UNIVERSITA' DEGLI STUDI DI SALERNO

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE ED
ELETTRICA E MATEMATICA APPLICATA

Project Work System Safety Engineering - Gruppo 1

Nome e Cognome	**Matricola**	**Email**
Beniamino Squitieri	0622702021	b.squitieri@studenti.unisa.it
Francesca Venditti	0622701969	f.venditti1@studenti.unisa.it
Adamo Zito	0622702026	a.zito32@studenti.unisa.it

ANNO ACCADEMICO 2023/2024

Contents

Capitolo 1

1 Introduzione

Il progetto si concentra su applicazioni safety-critical, in cui è fondamentale l'utilizzo di sistemi di comunicazione riconfigurabili in maniera rapida in caso di emergenze. L'architettura basata su Service Function Chain (SFC), ovvero reti di nodi virtualizzati collegati in serie, risultano essere un'ottima soluzione per la realizzazione di sistemi flessibili e adattabili. Consideriamo uno scenario in cui, per motivi di emergenza, è necessaria una rete rapidamente configurabile per il trasferimento dati, ad esempio dati meteorologici, da una o più stazioni di monitoraggio a una centrale operativa, in cui avviene l'effettiva elaborazione dei dati. Il nodo di rete che connette le stazioni di monitoraggio e il centro di elaborazione è un nodo di rete SFC.

1.1 Struttura del sistema

Il sistema è composto da tre sistemi connessi in serie: S1, S2 e S3.

Figure 1: Rappresentazione sistema in esame in serie

1. il ***sistema fault-tolerant S1*** (stazione di monitoraggio) è composto, a sua volta, da due sottosistemi, i cui componenti rilevanti per il modello di disponibilità sono: un sensore a due stati (funzionante e non funzionante) e un elemento di storage. In particolare, un sottosistema funziona se i suoi componenti sensore e storage sono funzionanti, mentre il sistema S1 funziona se almeno uno dei due sottosistemi è funzionante

2. ***nodo di rete SFC (sistema S2)***, potenzialmente ridondato, realizzato in accordo al modello a 5-layer, come rappresentato dalla figura sottostante.

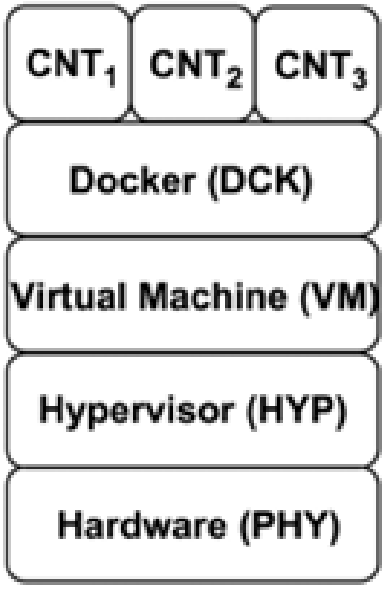

Figure 2: Nodo di rete SFC

Ogni nodo ha 3 container sul livello più alto, e, per il corretto funzionamento del nodo, c'è necessità che almeno un container sia funzionante

3. ***il sistema S3*** è composto da uno o più elaboratori, ognuno caratterizzato dai seguenti componenti:

   - una CPU
   - un elemento di storage
   - un software di analisi dei dati
   - un sistema operativo
   - una parte hardware.

   Si definisce guasto un elaboratore in S3 quando uno dei suoi componenti è guasto, mentre si ritiene l'intero sistema S3 non funzionante quando nessuno degli elaboratori che lo compongono è funzionante

1.2 Service Function Chain (SFC)

Le Service Function Chains (SFC) è un'architettura di rete che consente la definizione e la gestione dei flussi di traffico attraverso una serie di funzioni di servizio specifiche. In una rete SFC, le funzioni di servizio sono organizzate in una catena in cui il traffico di rete passa attraverso ciascuna funzione nella sequenza specificata. Tali reti offrono differenti vantaggi come:

- ***Flessibilità e Adattabilità:*** le reti SFC consentono la definizione dinamica delle catene di servizio, consentendo una rapida riconfigurazione delle funzioni di servizio in risposta alle esigenze specifiche dell'applicazione o a situazioni di emergenza

- ***Ottimizzazione delle risorse:*** L'architettura SFC consente di ottimizzare l'utilizzo delle risorse di rete, in quanto il traffico può essere instradato solo attraverso le funzioni di servizio necessarie. Questo aiuta a evitare l'utilizzo inefficiente delle risorse di elaborazione e larghezza di banda

- ***Gestione della sicurezza:*** Le reti SFC possono essere utilizzate per implementare servizi di sicurezza in modo distribuito lungo il percorso del traffico.

