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) [3].

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 quest'ultimi, vengono supportati presso il CNAF anche gli esperimenti non-LHC di astrofisica delle particelle e fisica dei neutrini [1].

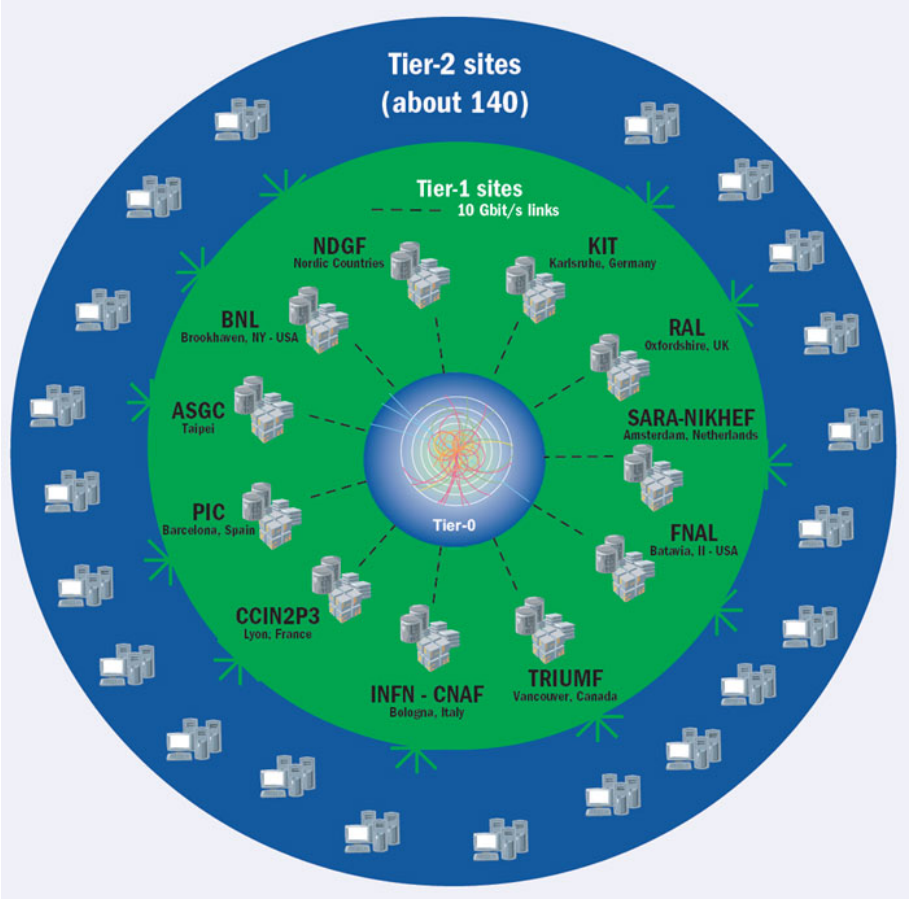

Figura 2.1: Struttura gerarchica del WLCG [5]

Il CNAF offre più di 46000 core distribuiti su 960 host fisici per un totale di circa 630 kHS06[1] di potenza di calcolo [8]. L'allocazione di queste risorse segue il paradigma del **High-Throughput Computing** (HTC), dove a differenza dell'High-Performance Computing, che mira a eseguire calcoli ad alta velocità, l'obiettivo dell'HTC è massimizzare il numero di operazioni compiute su un periodo di tempo prolungato.

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 [4].

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 a scelta in 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 [7]. 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 [9, 10].

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 [11]. 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

Definition 2.3.1. Un *guasto* è 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 [2].

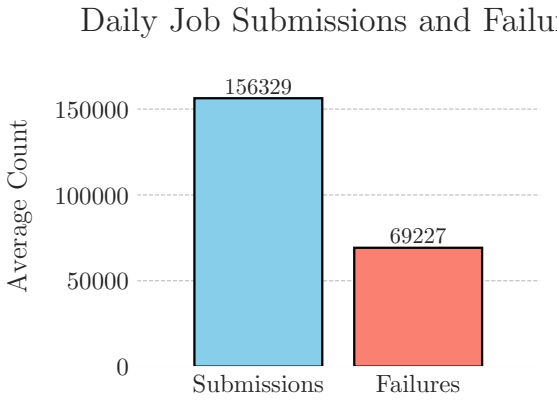

Figura 2.2: Media giornaliera di job sottomessi e falliti nel mese di Marzo 2023

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

3.1 Analisi esplorativa

Definition 3.1.1. Una serie di dati è una sequenza ordinata di punti dati, ed 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.

