Alma Mater Studiorum · Università di Bologna

SCUOLA DI SCIENZE

Corso di Laurea in Informatica per il Management

Machine Learning Approaches for Job Failure Prediction in HTC Systems

Relatore:

Prof. Moreno Marzolla

Presentata da:

Alessio Arcara

Correlatore:

Dott. Stefano Dal Pra

Seconda Sessione di Laurea

Anno Accademico 2022 - 2023

Sommario

Il CNAF gestisce un centro di calcolo dotato di oltre 46000 core distribuiti su 960 host fisici. I job vengono accodati e schedulati dal sistema batch (HTCondor) attraverso l'uso di algoritmi di "fairshare". Durante l'esecuzione, vengono monitorate grandezze quali il consumo di memoria e lo spazio su disco, che vengono campionate ogni tre minuti e raccolte in un database insieme ai dati di accounting relativi ai job terminati. Questi job possono variare notevolmente in termini di durata, da pochi minuti a più giorni. Questo studio esplora l'uso di tecniche di Machine Learning per prevedere il successo o il fallimento dei job, basandosi sull'evoluzione del loro stato nel tempo. In particolare, ci si è concentrati sui cosiddetti "job zombie", ossia quei job che, pur terminando, non rilasciano l'host fisico, causando una perdita di risorse fino al loro timeout. Un approccio iniziale, che prendeva in considerazione solo la prima ora di vita del job, ha permesso di identificare con un'accuratezza del 72% la classe meno rappresentata (i job zombie). Per migliorare l'accuratezza, si è preso in considerazione l'intero primo giorno, applicando tecniche di Deep Learning sia supervisionate (CNN, CNN+LSTM, LSTM e Transformer) che non supervisionate (autoencoder e variational autoencoder). Nonostante l'incremento della complessità dei modelli, le reti neurali hanno mostrato una tendenza all'overfitting a causa dell'estremo sbilanciamento dei dati.

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 1

Introduzione

Capitolo 2

Caso di studio

In questo capitolo presenteremo una panoramica del centro di calcolo presso il quale è stato svolto il tirocinio, guardando da dove provengono i dati e come vengono raccolti. Verranno infine presentati gli obiettivi di questo studio e della tesi che ne deriva.

2.1 Il cluster di calcolo del CNAF

Il **grid computing** è un'architettura di calcolo distribuito che collega computer sparsi geograficamente allo scopo di condividere risorse e potenza di calcolo per raggiungere uno scopo condiviso. Attualmente, il più grande sistema grid al mondo è il Worldwide LHC Computing Grid (WLCG), che nasce da una collaborazione internazionale che coinvolge oltre 170 centri di calcolo sparsi in più di 40 nazioni. Lo scopo del WLCG è fornire l'infrastruttura computazionale necessaria per gestire i dati generati dagli esperimenti effettuati con il Large Hadron Collider (LHC) [4].

Come illustrato nella figura 2.1, i centri di calcolo all'interno del WLCG sono strutturati secondo il modello MONARC, che li organizza in un sistema gerarchico di livelli, noti come Tier, ciascuno dei quali ha funzioni e responsabilità ben definite. In questo contesto si colloca il centro nazionale delle tecnologie informatiche e telematiche (CNAF), che ospita il Tier-1 per tutti e quattro gli esperimenti del LHC. Oltre a questi ultimi, vengono supportati presso il CNAF anche gli esperimenti non-LHC di astrofisica delle particelle e fisica dei neutrini [2].

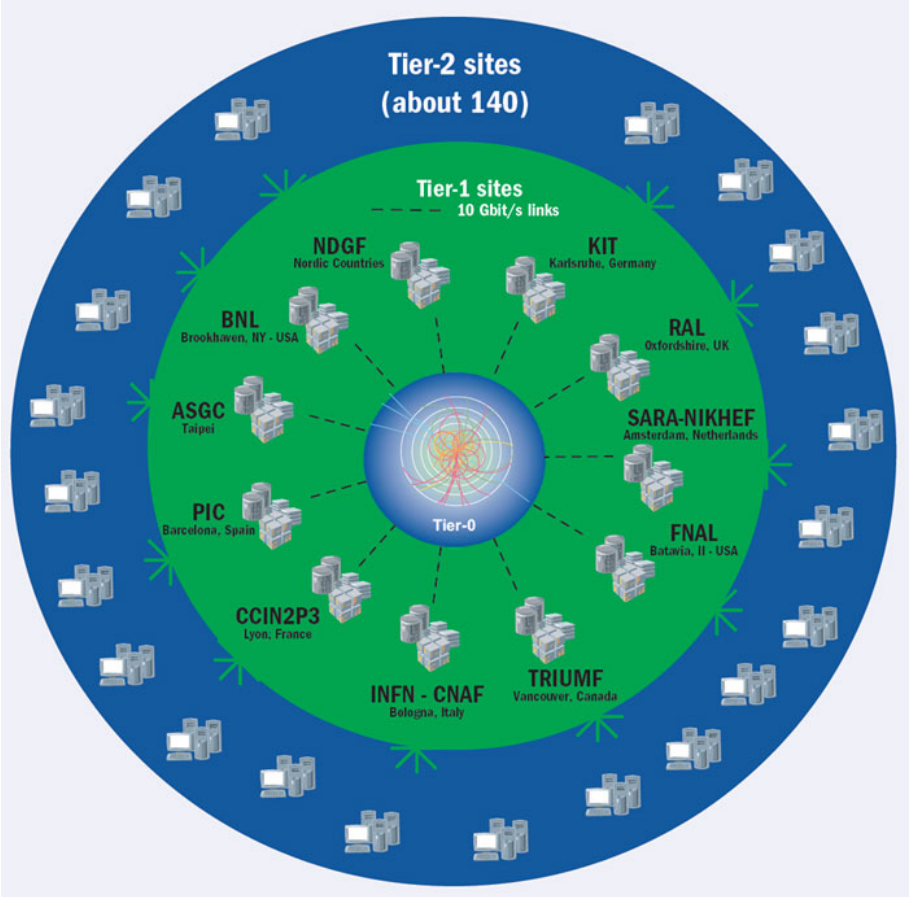

Figura 2.1: Struttura gerarchica del WLCG [6]

Il CNAF offre più di 46000 core distribuiti su 960 host fisici per un totale di circa 630 kHS06[1] di potenza di calcolo [11]. L'allocazione di queste risorse segue il paradigma del **High-Throughput Computing** (HTC), dove a differenza dell'High-Performance Computing, che mira a ridurre il tempo di esecuzione dei programmi, l'obiettivo dell'HTC è massimizzare il throughput, inteso come il numero di job completati per unità di tempo.

In questo sistema, gli utenti sono raggruppati in circa 50 gruppi distinti, ciascuno dei quali corrisponde a un esperimento scientifico specifico. A ogni gruppo è assegnata una quota di risorse che può utilizzare. Quando un utente ha bisogno di utilizzare queste risorse, può sottomettere un **job** al sistema, che rappresenta una o più operazioni

[1] metrica per misurare le prestazioni della CPU, sviluppata dal gruppo di lavoro HEPiX. È utilizzata per confrontare le risorse di calcolo in ambito scientifico.

computazionali.

Una volta sottomesso, il job non viene eseguito immediatamente, ma viene messo in una coda gestita da un batch system (HTCondor). Quest'ultimo è responsabile della schedulazione dei job in coda, decidendo quale job eseguire, quando e dove. Per farlo, utilizza algoritmi di "fairshare", che sono pensati per assicurare una distribuzione equa delle risorse computazionali disponibili, impedendo che un singolo utente o un intero gruppo possa monopolizzare tutte le risorse disponibili.

Se un gruppo non utilizza la quota di risorse assegnata, queste vengono redistribuite tra i gruppi attivi in proporzione alla loro quota. Questo meccanismo assicura che la farm di calcolo lavori quasi sempre alla sua massima capacità, ottimizzando l'uso delle risorse nel lungo termine [5].

2.2 La base di dati

Il caso di studio di questa tesi si basa su informazioni provenienti da due fonti principali: la prima è ottenuta attraverso il monitoraggio dei job in esecuzione, effettuato tramite il comando `condor_q` di HTCondor, eseguito ogni 3 minuti. La seconda proviene dai file *history*, generati automaticamente da HTCondor al termine dell'esecuzione di ciascun job.

Successivamente uno script estrae le informazioni rilevanti dai dati di accounting; queste informazioni vengono poi inserite nella tabella `htjob` di un database PostgresSQL. Analogamente, i dati di monitoraggio vengono raccolti e caricati su una tabella `hj`.