- ***Semplicità di gestione:*** L'architettura SFC semplifica la gestione delle funzioni di servizio , contribuendo a rendere più agevole la configurazione e la manutenzione della rete Quindi gli SFC offrono una struttura flessibile e dinamica che si adatta bene a contesti in cui è richiesta una gestione efficiente e una risposta rapida

1.3 Obiettivo del progetto

L'obiettivo principale del progetto è quello di affrontare diversi aspetti relativi alla valutazione della disponibilità stazionaria dei tre sistemi interconnessi: il sistema fault-tolerant S1, il nodo di rete SFC S2 e il sistema S3. Questa valutazione sarà basata su metodologie stocastiche e includerà la definizione di modelli di disponibilità per ciascun sistema. Inoltre, prevede la presentazione di una procedura di stima di massima verosimiglianza per i tassi di guasto e riparazione del componente hardware (PHY) del singolo nodo SFC. Questo processo sarà basato su dati forniti, inclusi tempo di riparazione e tempi di guasto censurati. Un altro obiettivo significativo è l'individuazione della configurazione ottimale, con opportuna ridondanza, dei tre sistemi (S1-S2-S3) al fine di garantire una disponibilità stazionaria pari o superiore a $A_0 = 0.999999$, considerando i costi specifici per ciascun sistema. Infine, il progetto prevede un'analisi di sensività, esplorando le variazioni nel tasso di guasto e riparazione del componente Docker (DCK) del nodo di rete SFC. Questa analisi mira a identificare valori critici di tali parametri che porterebbero a una diminuzione della disponibilità al di sotto del livello desiderato. Nei capitoli successivi verrano presentate le varie soluzioni proposte in dettaglio.

Capitolo 2

2 Availability

La disponibilità mira a garantire che il sistema sia robusto, accessibile e sicuro nell'uso quotidiano. Questo si traduce nella creazione di prodotti e sistemi che siano affidabili nel lungo termine, minimizzando il rischio di guasti, assicurando un'elevata disponibilità e garantendo la sicurezza degli utenti e dell'ambiente circostante.

Definizione 2.1. *La disponibilità di un sistema può essere formalmente definita come la sua capacità di trovarsi in uno stato idoneo per eseguire una specifica funzione richiesta, in condizioni predeterminate, in un istante temporale definito o dopo che è trascorso un tempo sufficiente*

In questa sezione, vengono fornite le basi teoriche al fine di avere una chiara comprensione delle soluzioni proposte. La disponibilità, o availability, è definita come la probabilità A(t) che un'unità sia operativa al tempo t. Essa è complementare all'unavailability U(t), che rappresenta il sistema nello stato non operativo, dove:

$$A(t) + U(t) = 1 \tag{1}$$

La disponibilità considera sia la reliability, che è la probabilità di funzionamento, sia la maintainability, che è la capacità di riparazione del sistema. La steady-state availability A si considera per $t \to \infty$, ed è data da:

$$A = \frac{MTTF}{MTTF + MTTR} \tag{2}$$

2.1 Models for availability

A questo punto è possibile rappresentare i sistemi di interesse.

> **Nota:** Un modello è un'astrazione matematica del sistema che mira a fornire una rappresentazione concettualizzata e semplificata del comportamento del sistema

I formalismi di modellazione possono essere classificati in:

- ***Non-state-space (or combinatorial) models***: I modelli analitici, utilizzati per affrontare la problematica, offrono soluzioni efficienti grazie alla possibilità di essere trattati analiticamente. Questi modelli sono in grado di catturare in modo efficace le relazioni tra i diversi componenti e le condizioni che possono portare al guasto di un sistema. Tuttavia, una limitazione di tali approcci risiede nella necessità di assumere l'indipendenza stocastica tra i componenti, un'ipotesi che potrebbe non riflettere completamente la complessità delle interazioni nel sistema. Alcuni esempi di questi modelli includono il Reliability Block Diagram (RBD), il Fault-Tree (FT) e i Reliability Graphs.

