scelto il coefficiente di silhouette perché rappresenta una misura di quanto sia corretta l'assegnazione dei punti ai rispettivi cluster, invece l'indice di Davies-Bouldin è una misura di quanto i cluster siano distanti quindi si basa più sulle caratteristiche geometriche del data set.

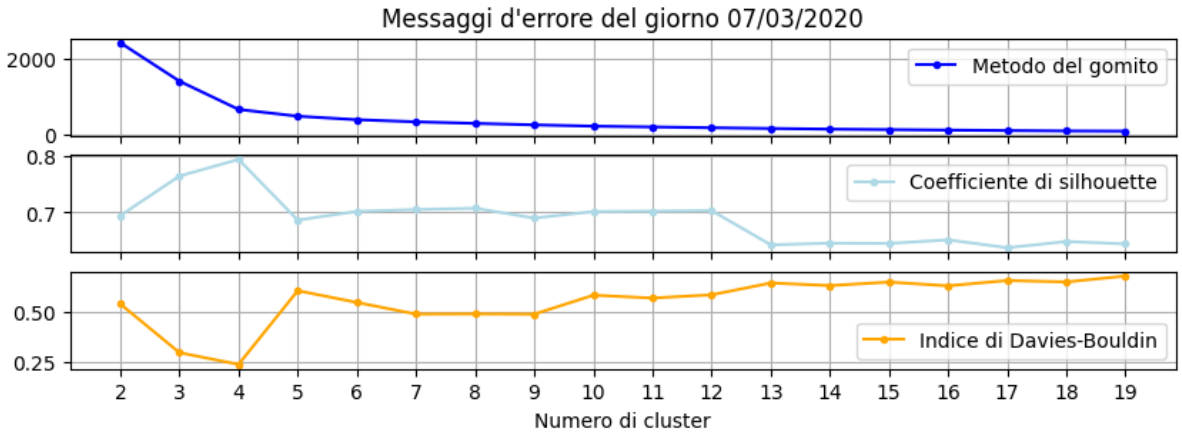

Figura 4.3: *Ricerca del numero ottimale di cluster per i messaggi d'errore del giorno 07/03/2020*

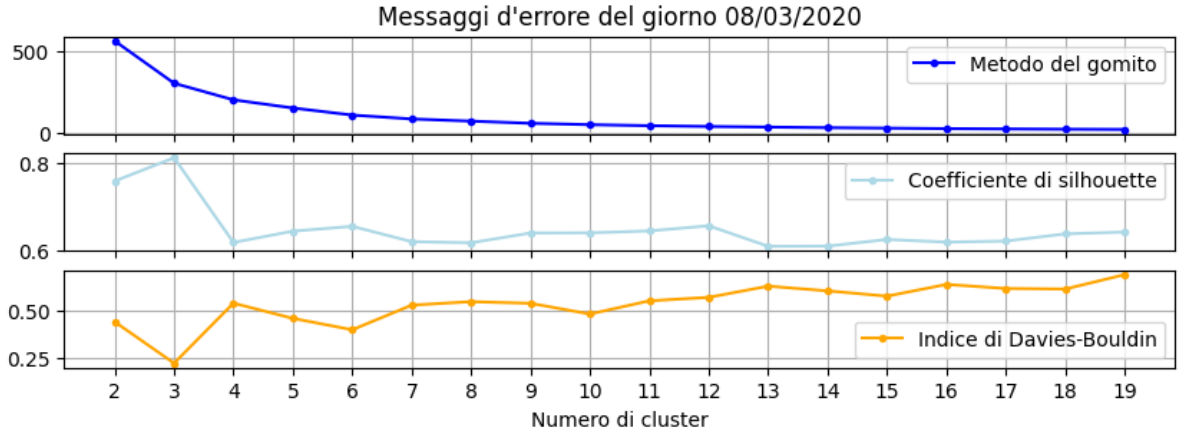

Figura 4.4: *Ricerca del numero ottimale di cluster per i messaggi d'errore del giorno 08/03/2020*

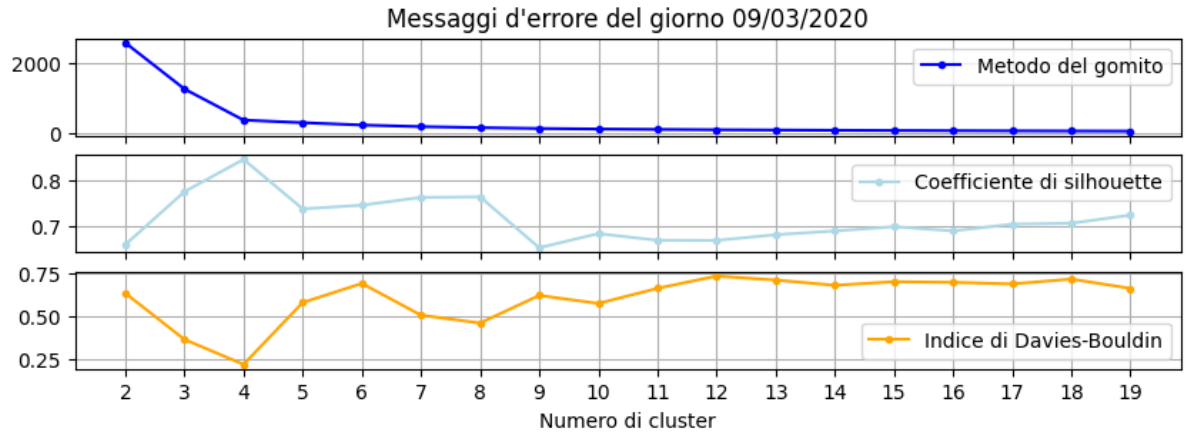

Figura 4.5: *Ricerca del numero ottimale di cluster per i messaggi d'errore del giorno 09/03/2020*

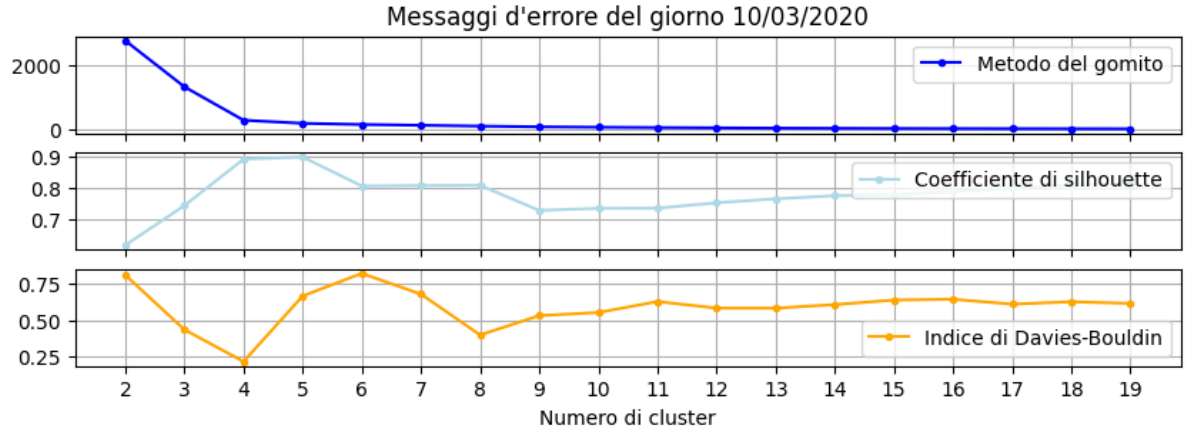

Figura 4.6: *Ricerca del numero ottimale di cluster per i messaggi d'errore del giorno 10/03/2020*

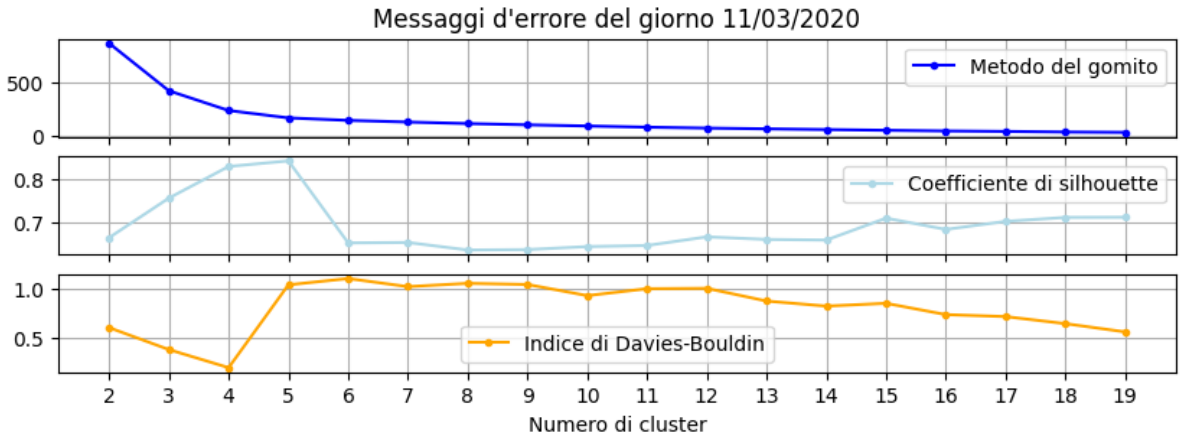

Figura 4.7: *Ricerca del numero ottimale di cluster per i messaggi d'errore del giorno 11/03/2020*