La raccolta dei dati è stata effettuata in due periodi distinti: il primo da settembre a dicembre 2021, e il secondo nel mese di marzo 2023. I dati sono stati immagazzinati in due database separati, identificati come `htm` per i dati del primo periodo e `htmnew` per quelli del secondo.

La tabella 2.1 mostra il numero totale di righe e lo spazio occupato su disco da ciascuna tabella nei database. Dato che la dimensione del dataset supera la capacità della memoria RAM a disposizione, risulta impossibile analizzare l'intero dataset. Pertanto, diventa necessario selezionare un sottoinsieme di dati da tali database per effettuare le analisi successive.

	htm	
	Righe	**Spazio (GB)**
`hj`	1971830783	343
`htjob`	30799153	14

	htmnew	
	Righe	**Spazio (GB)**
`hj`	1038471316	222
`htjob`	46605815	22

Tabella 2.1: Confronto delle dimensioni tra i database `htm` e `htmnew`

Le tabelle 2.2 e 2.3 offrono una panoramica sulle colonne presenti, distinguendo tra variabili categoriche e numeriche e fornendo una breve spiegazione su ciascuna colonna.

Le variabili si suddividono in base al tipo di dati che rappresentano e si suddividono in:

- *categorico nominale*, se contiene valori scelti tra un insieme finito;
- *categorico ordinale*, è simile, ma si definisce una relazione d'ordine tra i valori possibili;
- *numerico*, se è possibile quantificare le differenze tra valori.

2.3 Motivazione

Nel 1965, Gordon Moore, co-fondatore di Intel, pronosticò che il numero di transistor sarebbe raddoppiato ogni 18 mesi [9]. Tuttavia, il trend descritto da Moore arriverà a un termine quando la litografia, il processo usato per stampare i circuiti sui wafer di silicio, raggiungerà la scala atomica. Infatti, a scale atomiche i transistor incontrano fenomeni quantistici che ne disturbano il funzionamento, e le attuali tecniche di produzione diventano proibitive in termine di costi [12, 13].

I sistemi HTC e HPC sono divenuti strumenti fondamentali per il progresso della ricerca scientifica. Nonostante ciò, vi sono ancora molteplici problemi importanti in vari settori che non possono essere risolti con le capacità computazionali attuali [14]. Per proseguire l'evoluzione tecnologia nell'era post-legge di Moore è quindi necessario esplorare nuove direzioni. Una di queste riguarda l'incremento del numero di core.

Tabella 2.2: Schema della tabella `hj` del database

Colonna	**Tipo**	**Descrizione**
`ts`	Numerico (secondi)	Timestamp UNIX del momento in cui il job è stato eseguito
`jobid + idx`	Categorico (Nominale)	ID univoco del job
`queue`	Categorico (Nominale)	Gruppo di appartenenza dell'utente che ha sottomesso il job
`hn (hostname)`	Categorico (Nominale)	Host sul quale il job è in esecuzione
`js`	Categorico (Nominale)	Stato del job: 1 = In coda, 2 = In esecuzione, 3 = Rimosso, 4 = Completato, 5 = Sospeso
`nc`	Numerico (core)	Numero di core CPU impiegati dal job
`hsj`	Numerico (HS06)	Potenza di un core del host
`hsm`	Numerico (HS06)	Potenza totale del host
`cpt (cputime)`	Numerico (secondi)	Tempo di esecuzione sulla CPU del job
`rt (runtime)`	Numerico (secondi)	Tempo totale di esecuzione del job
`owner`	Testo	Utente che ha sottomesso il job (username UNIX)
`rss`	Numerico (KB)	Memoria RAM utilizzata dal job
`swp`	Numerico (KB)	Memoria SWAP utilizzata dal job
`sn (submitnode)`	Categorico (Nominale)	Nodo da cui è stato sottomesso il job
`disk`	Numerico (GB)	Spazio su disco utilizzato dal job

Tabella 2.3: Schema della tabella `htjob` del database

Colonna	Tipo	Descrizione
`jobid + idx`	Categorico (Nominale)	ID univoco del job
`username`	Testo	Utente che ha sottomesso il job (username UNIX)
`queue`	Categorico (Nominale)	Gruppo di appartenenza dell'utente che ha sottomesso il job
`fromhost`	Categorico (Nominale)	Nodo da cui è stato sottomesso il job
`jobname`	Testo	Nome del job
`exechosts`	Categorico (Nominale)	Host sul quale il job è in esecuzione
`submittimeepoch`	Numerico (secondi)	Timestamp UNIX del momento in cui il job è stato sottomesso
`starttimeepoch`	Numerico (secondi)	Timestamp UNIX del momento in cui il job è stato eseguito
`eventtimeepoch`	Numerico (secondi)	Timestamp UNIX del momento in cui il job è terminato
`stime`	Numerico (secondi)	Tempo di esecuzione sulla CPU per eseguire le chiamate al sistema per conto del job
`utime`	Numerico (secondi)	Tempo di esecuzione sulla CPU dedicato alle operazione che il job esegue direttamente
`runtime`	Numerico (secondi)	Tempo totale di esecuzione del job
`maxrmem`	Numerico (KB)	Massima memoria RAM utilizzata dal job
`maxrswap`	Numerico (KB)	Massima memoria SWAP utilizzata dal job
`exitstatus`	Categorico (Nominale)	= 0 è ok; != 0 è uscito con errore
`numprocessors`	Numerico (core)	Numero di core CPU impiegati dal job
`gpu`	Categorico (Nominale)	1 = gpu utilizzata; 0 = gpu non utilizzata
`completionepoch`	Numerico (secondi)	Timestamp UNIX del momento in cui il job è terminato
`jobstatus`	Categorico (Nominale)	Stato del job: 1 = In coda, 2 = In esecuzione, 3 = Rimosso, 4 = Completato, 5 = Sospeso

In questa tesi, un *guasto* è definito come un comportamento anomalo a livello software o hardware, che può causare stati illeciti (*errori*) nel sistema o nell'applicazione e che, nel peggiore dei casi, può causare l'interruzione dell'applicazione o del sistema (*fallimenti*).

Sfortunatamente, più core si aggiungono, maggiori sono le probabilità di riscontrare guasti hardware. In parallelo, all'aumentare della complessità hardware, si assiste a una crescente complessità del software, il che lo rende più suscettibile agli errori [3].

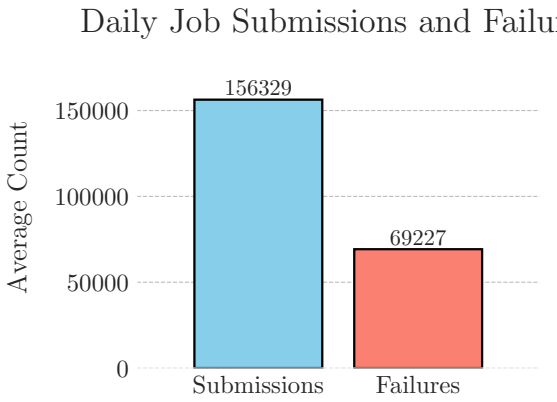

Figura 2.2: Media giornaliera di job sottomessi e falliti nel mese di Marzo 2023

Come evidenziato nella figura 2.2, si osserva che in un centro di calcolo come quello del CNAF, nel mese di Marzo 2023, sono stati sottomessi giornalmente in media 156329 job con un tasso di fallimento che supera il 40%. La frequenza con cui i job falliscono rappresenta una problematica significativa per i centri di calcolo: questa non solo causa uno spreco delle risorse del sistema, ma incide anche negativamente sull'efficienza generale e allunga i tempi d'attesa per i job in attesa di essere eseguiti.

Questa tesi si concentra sui sui fallimenti dei job piuttosto che sui fallimenti a livello di sistema, nonostante la disponibilità di dati degli host fisici del CNAF. Utilizzando tecniche di Machine Learning per identificare pattern nei dati storici, potrebbe essere possibile prevedere la riuscita o il fallimento di un job basandosi sul comportamento di job simili. Questo permetterebbe l'adozione di strategie proattive volte a prevenire i fallimenti prima che accadano, mitigando così i problemi sopracitati. In aggiunta, la capacità di informare l'utente circa il tasso di successo o fallimento di un job che sta per essere sottomesso potrebbe fornire una risorsa informativa utile.

Nel Capitolo 3, vedremo come l'analisi preliminare evidenzierà una categoria di job che falliscono, che risulta essere particolarmente interessante, soprattutto per l'importanza di identificarli e rimuoverli tempestivamente.

Capitolo 3

Analisi del database

In questo capitolo vengono presentate le analisi preliminari effettuate per il caso di studio in questione. Queste analisi hanno permesso di selezionare un sottoinsieme di job e un task di Machine Learning associato che possa apportare benefici al CNAF.