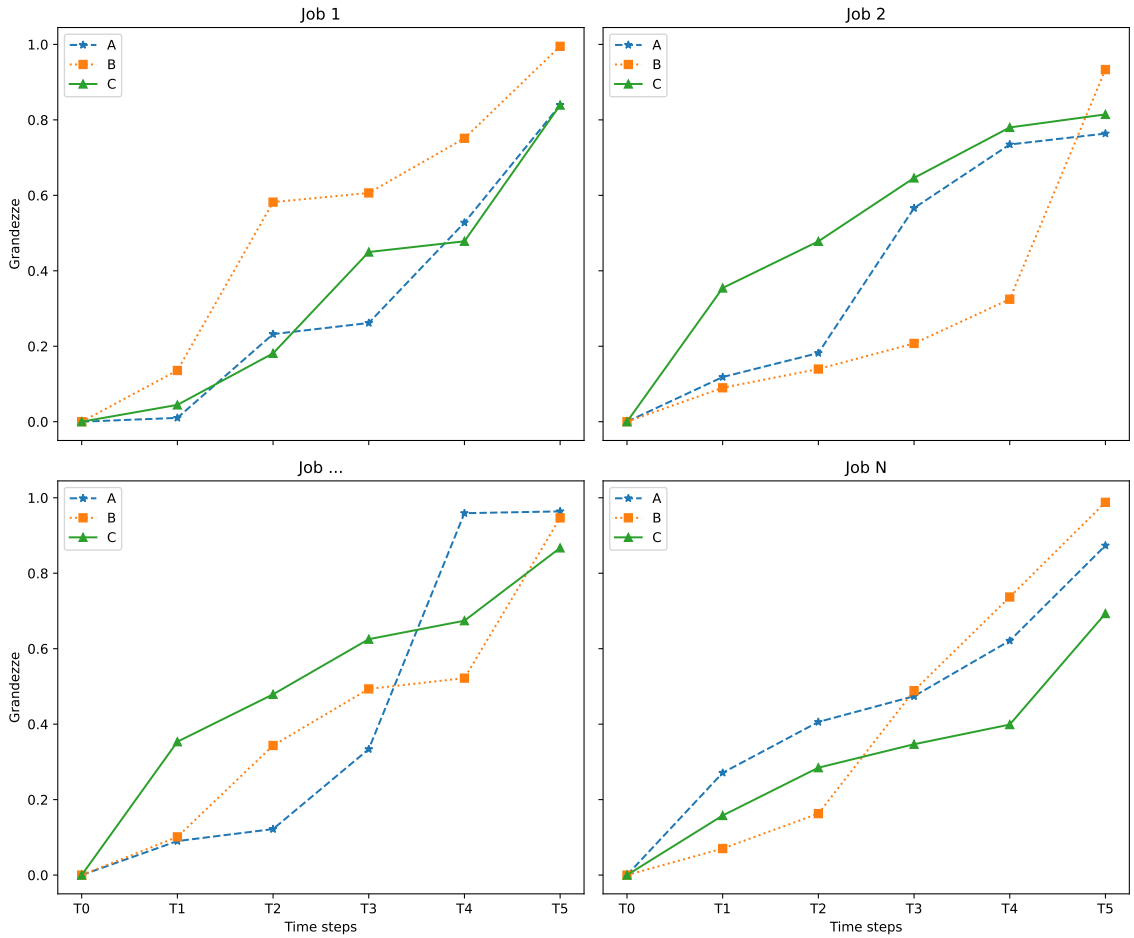

Figura 3.1: Rappresentazione di un job come serie storica multivariata

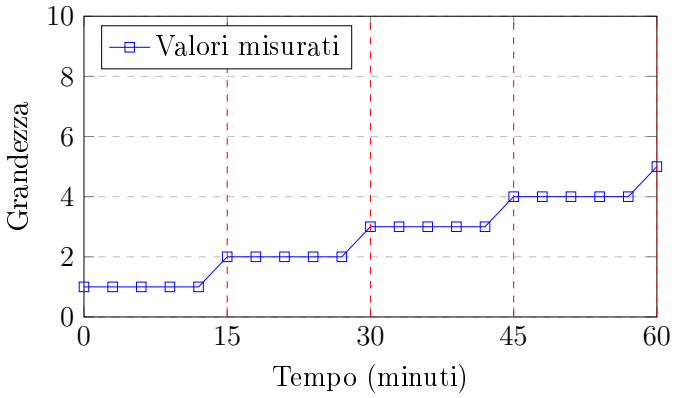

Figura 3.2: Frequenza di campionamento e aggiornamento dei job in HTCondor

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.