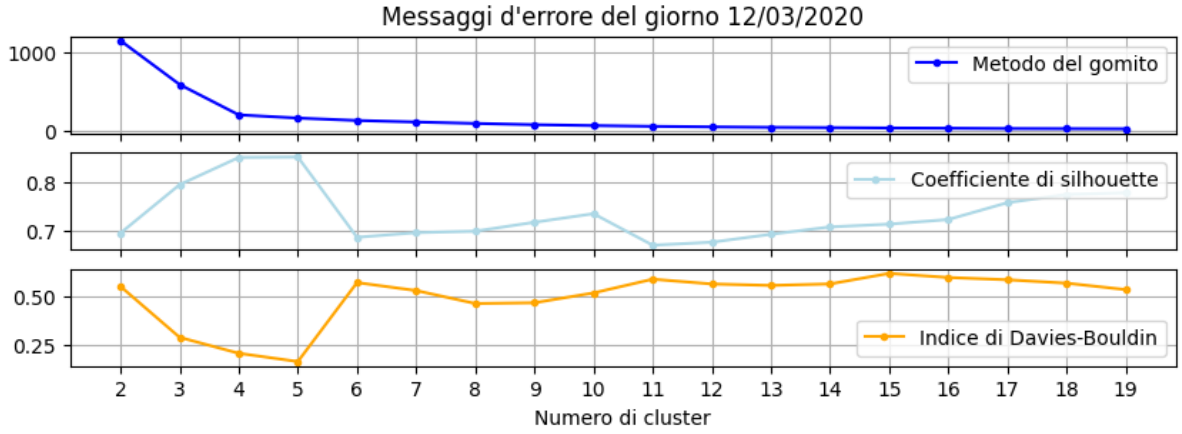

Figura 4.8: *Ricerca del numero ottimale di cluster per i messaggi d'errore del giorno 12/03/2020*

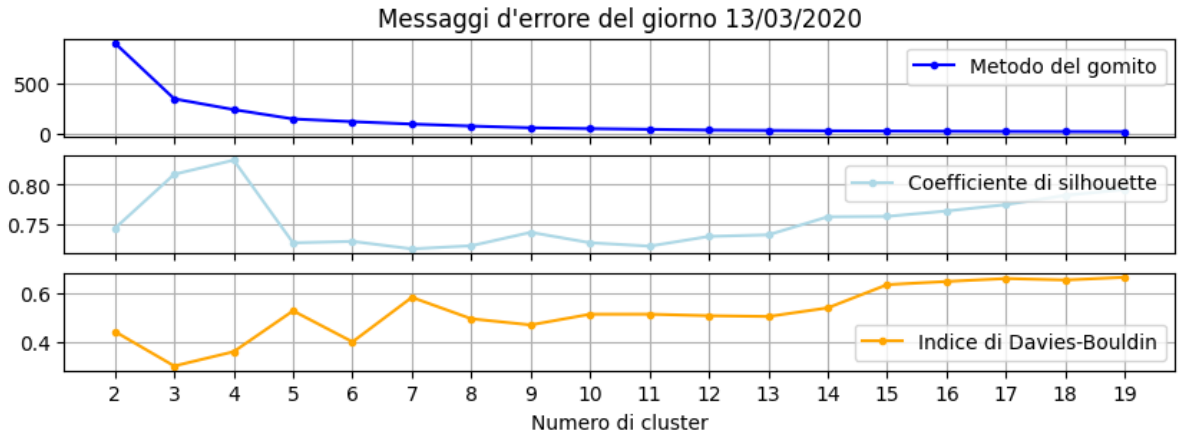

Figura 4.9: *Ricerca del numero ottimale di cluster per i messaggi d'errore del giorno 13/03/2020*

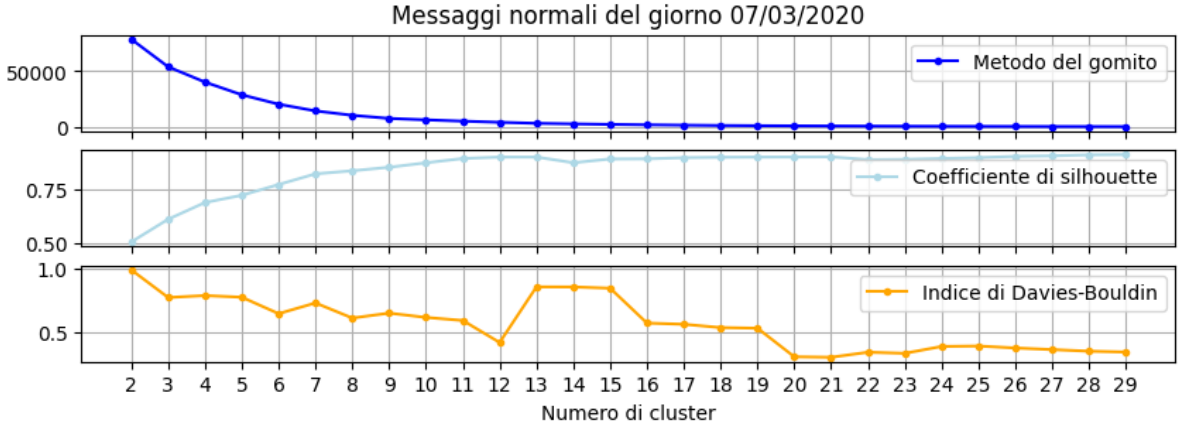

Figura 4.10: *Ricerca del numero ottimale di cluster per i messaggi normali del giorno 07/03/2020*

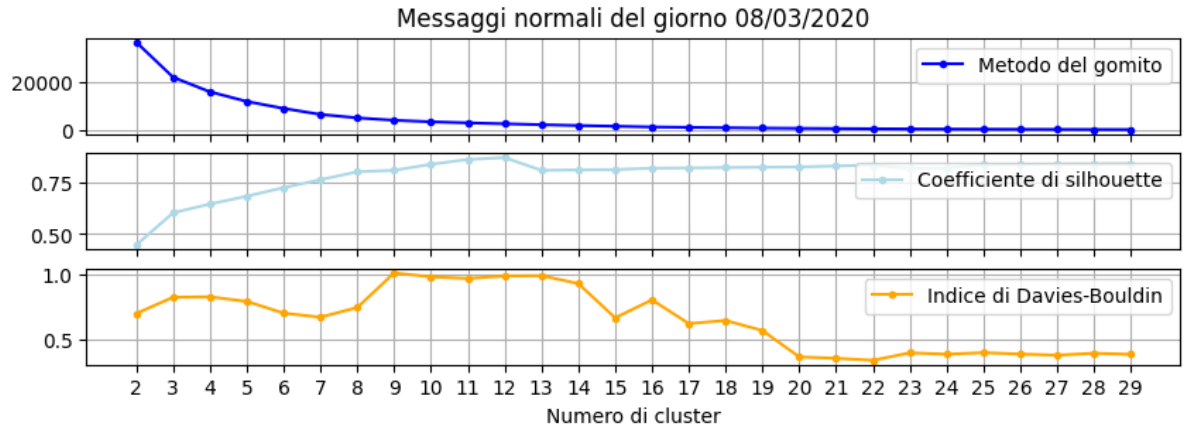

Figura 4.11: *Ricerca del numero ottimale di cluster per i messaggi normali del giorno 08/03/2020*

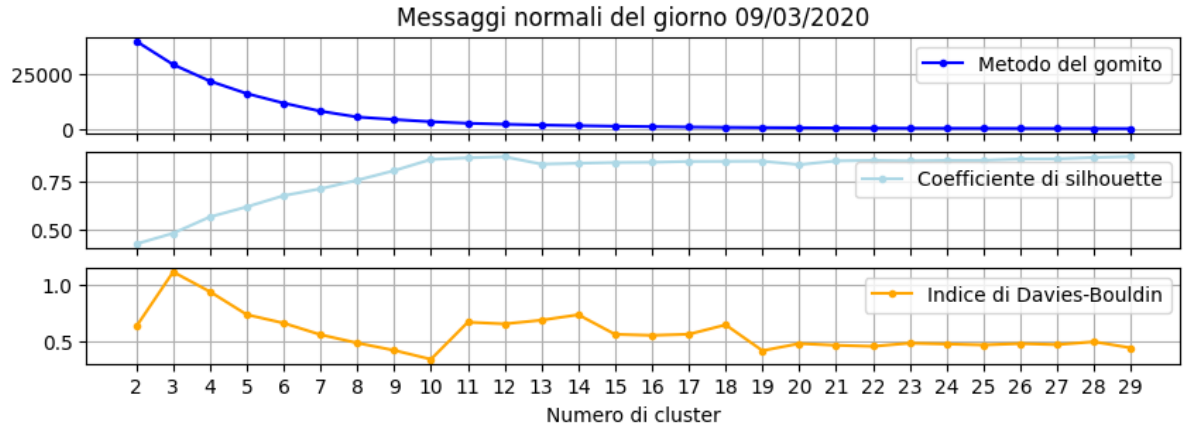

Figura 4.12: *Ricerca del numero ottimale di cluster per i messaggi normali del giorno 09/03/2020*