3.1 Analisi esplorativa

3.1.1 Caratterizzazione dei job

In questa tesi considereremo una serie di dati come una sequenza ordinata di punti dati, che esprime la dinamica di un certo fenomeno nel tempo. Quando questi dati sono ordinati in base al tempo, si parla di una **serie storica** (o **temporale**). Indipendentemente dal criterio utilizzato per ordinarli, i punti dati sono registrati seguendo intervalli di tempo equispaziati. Le serie temporali possono essere di due tipi: **univariate**, che coinvolgono una singola variabile misurata nel tempo, e **multivariate**, dove più variabili sono misurate contemporaneamente.

All'interno della tabella `hj`, che contiene i dati di monitoraggio dei job eseguiti da HTCondor, ogni riga può essere vista come un punto in una serie storica multivariata. In altre parole, ciascun job corrisponde a una serie storica multivariata distinta, dove le variabili rappresentano le diverse grandezze misurate durante il suo ciclo di vita, come illustrato nella figura 3.1. Sebbene queste serie condividano le stesse variabili, la durata di ciascun job, e di conseguenza la lunghezza delle relative serie storiche, cambia.

Come mostrato in figura 3.2, HTCondor campiona lo stato di ciascun job ogni tre minuti, ma aggiorna i valori ogni quindici minuti. Questo significa che ogni serie storica mostra un cambiamento effettivo nei valori solo ogni cinque campionamenti, risultando in una sequenza di cinque valori identici che si ripetono fino al prossimo aggiornamento.

Per quanto riguarda l'aggiornamento dei valori, HTCondor aggiorna un nuovo dato all'interno della serie storica solo quando il valore rilevato supera il precedente massimo. Di conseguenza, ciascuna serie storica può essere vista come una funzione monotona non decrescente, in cui ogni nuovo valore registrato è maggiore o uguale al precedente.

La durata dei job varia considerevolmente, come mostrato dalla distribuzione del numero di job rispetto alla loro durata in giorni, illustrata nella figura 3.3. Vi è una predominanza di job di breve durata, con un calo esponenziale del numero di job al crescere della durata.

Raggruppando i job in base alla loro durata in fasce orarie, fino a un massimo di 48 ore, e aggregando tutti quelli che superano tale soglia, è possibile esaminare il numero totale dei job, la frequenza dei loro fallimenti e il cumulativo della loro durata per ciascuna delle prime quarantotto ore, così come per quelli più lunghi. Come evidenziato nella figura 3.4, i job che durano meno di un'ora sono particolarmente numerosi e presentano un elevato tasso di fallimento. Nonostante ciò, il tempo speso sulle risorse di calcolo è pressoché trascurabile se confrontato con il tempo impiegato dai job di durata superiore.

Inoltre, se ci focalizziamo sulla prima ora e suddividiamo i job in intervalli di cinque minuti, possiamo notare che molti di essi hanno una durata inferiore ai cinque minuti, come si può vedere nella figura 3.5. Questo suggerisce che questi job potrebbero essere considerati come semplici tentativi; in altre parole, sono job che, per vari motivi, non trovano le condizioni necessarie per proseguire nella loro esecuzione e quindi falliscono.

In aggiunta, secondo quanto riportato nella tabella 3.1, un significativo 11% dei job viene terminato ancor prima di raggiungere la fase di esecuzione. Questi job, non giungendo alla fase di esecuzione, non effettuano alcun calcolo, il che sottolinea la presenza di un elevato numero di tentativi che risultano irrilevanti in termini di calcolo.

Pertanto, alla luce di queste considerazioni, nasce la seguente idea:

> *Predire il fallimento di un job di lunga durata è nettamente più importante rispetto alla previsione del fallimento di un job di breve durata.*