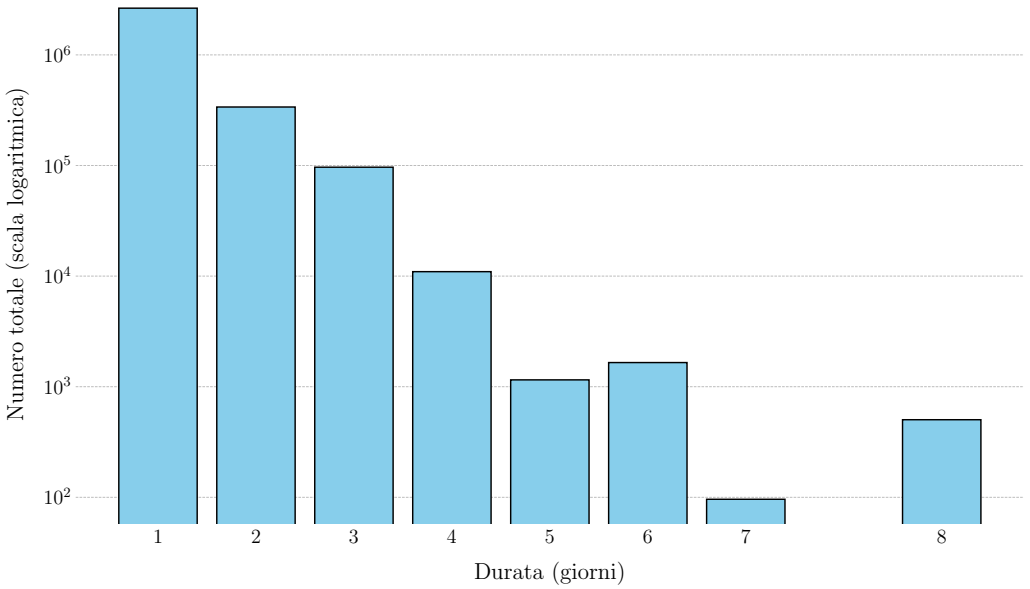

Figura 3.3:

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 esecuzioni 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.

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

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.

3.2 Job Zombie Prediction

Tabella 3.2: Descrizione dei dati di esecuzione dei job

Gruppo	**Job zombie**	**Job totali**	**perc_time_lost**	**Giorni persi**
lhcb	192	262251	0.073212	576
juno	151	10137	1.489593	453
atlas	45	270086	0.016661	135
lhcf	8	1594	0.501882	24
belle	2	42087	0.004752	6

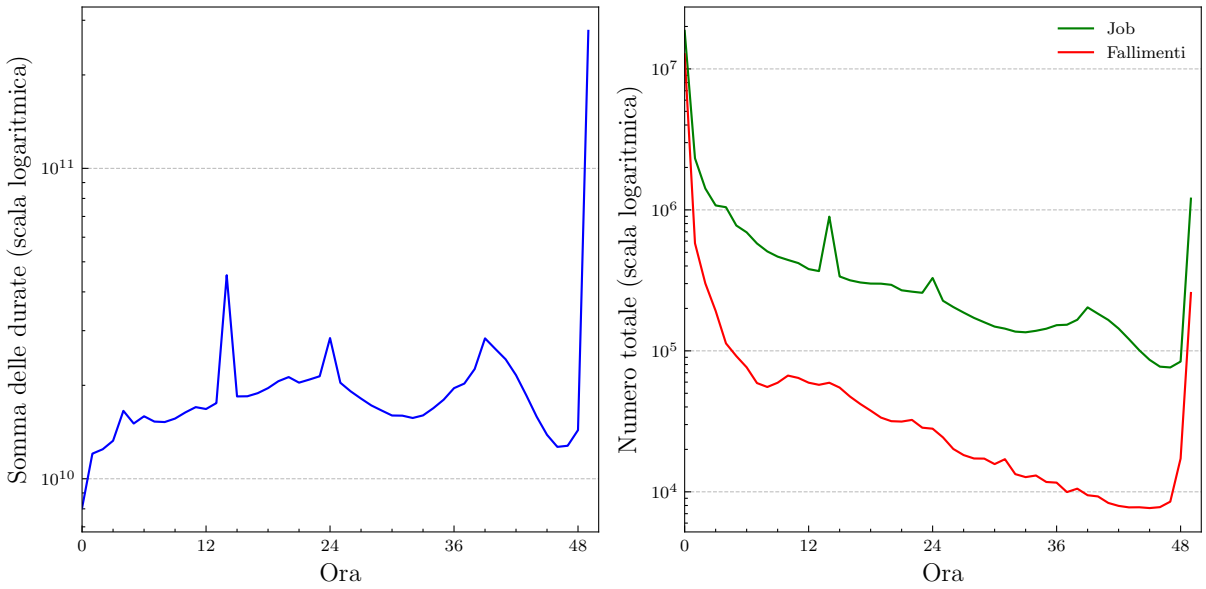

Figura 3.4: A sinistra, durata cumulativa dei job; a destra, numero totale di job e relativi fallimenti per ogni fascia oraria fino a 48 ore, mostrati su scala logaritmica