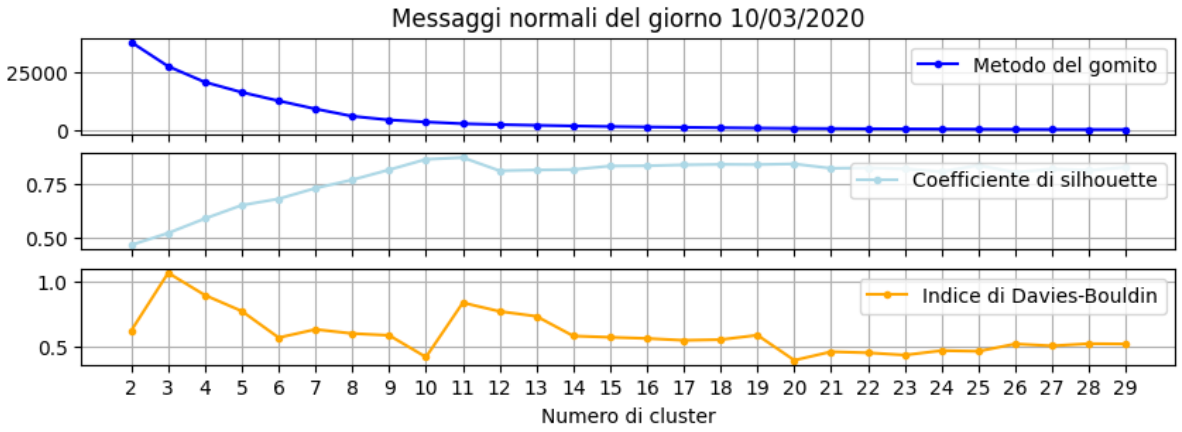

Figura 4.13: *Ricerca del numero ottimale di cluster per i messaggi normali del giorno 10/03/2020*

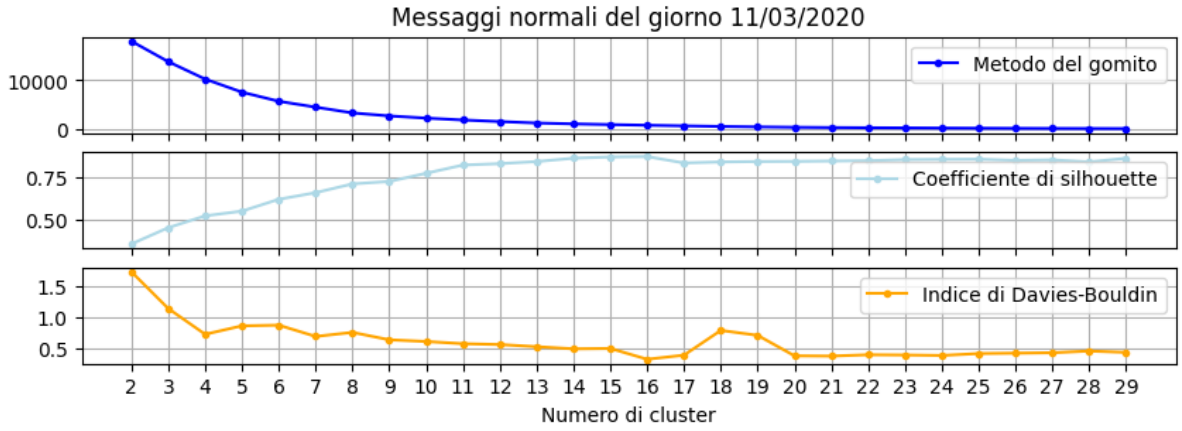

Figura 4.14: *Ricerca del numero ottimale di cluster per i messaggi normali del giorno 11/03/2020*

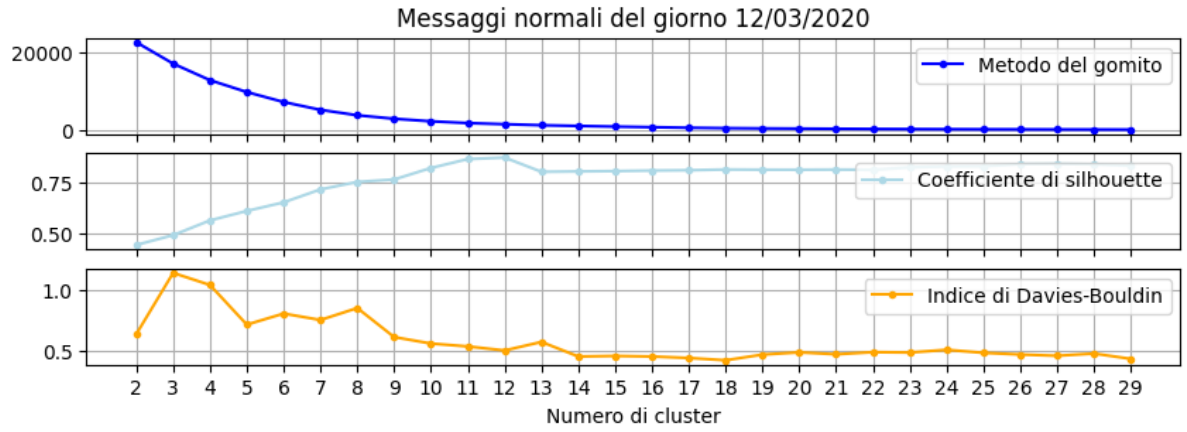

Figura 4.15: *Ricerca del numero ottimale di cluster per i messaggi normali del giorno 12/03/2020*

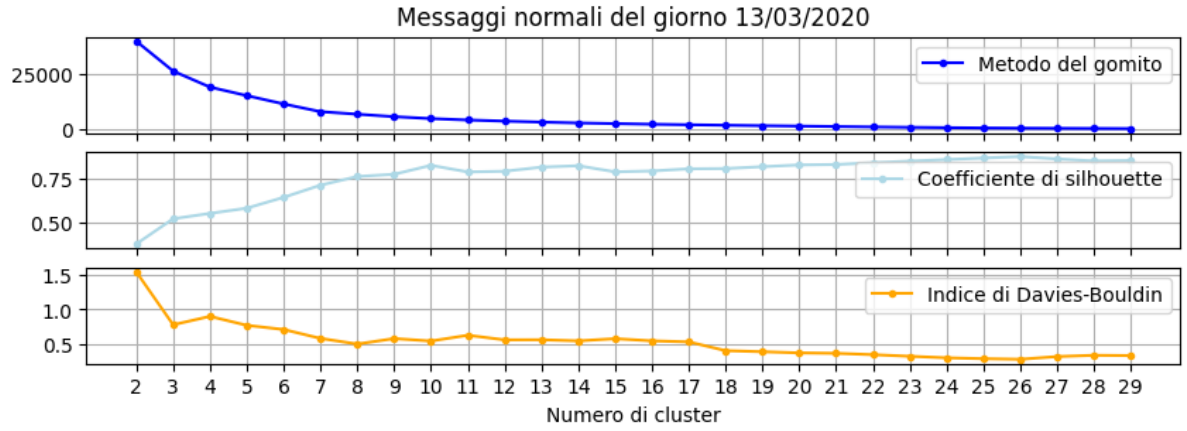

Figura 4.16: *Ricerca del numero ottimale di cluster per i messaggi normali del giorno 13/03/2020*

L'esito della ricerca del numero ottimale di cluster sui messaggi d'errore è riporta in Fig. 4.17, da cui si può vedere che nei primi giorni il numero ottimale è tre o quattro, mentre si vede un aumento a cinque cluster nei giorni dal 10 al 12, per poi tornare a quattro cluster il giorno 13. Il passaggio da quattro cluster a tre dal giorno 7 al giorno 8 potrebbe essere dovuto alla grande differenza nel numero di processi avvenuti nei due giorni.

In Fig. 4.18 è riportato l'esito della ricerca sui messaggi normali, da cui si vede una maggiore discordanza fra i risultati dei tre metodi. Dal grafico si vede che nei primi quattro giorni il numero di cluster è sostanzialmente costante intorno a 12, mentre il giorno 11 è presente un picco a 16 cluster, per poi tornare ad abbassarsi nei due giorni successivi.