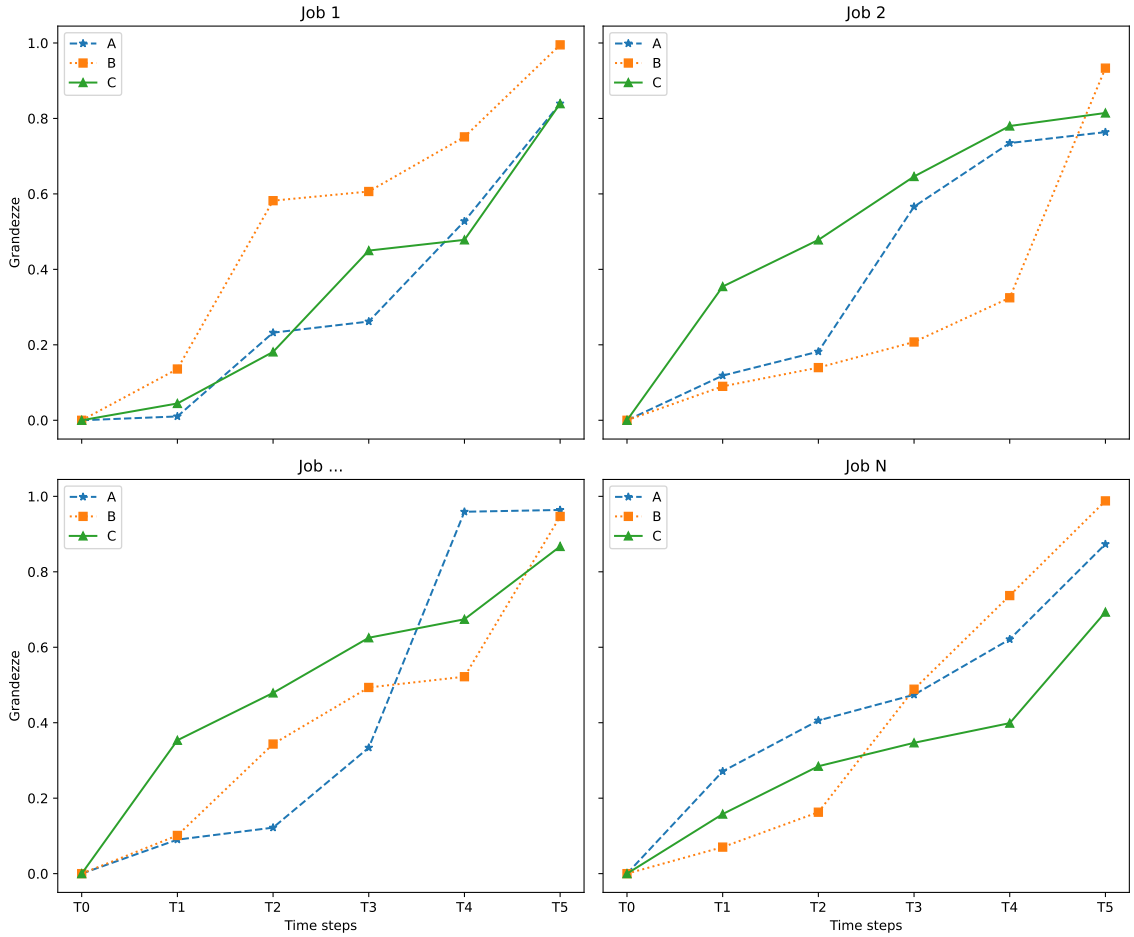

Figura 3.1: Rappresentazione di un job come serie storica multivariata

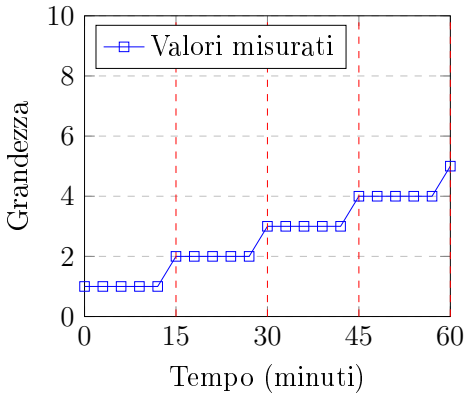

Figura 3.2: Frequenza di campionamento e aggiornamento dei job in HTCondor

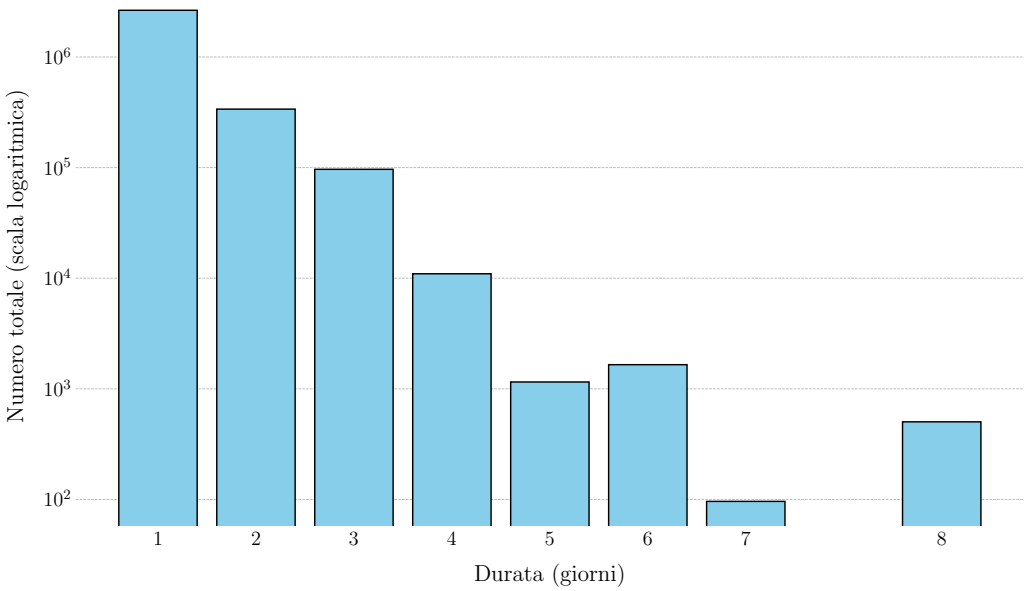

Figura 3.3: Distribuzione della durata dei job in giorni

Tabella 3.1: Percentuale di job in attesa rimossi senza aver effettuato alcun calcolo su un host fisico

Job in attesa rimossi	**Job eseguiti totali**	**Percentuale**
402,663	3,589,280	11.22

3.1.2 Caratterizzazione dei gruppi

L'analisi dei gruppi, come mostrato nella figura 3.6, conferma le osservazioni precedentemente fatte. La distribuzione della durata dei job per gruppo riflette l'andamento già notato nella figura 3.3. In particolare, si nota che con l'aumentare dei giorni, il numero di job che rimangono in esecuzione per quella durata diminuisce esponenzialmente.

La figura 3.7 evidenzia come i gruppi associati agli esperimenti LHC sottomettano un numero significativamente maggiore di job rispetto ai gruppi non-LHC. Questo elevato numero di job, tuttavia, non si traduce in un tasso di fallimento proporzionalmente alto, come osservato in precedenza nella figura 3.4. Questa differenza può essere attribuita ai meccanismi interni di controllo presenti nei gruppi LHC, i quali intervengono rimuovendo

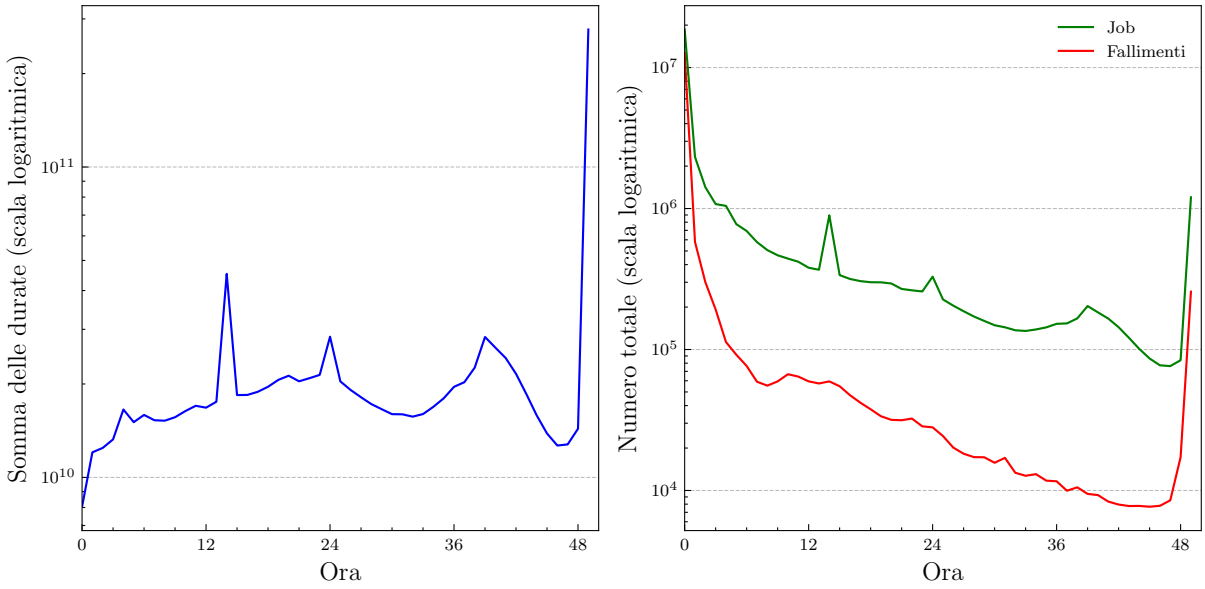

Figura 3.4: A sinistra, durata cumulativa dei job; a destra, numero totale di job e relativi fallimenti per ogni fascia oraria fino a 48 ore, mostrati su scala logaritmica

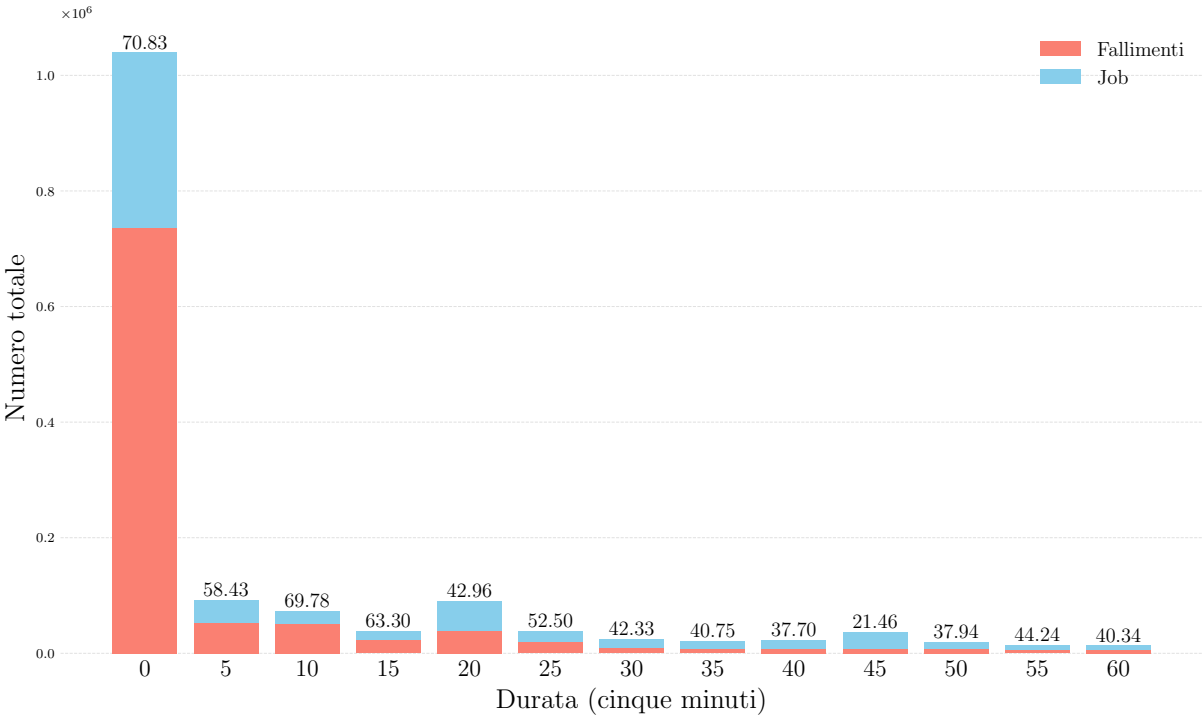

Figura 3.5: Distribuzione dei job nella prima ora, suddivisi in intervalli di cinque minuti