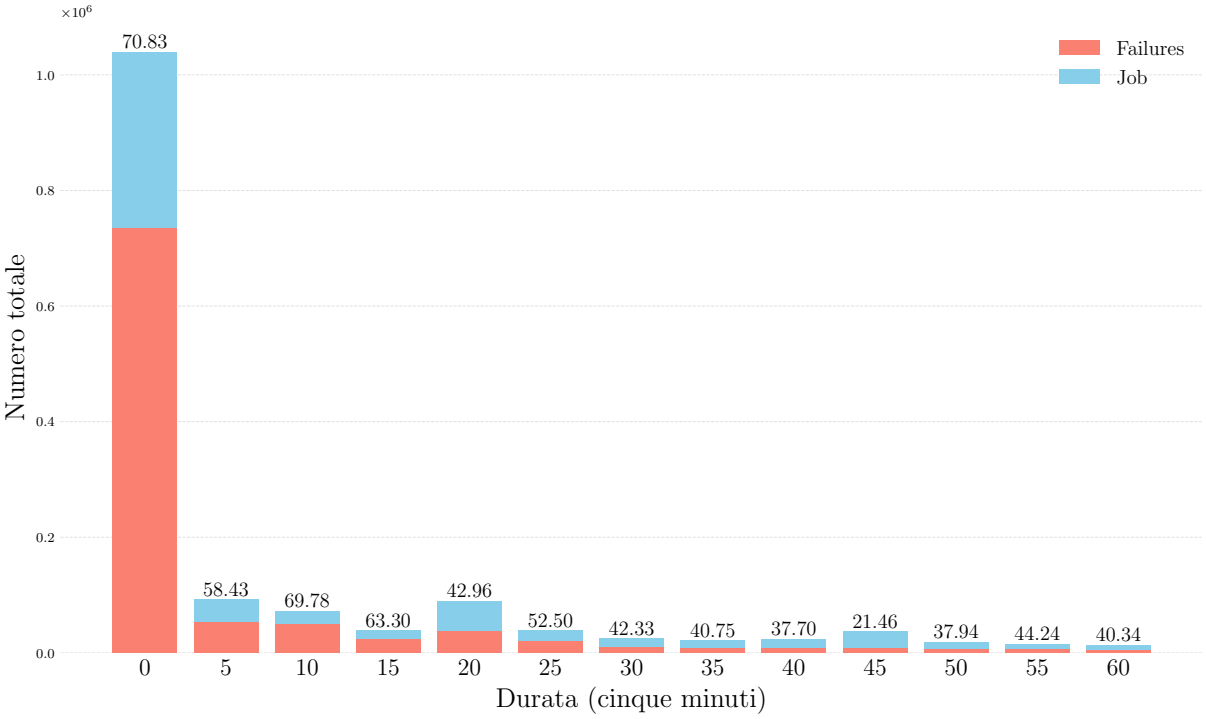

Figura 3.5: Distribuzione dei job nella prima ora, suddivisi in intervalli di cinque minuti

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 [6]. Passando da uno spazio ad alta dimensionalità a uno spazio bidimensionale o tridimensionale è possibile la visualizzazione dei dati attraverso uno scatterplot.

3.3 Preparazione dei dati per il task di ML

3.3.1 Trasformazione delle serie storiche multivariate multiple

3.3.2 Creazione delle feature

3.3.3 Labeling dei dati

3.3.4 Tecniche di bilanciamento dei dati

Capitolo 4

Applicazioni delle tecniche di Machine Learning

4.1 Selezioni dei modelli

4.1.1 Modelli supervisionati

4.1.2 Modelli non supervisionati

4.2 Valutazione delle performance

4.2.1 Metriche di valutazione

4.2.2 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] 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.

[2] 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.

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

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

[5] 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.

[6] Aurélien Géron. *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*. 2ª ed. O'Reilly Media, 2019, p. 233.

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

[8] 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.

[9] 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.

[10] 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.

[11] 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.