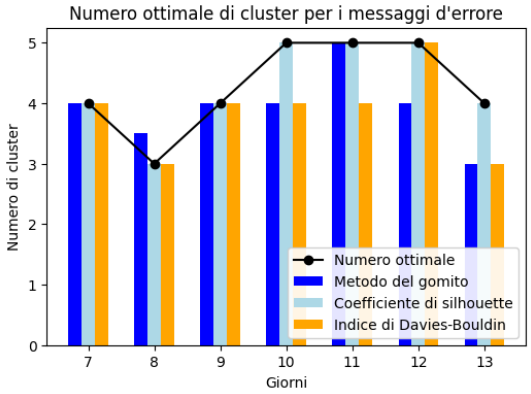

Figura 4.17: *Risultati della ricerca del numero ottimale di cluster sui messaggi d'errore*

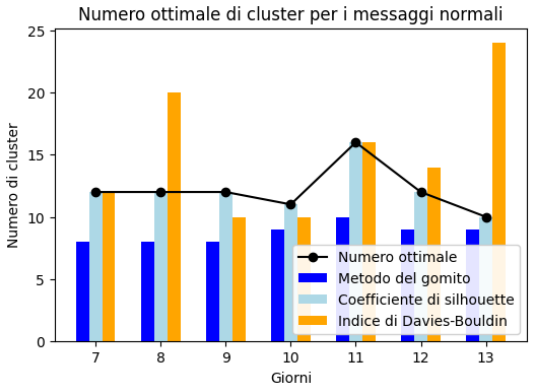

Figura 4.18: *Risultati della ricerca del numero ottimale di cluster sui messaggi normali*

4.2 Risultati del clustering per i messaggi d'errore

Osservando i cluster del giorno 7 (Fig. 4.19) si può vedere che questi si differenziano fra loro soprattutto per la lunghezza dei messaggi e il numero di volte in cui compare la stringa 'SRM_FAILURE'. In particolare si può notare che i cluster 1 e 3 sono molto simili, infatti l'unica differenza fra i due è il *client* che in un caso è 'ATLAS Pilot1' e nell'altro è 'ATLAS Data Management'. Mettendo poi a confronto il grafico del giorno 7 con quello del giorno 8 (Fig. 4.20) si trovano gli stessi cluster, fatta eccezione per il cluster 3 del giorno 7, che probabilmente scopare per via del grande calo nel numero di processi avvenuti nel giorno 8. Per i cluster presenti in entrambi i giorni si osservano delle occupazioni relative abbastanza simili.

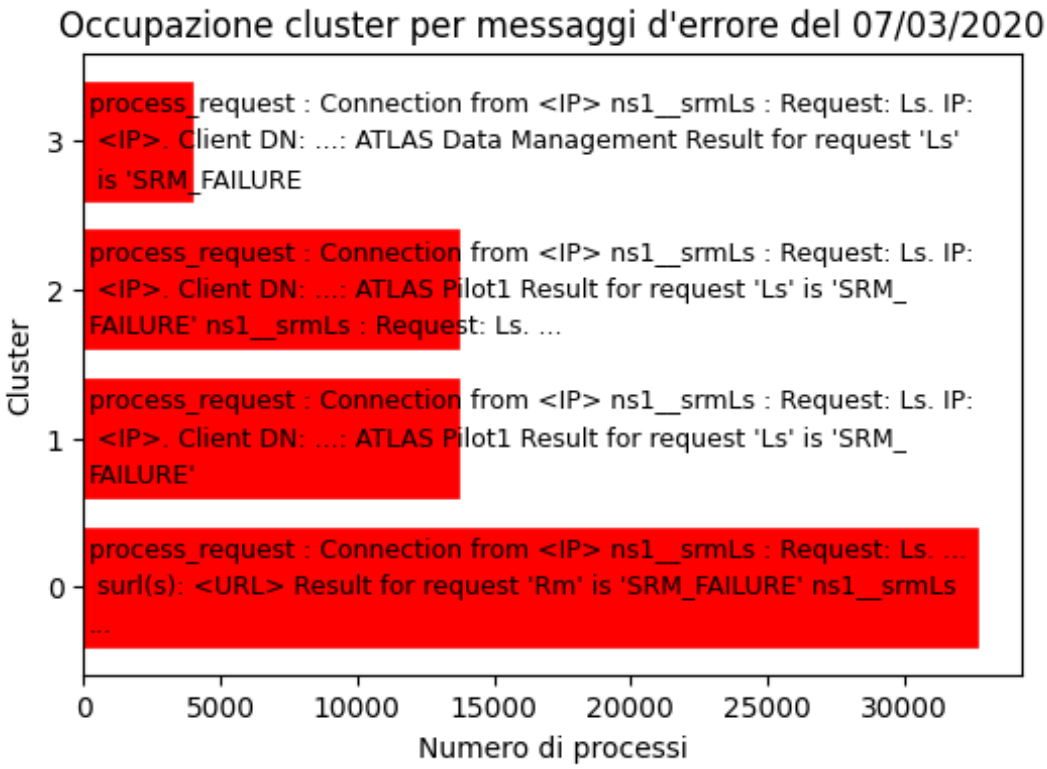

Figura 4.19: *Occupazione dei cluster per i messaggi d'errore del giorno 07/03/2020*

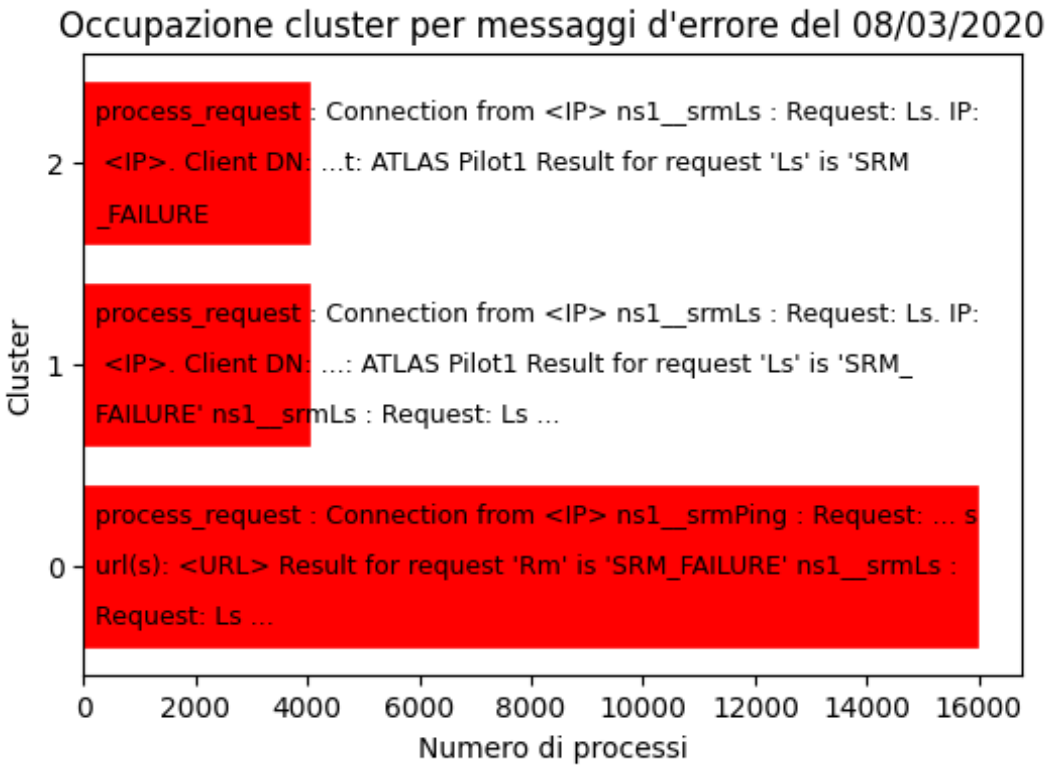

Figura 4.20: *Occupazione dei cluster per i messaggi d'errore del giorno 08/03/2020*

Se si va a confrontare il grafico del giorno 9 (Fig. 4.21) con quelli dei due giorni precedenti si nota che sono presenti gli stessi cluster del giorno 7, ma con delle occupazioni relative molto differenti, infatti il giorno 7 solo un cluster aveva un'occupazione molto elevata, mentre il giorno 9 sono presenti tre cluster con occupazione elevata.

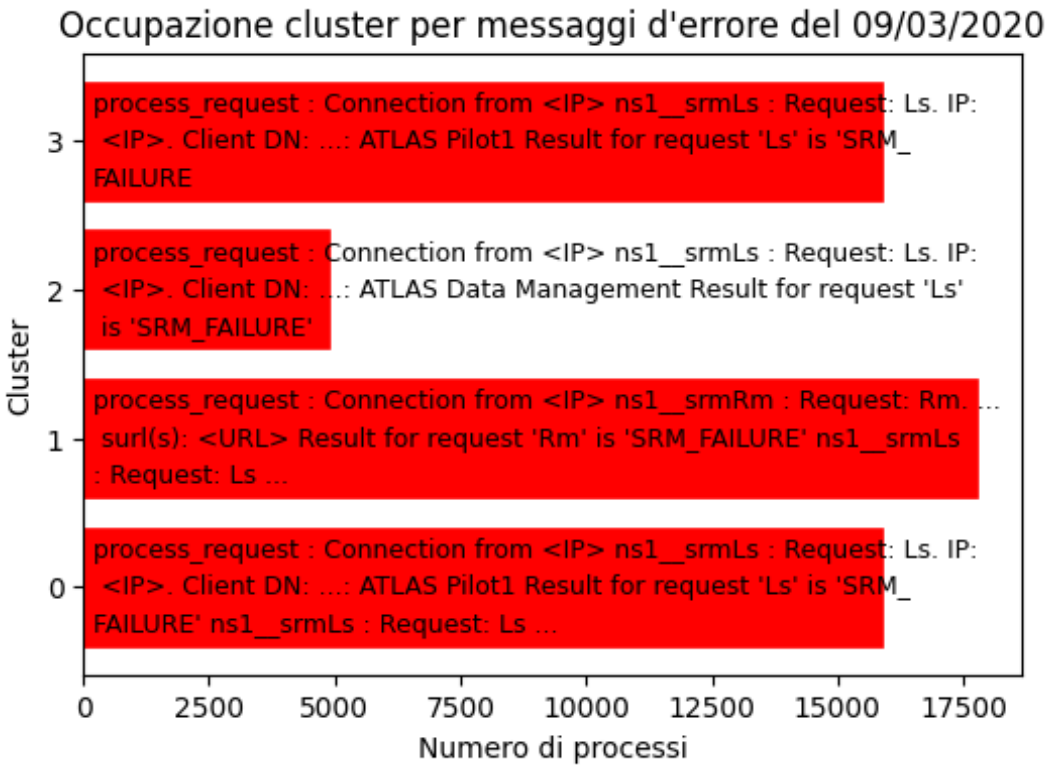

Figura 4.21: *Occupazione dei cluster per i messaggi d'errore del giorno 09/03/2020*