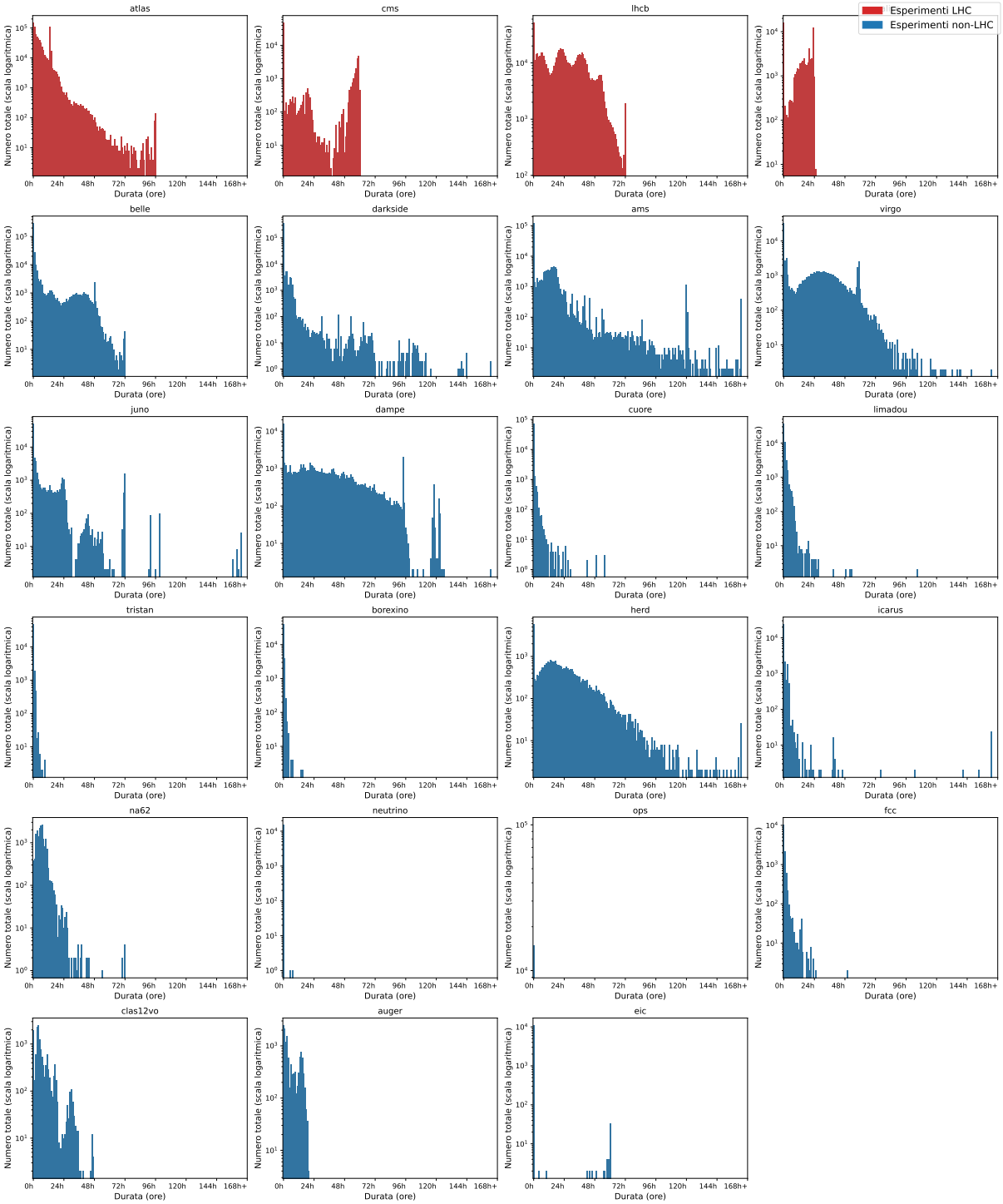

Figura 3.6: Distribuzione della durata dei job per gruppo

autonomamente i job problematici. Al contrario, alcune code non-LHC presentano un tasso di fallimento estremamente alto. Tuttavia, le ragioni specifiche di questi fallimenti rimangono a noi ignote, poiché HTCondor fornisce solo informazioni limitate e generiche sui motivi dei fallimenti.

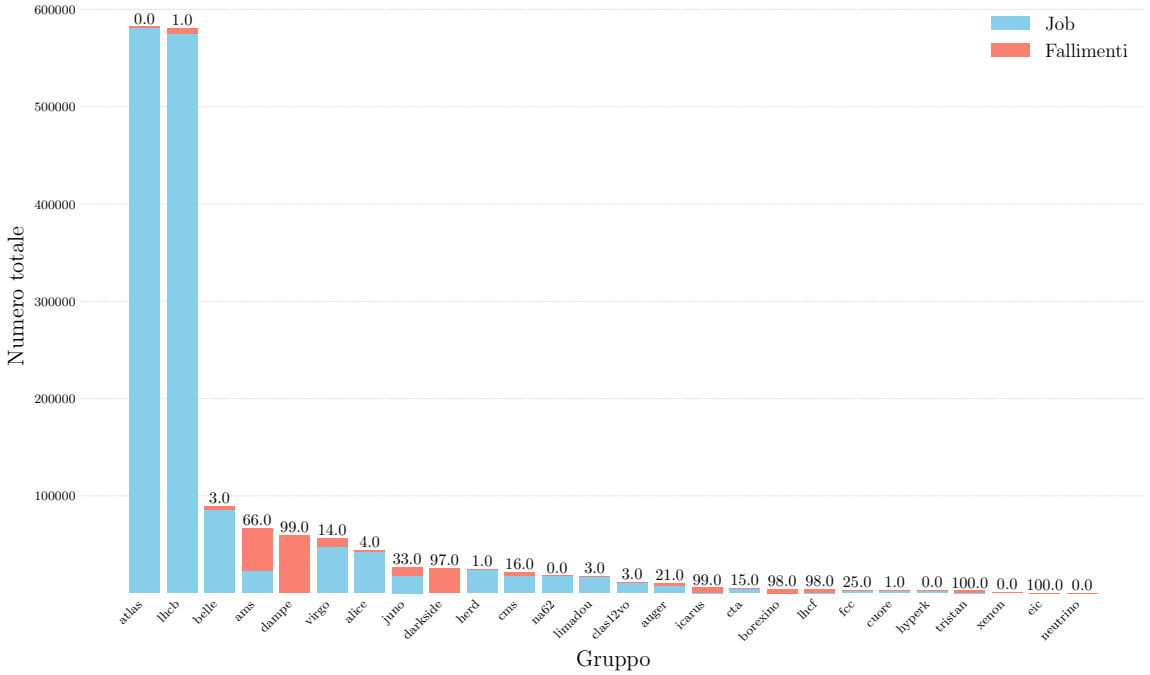

Figura 3.7: Distribuzione del numero totale dei job e dei loro fallimenti per gruppo

3.1.3 Analisi dei nomi dei job

L'analisi semantica dei nomi dei job può arricchire le feature a disposizione di un classificatore e migliorarne le prestazioni [1]. A tal fine, si sono analizzati i nomi dei job, applicando diverse tecniche di pulizia del testo per identificare i termini più frequenti.

La tabella 3.2 mostra i risultati dell'analisi semantica effettuata sui nomi dei job. Questa ha permesso di identificare i cosiddetti job `pilot`, i quali sono progettati per insediarsi in un host fisico e eseguire altri job, detti `payload`. Questi job non sono direttamente coinvolti nei calcoli, ma piuttosto servono a scandagliare le risorse disponibili alla ricerca di un host fisico.

Tabella 3.2: Frequenza dei nomi dei job

Nome	Frequenza
dirac	142868
pilotwrapper	142868
script	72209
eposlhc	70736
p5600	42766
p100	27823
alessr	21568
bi210	7544
pileup	2522
bi214	754

Tuttavia, l'analisi non ha fornito i risultati aspettati. Invece di rilevare diverse tipologie di job, i nomi tendono spesso a riflettere l'esperimento scientifico a cui sono associati, informazione meno utile a comprendere le specifiche funzioni dei job.

3.2 Job Zombie Prediction

Dall'idea delineata nella sezione 3.1.1, siamo interessati a prevedere il fallimento di quei job che garantiscono il maggior payoff. Il payoff è il beneficio ottenuto, in termini di risparmio di risorse, interrompendo tempestivamente un job che è destinato a fallire. Rappresentando il concetto di payoff associato ai job, la figura 3.8 classifica i job in base alla durata, suddividendoli in tre categorie: corte, medi e lunghi. È chiaro , come già detto, che interrompere job lunghi sia più significativo. In questa sezione introduciamo un sottotipo di job lunghi, i **job zombie**, la cui interruzione ci può garantire il MASSIMO payoff.

I job zombie sono job che, sebbene ancora in esecuzione, si sono "bloccati" a un certo punto della loro attività, cessando di svolgere calcoli utili. Questi job entrano in uno stato di "coma", incapaci di terminare autonomamente la loro esecuzione e continuano

Figura 3.8: Rappresentazione dei job in base alla loro durata

a occupare inutilmente risorse di calcolo finché, una volta raggiunto il limite massimo di tempo di esecuzione impostato dal sistema batch, si attiva un evento di timeout che comporta la rimozione dei job dal sistema. In HTCondor, questo limite è attualmente fissato a 3 giorni per i job di tipo `grid` e a 7 giorni per i job di tipo `local`. I job `local`, sottomessi all'interno della stessa "farm" di calcolo tramite il nodo "sn-02" da utenti che fanno parte dell'organizzazione, beneficiano di maggiore libertà. Al contrario, i job `grid` vengono sottomessi tramite nodi, identificati come "ce0x-htc", accessibili agli utenti esterni all'organizzazione.

Tabella 3.3: Rapporto tra i job zombie e il totale dei job per ciascun gruppo

Gruppo	**Job Zombie**	**Job totali**	**Percentuale**	**Giorni di calcolo persi**
LHCb	192	262,251	0.073%	576
JUNO	151	10,137	1.49%	453
ATLAS	45	270,086	0.017%	135
LHCf	8	1,594	0.502%	24
Belle	2	42,087	0.005%	6

La tabella 3.3 mostra un totale di 1,194 giorni di calcolo persi a causa di job zombie, sottolineando l'entità del problema.

> *Un classificatore progettato per identificare precocemente i job zombie potrebbe liberare risorse occupate inutilmente da tali job, consentendo a nuovi job, potenzialmente produttivi, di iniziare l'esecuzione. Ciò non solo ridurrebbe lo spreco di risorse, ma aumenterebbe il throughput del centro di calcolo.*

Purtroppo, poiché HTCondor registra solamente i nuovi massimi nel consumo di risorse e non tiene traccia dei decrementi, un job zombie potrebbe non utilizzare più risorse, ma

apparire come se le stesse ancora utilizzando. Questi job possono quindi nascondersi dietro ad altri job che sono in esecuzione e che stanno utilizzando le risorse, rendendo difficile la loro identificazione. Questa problematica è confermata nella figura 3.9: sebbene si osservi il consumo di RAM, SWAP e disco, la mancata registrazione dei decrementi da parte di HTCondor non permette di distinguere i job effettivamente in esecuzione da quelli zombie.

Un altro problema riguarda la rarità dei job zombie; ad esempio, considerando i dati di marzo 2023, su un totale di 950,558 job, solo 441 sono job zombie, corrispondendo a una percentuale dell'appena dello 0,046%. Questo sbilanciamento può comportare difficoltà nell'addestramento di classificatori robusti e porta anche alla necessità di un'estesa raccolta di dati nel tempo.

Per stabilire se l'applicazione di tecniche di Machine Learning è fattibile nel rilevare i cosiddetti "job zombie", è essenziale verificare preliminarmente la presenza di cluster ben definiti all'interno del dataset. L'obiettivo sarebbe quello di distinguere con precisione questi job anomali dagli altri.

L'algoritmo t-Distribuited Stochastic Neighbor Embedding (t-SNE) consente di ridurre la dimensionalità dei dati preservando la vicinanza tra i punti simili e distanziando quelli dissimili [8]. Passando da uno spazio ad alta dimensionalità a uno spazio bidimensionale o tridimensionale è possibile la visualizzazione dei dati attraverso uno scatterplot.

Tuttavia, t-SNE può essere impraticabile con dataset di grandi dimensioni a causa della sua complessità computazionale dell'ordine di $\mathcal{O}(N^2)$ [10], dove N rappresenta il numero delle istanze. Data la rarità dei job zombie, è necessario selezionare un ampio periodo di monitoraggio per raccogliere un campione sufficiente di tali anomalie. Di conseguenza, si accumula un gran numero di istanze, complicando l'uso di t-SNE.

Per ovviare a questo problema, si può ricorrere a un'**autoencoder**, un tipo specifico di rete neurale progettata per comprimere i dati in una rappresentazione a dimensionalità ridotta per poi ricostruire un output il più possibile simile all'input originale. Come illustrato in figura 3.10, un autoencoder è composto da due parti: una funzione di codifica (*encoder*) che trasforma l'input in una rappresentazione compressa ($h = f(x)$) e una funzione di decodifica (*decoder*) che ricostruisce l'input a partire dalla rappresentazione compressa ($r = g(h)$). L'obiettivo è che la rete impari una funzione $g(f(x))$ che non restituisca x ma piuttosto una rappresentazione semplificata dell'input [7].

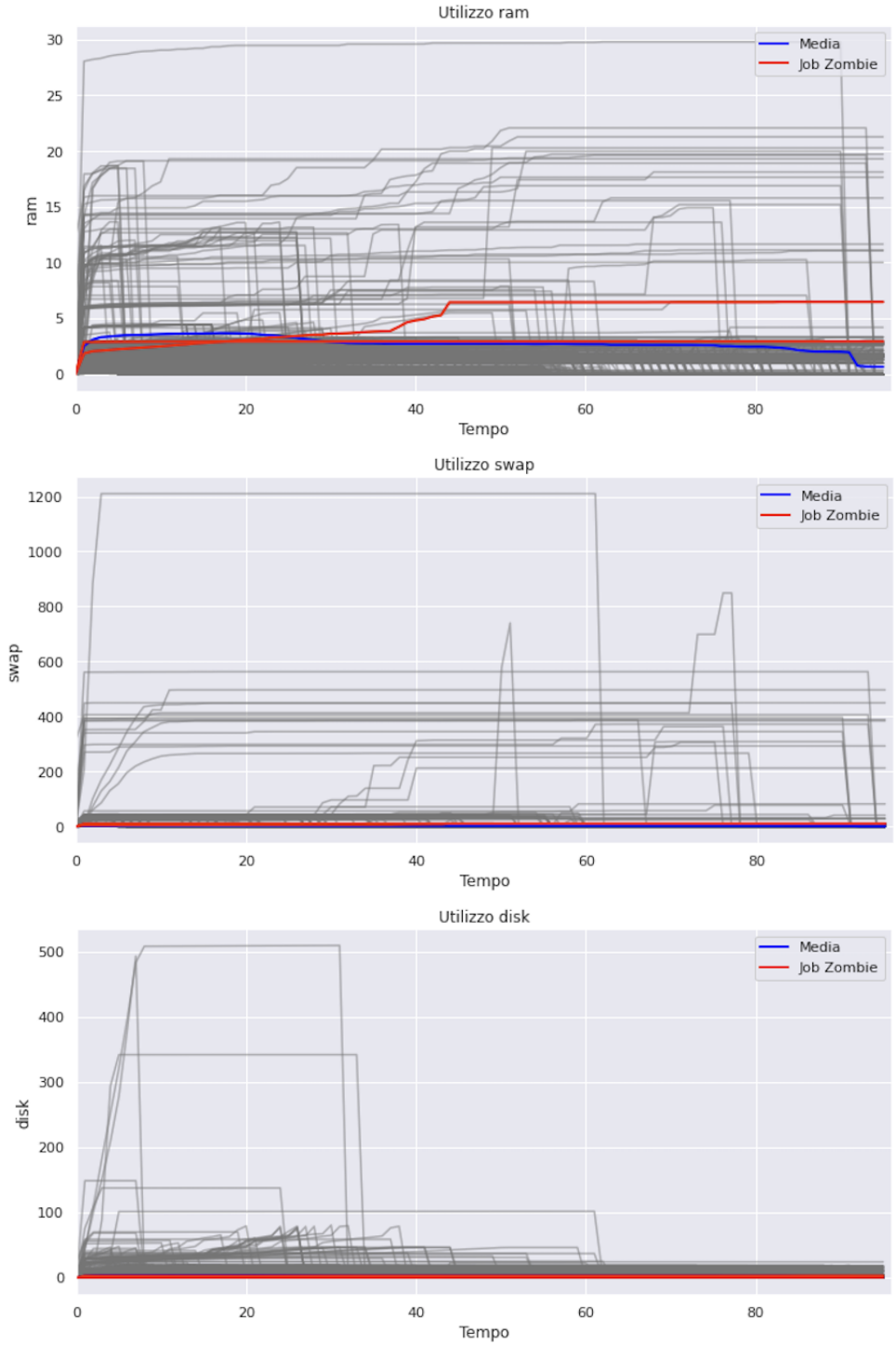

Figura 3.9: Utilizzo di RAM, SWAP e disco su intervalli di 15 minuti nelle prime 24 ore

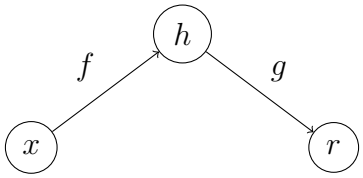

Figura 3.10: Struttura generica di un autoencoder [7]

Se i dati sono caratterizzati da relazioni non lineari, gli autoencoder con funzioni di codifica e decodifica non lineari sono in grado di modellare e preservare tali relazioni anche dopo la compressione dei dati in uno spazio a dimensione ridotta [7].