Nel grafico del giorno 10 (Fig. 4.22) è presente un nuovo cluster, il numero 4, contenente messaggi di fallimento in seguito alla richiesta 'Abort files'; questo cluster ha un'occupazione molto bassa se comparata agli altri quattro che hanno un'occupazione molto elevata e simile fra loro.

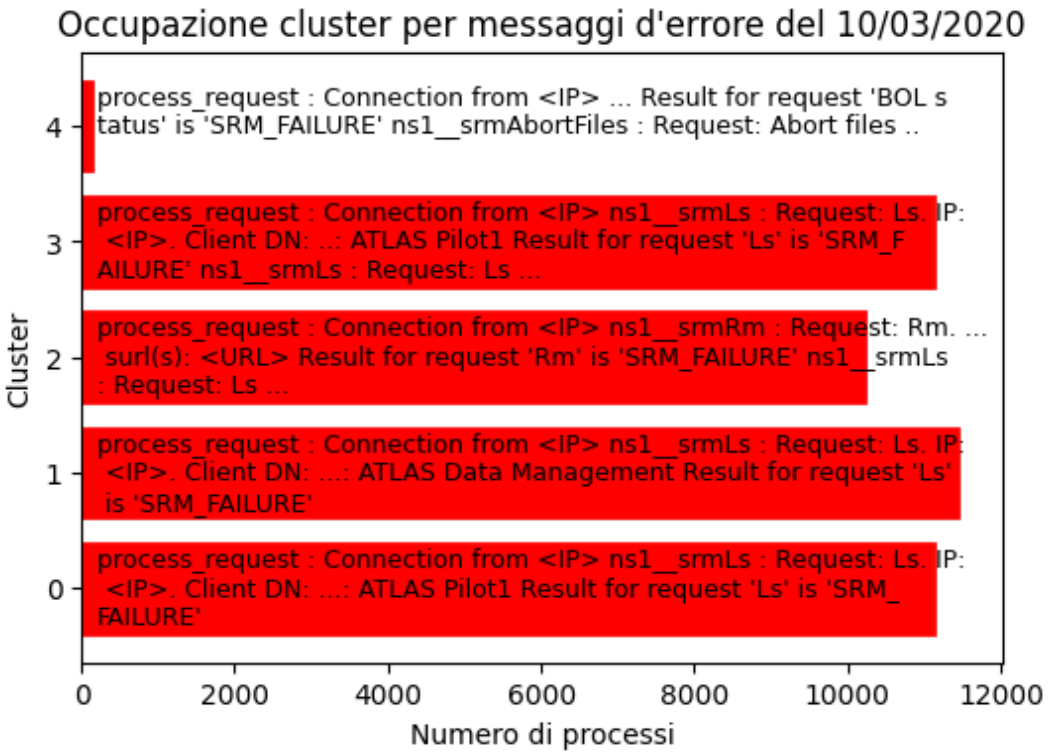

Figura 4.22: *Occupazione dei cluster per i messaggi d'errore del giorno 10/03/2020*

Nel grafico del giorno 11 (fig 4.23) sono presenti cinque cluster come nel giorno precedente, ma in questo caso il cluster 4 si differenzia dagli altri per la maggiore frequenza con cui compare 'SRM_FAILURE'. Gli altri quattro cluster sono gli stessi dal giorno 10 al giorno 11, ma nel giorno 11 torna ad esserci un solo cluster con occupazione elevata, lo stesso che aveva un'occupazione elevata nei primi giorni.

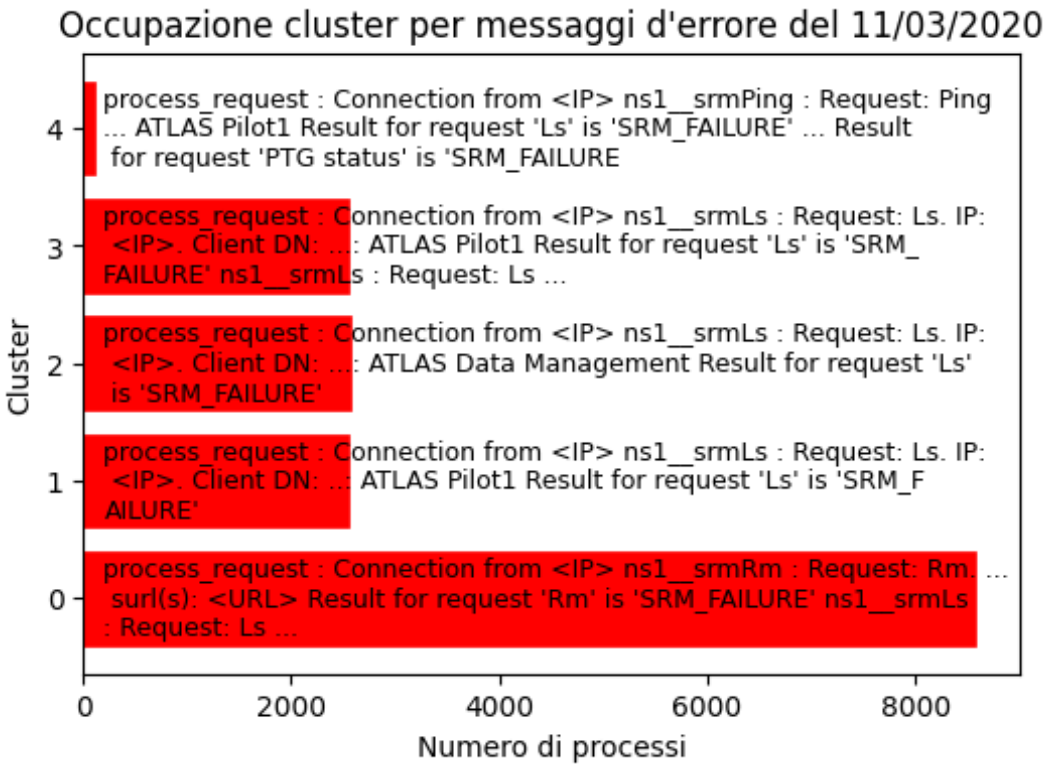

Figura 4.23: *Occupazione dei cluster per i messaggi d'errore del giorno 11/03/2020*

Nel grafico del giorno 12 (Fig. 4.24) sono presenti ancora cinque cluster, i primi quattro sono gli stessi presenti nei primi giorni e anche la loro occupazione relativa è molto simile a quella dei cluster del giorno 7. Il cluster 4 ha un occupazione bassissima e contiene messaggi tutti uguali fra loro.

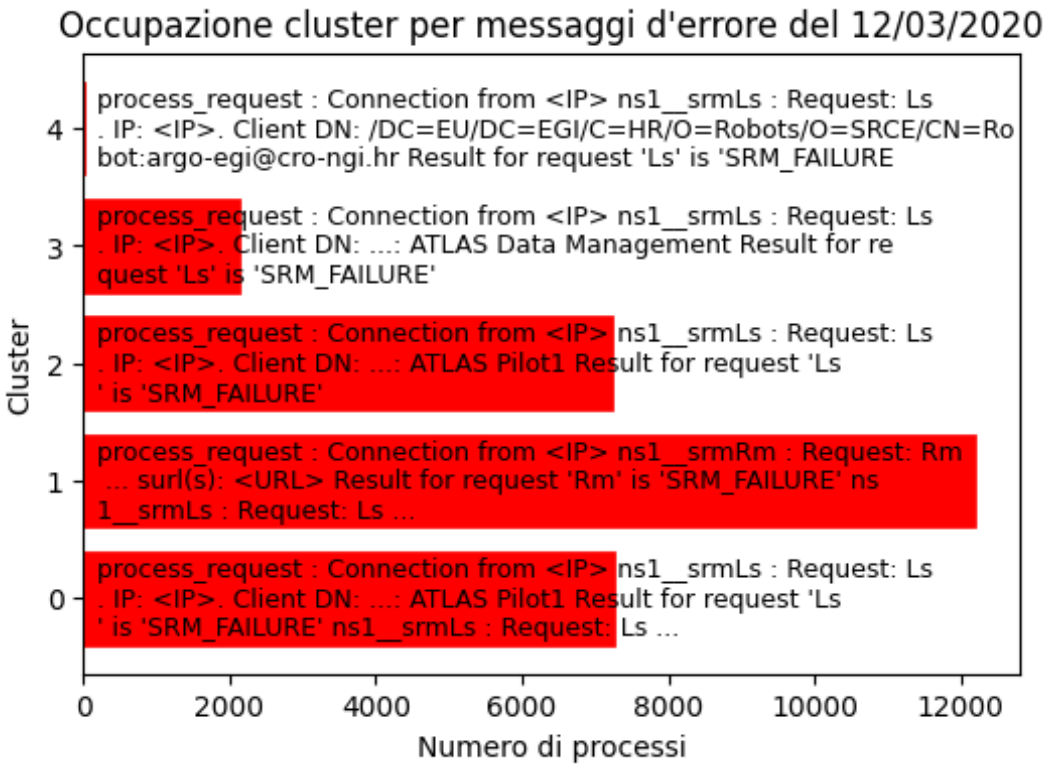

Figura 4.24: *Occupazione dei cluster per i messaggi d'errore del giorno 12/03/2020*