La figura 3.11 mostra come, l'applicazione di autoencoder per la compressione dei dati, seguita dall'analisi t-SNE, ha permesso l'identificazione di cluster distinti di job zombie accumulatisi nel 2021. È tuttavia importante prestare attenzione all'interpretazione dei risultati ottenuti con t-SNE, infatti le distanze tra i cluster o le loro dimensioni apparenti non sono significative [15]. In aggiunta, la composizione di questi cluster risulta indipendente dal gruppo che ha sottomesso i job. La prevalenza di job provenienti dagli esperimenti LHCb e ATLAS non riflette altro che l'abbondante sottomissione di job da parte degli esperimenti LHC.

Ulteriormente, esaminando i job relativi agli esperimenti LHC, in particolare quelli relativi ad ATLAS, durante la seconda metà di settembre 2021, si è notato che alcuni job zombie si raggruppano in cluster distinti, come illustrato nella figura 3.12. Questo suggerisce che questi job potrebbero essere identificati con maggiore facilità. D'altra parte, vi sono job che si mescolano tra quelli normali, il che potrebbe rendere la loro individuazione più complessa.

In conclusione,

> *l'obiettivo di questa tesi è verificare che i job zombie possano essere identificati attraverso l'applicazione di modelli di Machine Learning.*

Nel capitolo 4, esploreremo due diversi modi di modellare il problema con il Machine Learning, applicando tecniche specifiche per ciascuno. Successivamente, nel capitolo 5, valuteremo e confronteremo i risultati ottenuti attraverso i due approcci e le relative tecniche utilizzate.

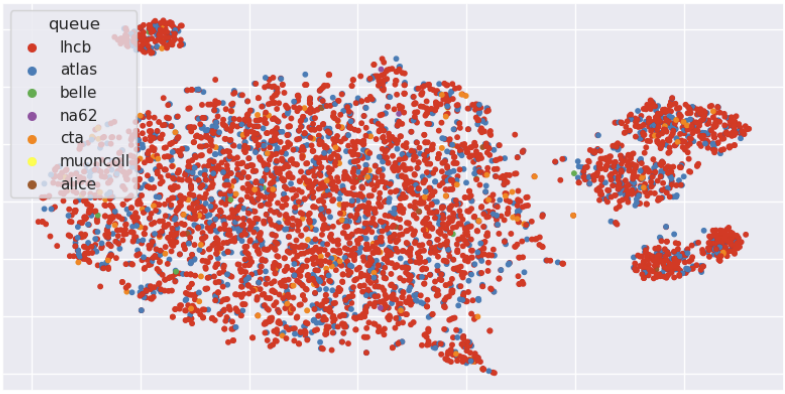

Figura 3.11: Visualizzazione job zombie del 2021 tramite t-SNE

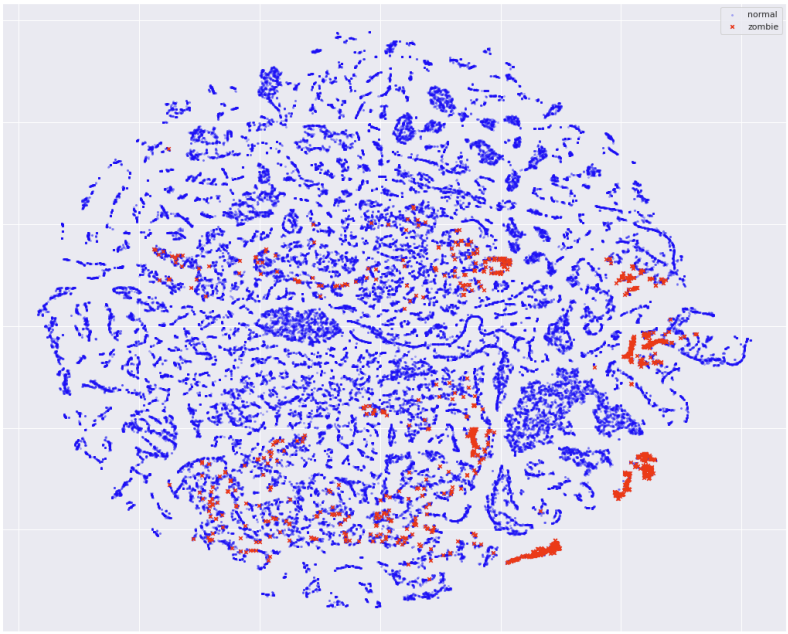

Figura 3.12: Visualizzazione job ATLAS di settembre 2021 tramite t-SNE

Capitolo 4

Applicazioni delle tecniche di Machine Learning

Il workflow del machine learning è produrre dei modelli di machine learning addestrati sui dati che possono. Prima di parlare di come si crea e che modelli di machine learning andremo a creare, parleremo di come automatizzare la creazione di questi modelli. However, the real challenge isn't building an ML model, the challenge is building an integrated ML system and to continuously operate it in production.

create

In questo capitolo prima di parlare degli algoritmi di Machine Learning utilizzati, spiegheremo come arrivare a poter ottenere un modello, e ciò verrà affatto attraverso la creazione di una pipeline.

Una pipeline consiste in una sequenza di più step indipendenti che sono linkati tra di loro per risolvere un task. Le pipeline nel machine learning consistono in step multipli sequenziali che fanno tutto da estrarre i dati a preprocessarli per l'addestramento del modello, al deployment e la sua valutazione. Possiamo vedere una pipeline come un DAG (Directed Acyclic Graph), dove i dati fluiscono in un solo senso. Acyclic perché non sono permessi loop e dove ogni nodo corrisponde a uno step di Machine Learning

The four main steps in an ML pipeline are data preparation, model training, model deployment,

linkando insieme diversi step insieme noi possiamo automatizzare il workflow che serve

per produrre un modello di Machine Learning. Ciò è importante, perché se pensiamo di avere tanti step di trasformazioni, e a ogni set di dati che utilizziamo bisogna applicare le stesse trasformazioni, avendo incapsulato in un'unica entità che utilizziamo possiamo salvare tempo e effort.

Rapid experiment: The steps of the ML experiment are orchestrated. The transition between steps is automated, which leads to rapid iteration of experiments and better readiness to move the whole pipeline to production.

Modularized code for components and pipelines: To construct ML pipelines, components need to be reusable, composable, and potentially shareable across ML pipelines. Therefore, while the EDA code can still live in notebooks, the source code for components must be modularized. I

Nelle prossime sezioni parleremo di tutti gli step legati all'estrazione dei dati, alla loro Preprocessing, al model building, e come in ogni step è stato osservato come poter integrare

Nelle prossime sezioni parleremo di tutti gli step legati a un workflow di machine learning e di come è stato osservato un occhio di riguardo rispetto alla produzione della pipeline.

Ometteremo solo di trattare lo step iniziale del Data collection, ovvero raccogliere i dati grezzi da diverse fonti di dati e immagazzinarle, poiché è stato parte del lavoro iniziale del Dott. Stefano Dal Pra che tramite un cron e diversi script ha collezionato i dati grezzi dalle diverse fonti come dai file history Stefano Dal Pra e del monitoraggio dei job da HtCondor e non riguarda questa trattazione.

4.1 Estrazione dei dati

Data extraction: You select and integrate the relevant data from various data sources for the ML task. We cannot directly input the collected data to train the model without pr-processing it, as it may generate an abrupt result.

Procediamo con l'estrazione di un dataset tramite la query SQL che interroga le tabelle `hj` e `htjob`. Ricordando che:

- La tabella `hj` contiene lo stato dei job, rappresentato da serie storiche di misurazioni (come `runtime`, `ram`, `swap`, `disk`, ecc.), durante la loro esecuzione.
- La tabella `htjob` fornisce informazioni sull'esito dei job, ovvero se sono falliti o meno.

La query esegue le seguenti operazioni:

- Seleziona i job che hanno iniziato e finito la loro esecuzione nel periodo temporale specificato.
- Esegue un JOIN delle tabelle utilizzando l'identificativo univoco di ciascun job (*jobid.idx_submitnode*) e il timestamp. Questo timestamp sfrutta l'indice presente nella tabella `hj` per gestire in maniera efficiente le grandi dimensioni di questa tabella[1]. Poiché la tabella `hj` contiene più record per ogni job, la query li raggruppa per job. In seguito, mediante l'uso dell'operatore `ARRAY_AGG`, le serie storiche vengono trasformate in liste di valori.
- Filtra i job con un tempo di esecuzione superiore a un'ora (`runtime > 3600`), in quanto i job più brevi sono considerati irrilevanti per lo scopo dello studio.

Il risultato di questa query è un dataset come mostrato nella figura 4.2, dove ogni riga rappresenta un job e le colonne includono:

- `job`: identificativo univoco per ogni job.
- `queue`: gruppo di appartenenza dell'utente che ha sottomesso il job.
- `fail`: una variabile booleana che indica se il job è fallito.
- `mint` e `maxt`: il tempo minimo e massimo di esecuzione del job.
- `t`, `ram`, `swap`, `disk`: liste di valori che rappresentano le serie storiche di misurazioni.