Nel grafico del giorno 13 (Fig. 4.25) si tornano ad avere quattro cluster e sono gli stessi quattro cluster presenti in tutti i giorni precedenti, eccetto il giorno 8.

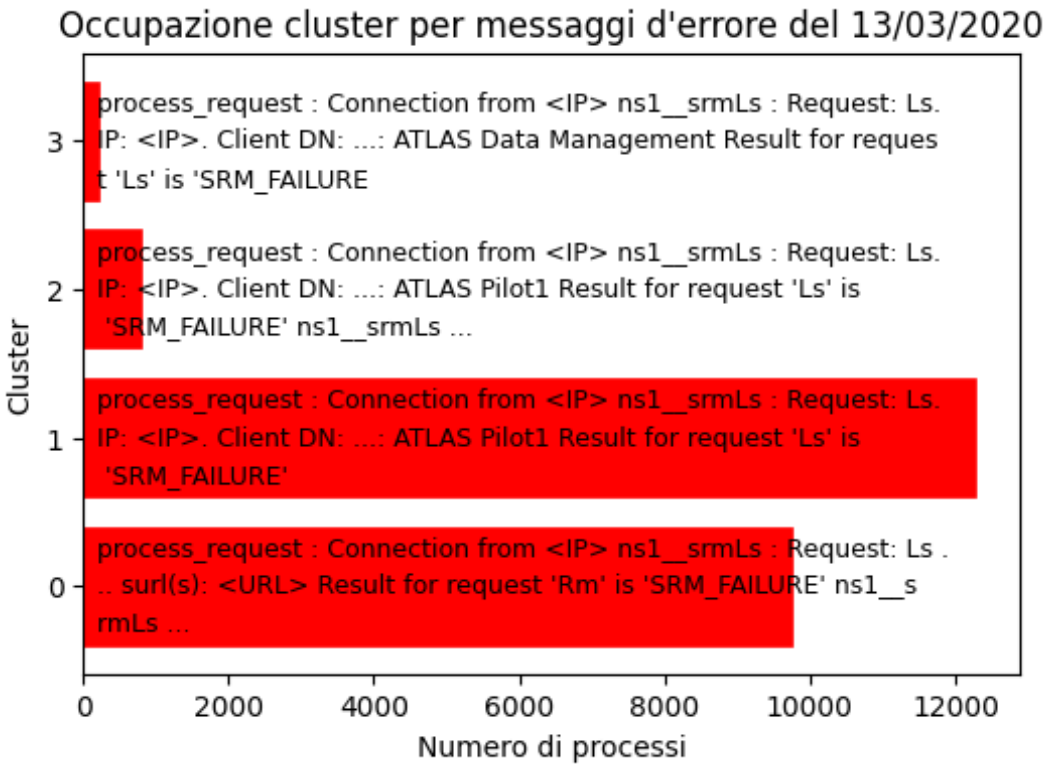

Figura 4.25: *Occupazione dei cluster per i messaggi d'errore del giorno 13/03/2020*

4.3 Risultati del clustering per i messaggi normali

Dai grafici dei primi quattro giorni, dal 7 al 10 (Fig. 4.26, 4.27, 4.28 e 4.29) si vede che il numero di cluster rimane praticamente costante e in media pari a dodici cluster.

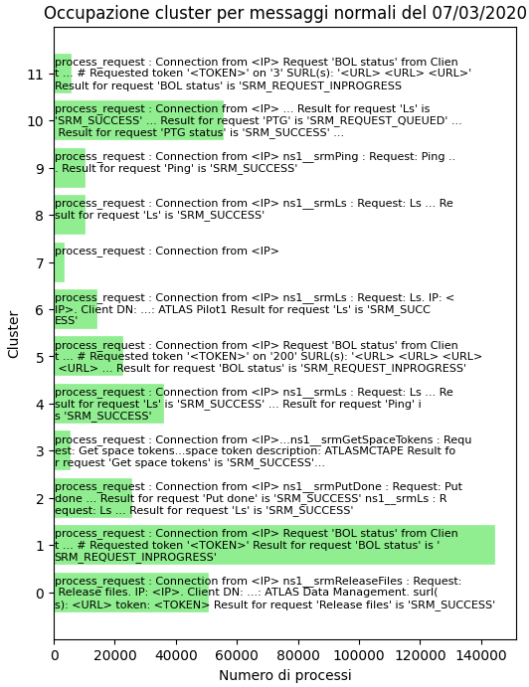

Figura 4.26: *Occupazione dei cluster per i messaggi normali del giorno 07/03/2020*

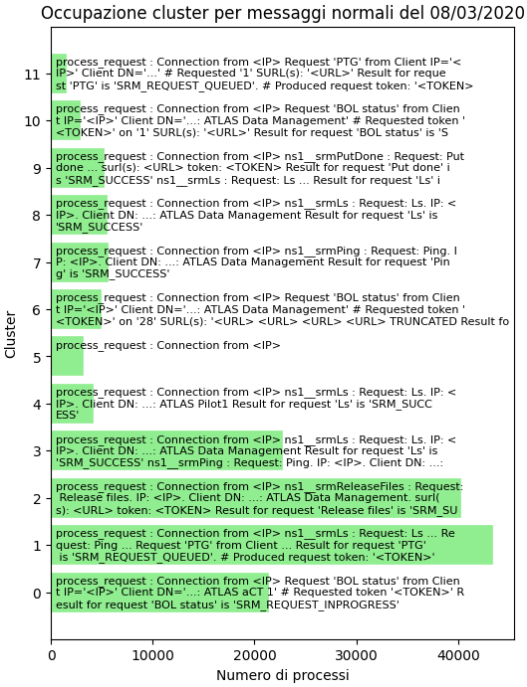

Figura 4.27: *Occupazione dei cluster per i messaggi normali del giorno 08/03/2020*

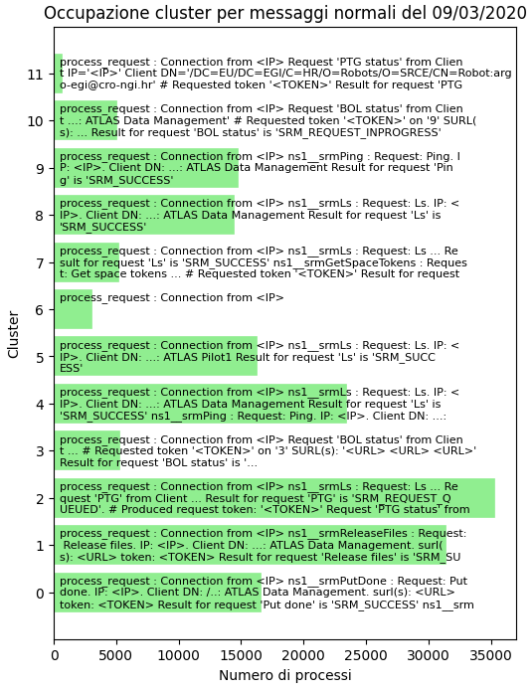

Figura 4.28: *Occupazione dei cluster per i messaggi normali del giorno 09/03/2020*

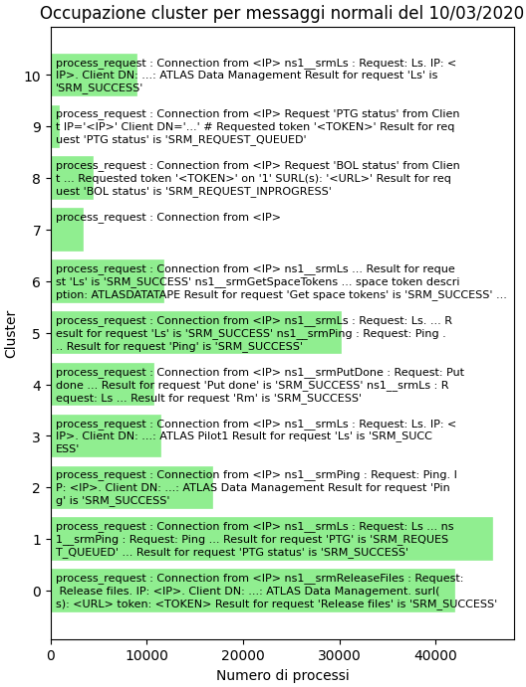

Figura 4.29: *Occupazione dei cluster per i messaggi normali del giorno 10/03/2020*