[1] La logica dietro questo consiste nel selezionare un job da `htjob` e successivamente cercarlo in `hj` limitando la ricerca ai record che rientrano nel periodo in cui `hj.ts` è compreso tra `htjob.starttimeepoch` e `htjob.eventtimeepoch`. Questo permette di restringere notevolmente la ricerca nella tabella `hj` per ogni job selezionato da `htjob` ed evitare di scansionare l'intera tabella.

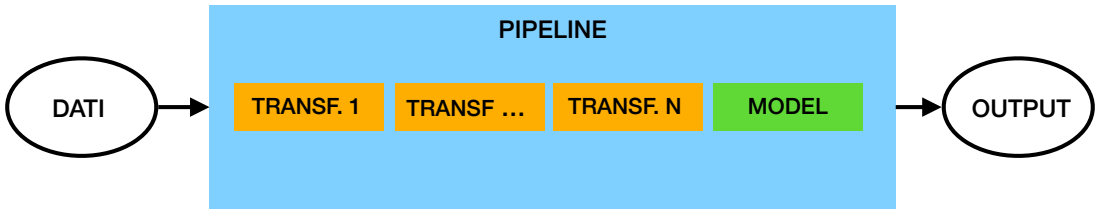

Figura 4.1

Il dataset ottenuto comunque non è sufficiente per poter applicare direttamente un algoritmo di Machine Learning, infatti le colonne che rappresentano le serie storiche univariate contengono un tipo di dato complesso, come delle liste. Questo ha bisogno di essere preprocessato e aggiungere ulteriori informazioni che si possono astrarre dai dati e rendere i dati utilizzabili da un algoritmo. Dobbiamo risolvere tutti i problemi spiegati nella sezione delle serie storiche 4.1.

	job	queue	fail	mint	maxt	t	ram	swap	disk
0	9242718.0_ce04-htc	juno	0	1678695481	1678744082	[80, 80, 80, 80, 261, 261, 261, 261, 440, 440,...	[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, ...	[7e-06, 7e-06, 7e-06, 7e-06, 7e-06, 7e-06, 7e-...	[7e-06, 7e-06, 7e-06, 7e-06, 7e-06, 7e-06, 7e-...
1	9242720.0_ce04-htc	juno	0	1678695481	1678721221	[80, 80, 80, 80, 260, 260, 260, 260, 440, 440,...	[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, ...	[7e-06, 7e-06, 7e-06, 7e-06, 7e-06, 7e-06, 7e-...	[7e-06, 7e-06, 7e-06, 7e-06, 7e-06, 7e-06, 7e-...
2	9242734.0_ce04-htc	juno	0	1678695481	1678747322	[79, 79, 79, 79, 260, 260, 260, 260, 439, 439,...	[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, ...	[7e-06, 7e-06, 7e-06, 7e-06, 7e-06, 7e-06, 7e-...	[7e-06, 7e-06, 7e-06, 7e-06, 7e-06, 7e-06, 7e-...
3	9242733.0_ce04-htc	juno	0	1678695481	1678742461	[79, 79, 79, 79, 260, 260, 260, 260, 439, 439,...	[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, ...	[7e-06, 7e-06, 7e-06, 7e-06, 7e-06, 7e-06, 7e-...	[7e-06, 7e-06, 7e-06, 7e-06, 7e-06, 7e-06, 7e-...
4	9242731.0_ce04-htc	juno	0	1678695481	1678710781	[79, 79, 79, 79, 260, 260, 260, 260, 439, 439,...	[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, ...	[7e-06, 7e-06, 7e-06, 7e-06, 7e-06, 7e-06, 7e-...	[7e-06, 7e-06, 7e-06, 7e-06, 7e-06, 7e-06, 7e-...

Figura 4.2: Le prime cinque righe del dataset

4.2 Preparazione dei dati

Gli algoritmi di Machine Learning richiedono che i dati siano dei numeri.

Inoltre, Se la qualità o la quantità dei dati sono insufficienti, noi andremo incontro a una situazione comunemente chiamata “garbage in, garbage out” e non importa quanto

sia figo il nostro algoritmo di machine learning, perché non avremo risultati soddisfacenti. Infatti più è fancier l'algoritmo, più dati avremo bisogno.

Data l'esistenza di dati di bassa qualità, per aumentare la qualità del modello che vogliamo utilizzare, i dati DEVONO ESSERE PREPARATI.

Rumore nei dati e errori possono essere corretti Complex nonlinear relationships may be teased out of the data. r how to best expose the underlying structure of the problem to the learning algorithms. This is the guiding light.

4.2.1 Trasformazione delle serie storiche multivariate multiple

4.2.2 Creazione delle feature

4.2.3 Etichettatura dei dati

4.2.4 Tecniche di bilanciamento dei dati

4.3 Selezioni dei modelli

4.3.1 Modelli supervisionati

4.3.2 Modelli non supervisionati

4.4 Valutazione delle performance

4.4.1 Metriche di valutazione

4.4.2 Convalida incrociata

Capitolo 5

Analisi dei risultati

5.1 Confronto tra i modelli

5.2 Interpretazione dei risultati

Capitolo 6

Conclusioni e sviluppi futuri

6.1 Sintesi dei risultati

6.2 Limitazioni dello studio e proposte per ricerche future

Bibliografia

[1] Anupong Banjongkan et al. «A Study of Job Failure Prediction at Job Submit-State and Job Start-State in High-Performance Computing System: Using Decision Tree Algorithms». In: *Journal of Advances in Information Technology* 12 (2021), pp. 84–92. URL: https://api.semanticscholar.org/CorpusID:234326555.

[2] G Bortolotti et al. «The INFN Tier-1». In: *Journal of Physics: Conference Series* 396.4 (dic. 2012), p. 042016. DOI: 10.1088/1742-6596/396/4/042016. URL: https://dx.doi.org/10.1088/1742-6596/396/4/042016.

[3] Franck Cappello et al. «Toward Exascale Resilience: 2014 update». In: *Supercomputing Frontiers and Innovations* 1.1 (giu. 2014), pp. 5–28. DOI: 10.14529/jsfi140101. URL: https://superfri.org/index.php/superfri/article/view/14.

[4] CERN. *Worldwide LHC Computing Grid*. 2023. URL: https://wlcg.web.cern.ch (visitato il 28/10/2023).

[5] CNAF. *WLCG Tier-1 data center - Calcolo*. URL: https://www.cnaf.infn.it/calcolo/ (visitato il 28/10/2023).

[6] Stefano Dal Pra et al. «Evolution of monitoring, accounting and alerting services at INFN-CNAF Tier-1». In: *EPJ Web of Conferences* 214 (gen. 2019), p. 08033. DOI: 10.1051/epjconf/201921408033.

[7] Ian Goodfellow, Yoshua Bengio e Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

[8] Laurens van der Maaten e Geoffrey Hinton. «Visualizing Data using t-SNE». In: *Journal of Machine Learning Research* 9 (2008), pp. 2579–2605. URL: https://www.jmlr.org/papers/volume9/vandermaaten08a/vandermaaten08a.pdf.

[9] Gordon E. Moore. «Cramming more components onto integrated circuits». In: *Electronics* 38.8 (1965), pp. 114–117.

[10] Nicola Pezzotti et al. «Approximated and User Steerable tSNE for Progressive Visual Analytics». In: *IEEE Transactions on Visualization and Computer Graphics* 23.7 (2017), pp. 1739–1752. DOI: 10.1109/TVCG.2016.2570755.

[11] Andrea Rendina. *INFN-T1 site report*. https://indico.cern.ch/event/1200682/contributions/5087586/attachments/2538178/4368754/20221031_InfnT1_site_report.pdf. Accessed: 2023-10-28. 2022.

[12] J. M. Shalf e R. Leland. «Computing Beyond Moore's Law». In: *Computer* 48.12 (dic. 2015), pp. 14–23. ISSN: 1558-0814. DOI: 10.1109/MC.2015.374.

[13] Thomas N. Theis e H.-S. Philip Wong. «The End of Moore's Law: A New Beginning for Information Technology». In: *Computing in Science & Engineering* 19.2 (2017), pp. 41–50. DOI: 10.1109/MCSE.2017.29.

[14] Oreste Villa et al. «Scaling the Power Wall: A Path to Exascale». In: *SC '14: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. 2014, pp. 830–841. DOI: 10.1109/SC.2014.73.

[15] Martin Wattenberg, Fernanda Viégas e Ian Johnson. «How to Use t-SNE Effectively». In: *Distill* (2016). DOI: 10.23915/distill.00002. URL: http://distill.pub/2016/misread-tsne.