Il grafico del giorno 11 (Fig. 4.30) mostra che il numero di cluster aumenta fino a sedici. Da notare il cluster 14 che raggruppa i messaggi con la stringa 'Protocol check failed'; questa stringa in realtà rappresenta un errore, ma al momento della divisione dei file sono state utilizzate le parole 'error' e 'failure' per distinguere i messaggi d'errore quindi questi processi sono stati ignorati in fase di divisione. Si può comunque vedere qual è la percentuale di processi in una giornata che contengono questo errore; il giorno 7 la percentuale è pari a allo 0.1% e sale allo 0.25% il giorno 8. Nei due giorni successivi la percentuale rimane abbastanza costante con 0.18% il giorno 9 e 0.21% il giorno 10. Il giorno 11 la percentuale di processi con 'protocol check failed' sale al 0.34% e anche i giorni successivi rimane alta con 0.29% il giorno 12 e 0.38% il giorno 13.

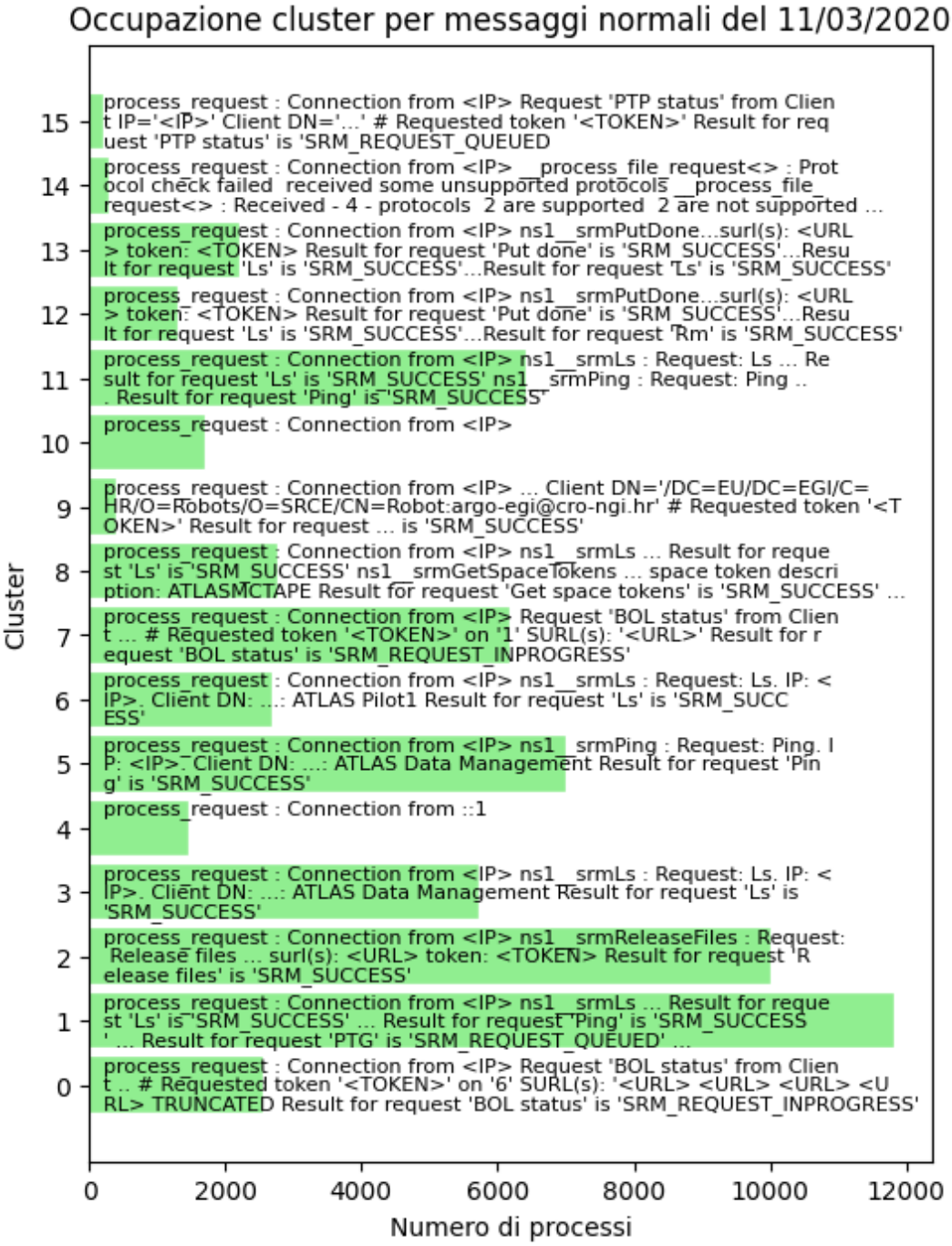

Figura 4.30: *Occupazione dei cluster per i messaggi normali del giorno 11/03/2020*

Il grafico del giorno 12 (Fig. 4.31) ha dodici cluster, come i grafici dei primi giorni, mentre quello del giorno 13 (Fig. 4.32) ne ha dieci. Entrambi i grafici non presentano

più il cluster con la stringa 'protocol check failed'.

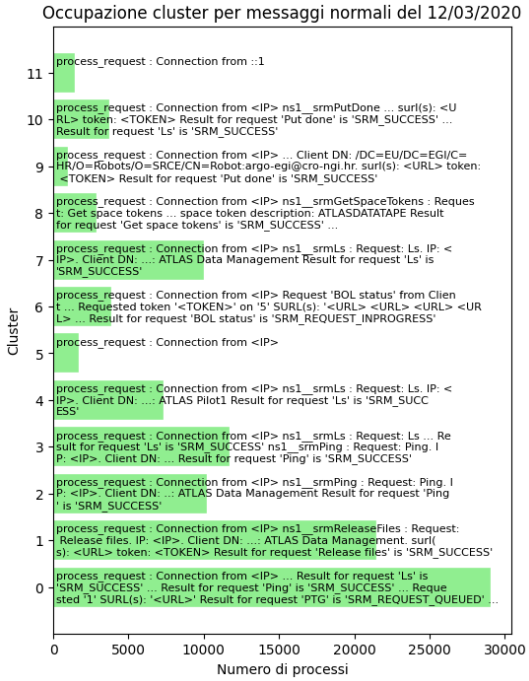

Figura 4.31: *Occupazione dei cluster per i messaggi normali del giorno 12/03/2020*

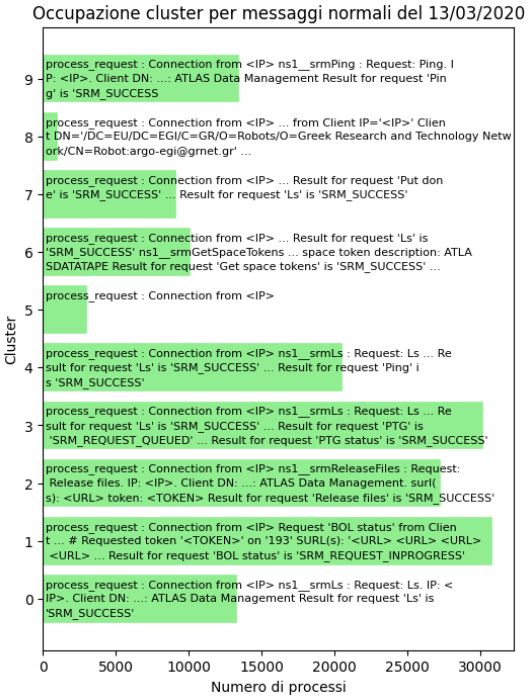

Figura 4.32: *Occupazione dei cluster per i messaggi normali del giorno 13/03/2020*

I risultati dell'analisi indicano che nei giorni che vanno dall'8 al 10 hanno iniziato a presentarsi dei problemi nel sistema di Frontend, come si può intuire dall'aumento percentuale dei processi contenenti 'protocol check failed' il giorno 8, seguito dall'aumento della percentuale di messaggi d'errore il giorno 9 e il successivo aumento del numero di cluster il giorno 10. Questi problemi sono peggiorati il giorno 11 con l'ulteriore aumento dei processi contenenti 'protocol check failed', accompagnato dall'aumento nel numero di cluster per i messaggi normali. A supporto di queste ipotesi, all'interno del logfiles del giorno 11 è indicato che il servizio di StoRM è stato riavviato a seguito dell'identificazione di un problema.

4.4 Sviluppo

Il lavoro presentato in questa tesi ha lo scopo di studiare e testare uno dei possibili algoritmi di apprendimento non supervisionato, che permette di semplificare l'analisi dei

logfiles. Gli sviluppi più prossimi di questo lavoro sono il miglioramento dell'algoritmo e la sua implementazione in un programma in grado di analizzare quotidianamente i file, riconoscendo e riorganizzando le informazioni principali, utili per valutare lo stato del sistema. Queste informazioni, messe a disposizione degli amministratori di sistema, permetterebbero di individuare con più facilità un problema che ha reso necessario il riavvio del servizio, o addirittura individuare un'anomalia prima che questa comporti un malfunzionamento. Tutto ciò oltre a rendere più efficiente il servizio di StoRM comporterebbe anche un alleggerimento nel carico di lavoro degli operatori umani incaricati di ispezionare i logfiles. Lo scopo ultimo sarebbe poi quello di implementare un sistema completamente autonomo in grado di riconoscere le anomalie e agire di conseguenza, senza l'intervento umano.

Conclusioni

Come si è visto nel primo capitolo di questa tesi, gli esperimenti di fisica delle alte energie che hanno luogo nei laboratori del CERN producono un'ingente quantità di dati, che devono poi essere archiviati, manipolati e condivisi in tutto il resto del mondo. In risposta a queste esigenze sono state create la WLCG, i centri di calcolo, come il Tier-1 del CNAF di Bologna, e sono stati sviluppati sistemi di gestione dello staorage come StoRM. Per aumentare l'efficienza di queste infrastrutture, in previsione del futuro incremento di dati prodotti dagli esperimenti, una delle possibili soluzioni è sviluppare un sistema che in autonomia sia in grado di analizzare i logfiles prodotti dai servizi di StoRM evidenziando la presenza di anomalie, così da poter intervenire prima di un malfunzionamento.

Per poter raggiungere questo obbiettivo è prima necessario individuare un algoritmo di apprendimento non supervisionato adatto all'analisi del file di log. Lo scopo di questa tesi è quello di testare un algoritmo di apprendimento non supervisionato, mostrando la sua utilità anche per possibili sviluppi di Predictive Maintenance da utilizzare per la gestione del Tier-1. In questo lavoro sono stati utilizzati i file di log prodotti nell'arco di una settimana dallo StoRM Frontend del Tier-1 di Bologna, che sono stati processati in modo da potervi applicare l'algoritmo di clustering scelto, ossia l'algoritmo K-means. Il codice sviluppato per questo lavoro è stato scritto interamente in Python e sopratutto utilizza esclusivamente librerie open-source come Pandas e Scikit-learn, che vengono sottoposte a continui aggiornamenti e controlli, il che rende il codice efficiente e liberamente fruibile.

I risultati mostrano che l'algoritmo mette in evidenza delle anomalie, evidentemente osservate anche dagli operatori umani che hanno deciso di riavviare il servizio di StoRM in data 11/03/2020. Le anomalie sono emerse osservando i cluster dei messaggi d'errore, che nei giorni precedenti al riavvio sono aumentati in numero e in occupazione. Questi risultati indicano che l'algoritmo è in grado di mettere in evidenza le informazioni utili al monitoraggio dal sistema e di conseguenza può essere utilizzato da un operatore umano per analizzare i file di log in maniera più efficiente e veloce. Questo lavoro mostra che è possibile sviluppare un sistema in grado di monitorare autonomamente i file di log e rappresenta il primo passo per la sua realizzazione.

Bibliografia

[1] CERN, *The Large Hadron Collider*. indirizzo: https://home.cern/science/accelerators/large-hadron-collider.

[2] ——, *Accelerators*. indirizzo: https://home.cern/science/accelerators.

[3] ——, *ATLAS*. indirizzo: https://home.cern/science/experiments/atlas.

[4] ——, *CMS*. indirizzo: https://home.cern/science/experiments/cms.

[5] ——, *ALICE*. indirizzo: https://home.cern/science/experiments/alice.

[6] ——, *LHCb*. indirizzo: https://home.cern/science/experiments/lhcb.

[7] ——, *MoEDAL-MAPP*. indirizzo: https://home.web.cern.ch/science/experiments/moedal-mapp.

[8] ——, *FASER*. indirizzo: https://home.web.cern.ch/science/experiments/faser.

[9] ——, *High-Luminosity LHC*. indirizzo: https://home.cern/science/accelerators/high-luminosity-lhc.

[10] ATLAS Experiment, *The ATLAS Experiment*. indirizzo: https://atlas.cern/about.

[11] The ATLAS Collaboration, G. Aad, E. Abat et al., "The ATLAS Experiment at the CERN Large Hadron Collider," *Journal of Instrumentation*, vol. 3, n. 08, pp. 1–16, 53–109, 218–256, ago. 2008. DOI: 10.1088/1748-0221/3/08/s08003. indirizzo: https://doi.org/10.1088/1748-0221/3/08/s08003.

[12] CMS Experiment, *Detector*. indirizzo: https://cms.cern/detector.

[13] CERN, *How did we discover the Higgs boson?* Indirizzo: https://home.cern/science/physics/higgs-boson/how.

[14] The ATLAS Collaboration, "Observation of a new particle in the search for the Standard Model Higgs boson with the ATLAS detector at the LHC," *Physics Letters B*, vol. 716, n. 1, pp. 1–29, set. 2012, ISSN: 0370-2693. DOI: https://doi.org/10.1016/j.physletb.2012.08.020. indirizzo: https://www.sciencedirect.com/science/article/pii/S037026931200857X.

[15] CERN, *What have we learned since the Higgs boson discovery?* Indirizzo: https://home.cern/science/physics/higgs-boson/ten-years.

[16] ——, *Storage*. indirizzo: https://home.cern/science/computing/storage.

[17] WLCG, *TIERS OF WLCG*. indirizzo: https://wlcg-public.web.cern.ch/tiers.

[18] L. Morganti, A. Costantini, L. dell'Agnello e F. Fornari, "INFN-CNAF Annual Report 2019," 2019. indirizzo: https://www.cnaf.infn.it/wp-content/uploads/2020/05/cnaf-annual-report-2019.pdf.

[19] F. Schmuck e R. Haskin, "Gpfs: A shared-disk file system for large computing clusters Proceedings of the 1st USENIX Conference on File and Storage Technologies FAST'02 (Berkeley, CA, USA: USENIX Association)," pp. 16–16, 2002. indirizzo: http://dl.acm.org/citation.cfm?id=1973333.1973349.

[20] IBM, *Tivoli Storage Manager*. indirizzo: https://www.ibm.com/support/knowledgecenter/SSGSG7_7.1.0/com.ibm.itsm.srv.doc/c_tsmintro.html.

[21] StoRM, *StoRM: a Manager for Storage Resource in Grid*. indirizzo: http://italiangrid.github.io/storm/documentation/functional-description/1.11.2/.

[22] P. McCorduck, "Machines who think : a personal inquiry into the history and prospects of artificial intelligence," in 2ª ed. Natick, Massachusetts: A K Peters, Ltd., 1940, cap. 5. indirizzo: https://monoskop.org/images/1/1e/McCorduck_Pamela_Machines_Who_Think_2nd_ed.pdf.

[23] A. Radovic, M. Williams, D. Rousseau et al., "Machine learning at the energy and intensity frontiers of particle physics," *Nature*, vol. 560, pp. 41–48, 2018, ISSN: 1476-4687. DOI: 10.1038/s41586-018-0361-2. indirizzo: https://doi.org/10.1038/s41586-018-0361-2.

[24] A. M. Turing, "I.—COMPUTING MACHINERY AND INTELLIGENCE," *Mind*, vol. LIX, n. 236, pp. 433–460, ott. 1950, ISSN: 0026-4423. DOI: 10.1093/mind/LIX.236.433. indirizzo: https://doi.org/10.1093/mind/LIX.236.433.

[25] A. L. Samuel, "Some Studies in Machine Learning Using the Game of Checkers," *IBM Journal of Research and Development*, vol. 3, n. 3, pp. 210–229, 1959. DOI: 10.1147/rd.33.0210.

[26] Jupyter, *About Us*. indirizzo: https://jupyter.org/about.

[27] Pandas, *About pandas*. indirizzo: https://pandas.pydata.org/about/.

[28] F. Pedregosa, G. Varoquaux, A. Gramfort et al., "Scikit-learn: Machine Learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[29] NumFOCUS, *The NumFOCUS Mission*. indirizzo: https://numfocus.org/community/mission.

[30] Scikit-learn, *K-means*. indirizzo: https://Scikit-learn.org/dev/modules/clustering.html?highlight=k+means#k-means.

[31] ——, *sklearn.metrics: Metrics*. indirizzo: https://Scikit-learn.org/dev/modules/classes.html?highlight=sklearn+metrics#module-sklearn.metrics.

[32] L. Clissa, *fts-errors-clustering*, url https://github.com/operationalintelligence/fts-errors-clustering.git, 2019.

[33] Scikit-learn, *Text feature extraction*. indirizzo: https://Scikit-learn.org/stable/modules/feature_extraction.html#text-feature-extraction.

[34] ——, *Truncated singular value decomposition and latent semantic analysis*. indirizzo: https://scikit-learn.org/stable/modules/decomposition.html#lsa.