- ***State-space models:*** Questi modelli, in grado di considerare dipendenze statistiche, di stato e temporali, offrono una notevole potenza di modellazione. Tuttavia, presentano sfide significative dal punto di vista della trattabilità analitica, soprattutto quando lo spazio di stato diventa di dimensioni elevate. I metodi basati sullo spazio di stato sono generalmente classificati in Markoviani (omogenei e non omogenei) e non Markoviani. Nel dettaglio, i modelli Markoviani si suddividono ulteriormente in tempo-discreto e tempo-continuo. Nella nostra analisi, adotteremo approcci omogenei, come le Continuous Time Markov Chain (CTMC) e le Stochastic Reward Net, quest'ultime rappresentanti un caso particolare delle Reti di Petri Stocastiche. Questa scelta è motivata dalla loro capacità di modellare efficacemente i sistemi.

- ***Multi-level models:*** Questi approcci combinano la robustezza di modelli nello spazio di stato con l'efficienza tipica dei modelli al di fuori di tale contesto.

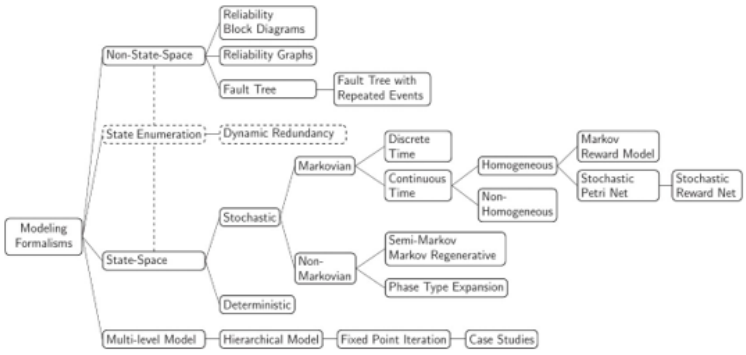

Figure 3: Rappresentazione schematica delle possibili tecniche di rappresentazione di un sistema.

Fault Tree Analysis

Con il termine Fault Tree si indica una tecnica grafica per l'analisi dell'affidabilità e della sicurezza di sistemi complessi (nota anche come FTA) che nasce negli anni '60 del secolo scorso. Tale approccio viene affrontato nella norma IEC 61025. La costruzione di un fault tree si basa su una logica che prende come punto di partenza una condizione di guasto (più in generale una condizione di malfunzionamento) del sistema, denominata Top Event, che viene passo dopo passo scomposta attraverso alcune combinazioni o sequenze di eventi che sono all'origine di quelli considerati nel passo precedente, dando origine quindi ad un grafico che assume l'aspetto di un albero. L'obiettivo di tale metodo è quindi, quello di individuare le cause originarie di un evento indesiderato e di dare loro un ordine gerarchico, secondo una scala di complessità strutturale decrescente.

Partendo dal Top Event, la scomposizione in eventi di livello gerarchico inferiore viene effettuata sulla base di relazioni logiche tra eventi, rappresentato dalle porte AND e OR. In particolare, l'output event (fault) da una porta AND è la conseguenza del verificarsi contemporaneo degli eventi entrati in quella porta, mentre l'evento di uscita OR è la conseguenza di almeno uno degli eventi entrati in quella porta. Questa tipologia tecnica viene usata come analisi qualitativa, per fornire l'espressiona logica del Top Event in funzione di alcuni eventi base. L'analisi qualitativa presenta tutte le modalità di guasto la cui combinazione determina il verificarsi del Top Event e individua i potenziali punti deboli della struttura. Tuttavia, FTA può essere utilizzata anche per effettuare direttamente valutazioni quantitative delle probabilità di verificarsi del Top Event, partendo della probabilità di verificarsi degli eventi al livello gerarchico più basso, oppure può essere utilizzata come base per l'applicazione di altri metodi quantitativi (come ad esmepio il metodo di simulazione Monte Carlo).

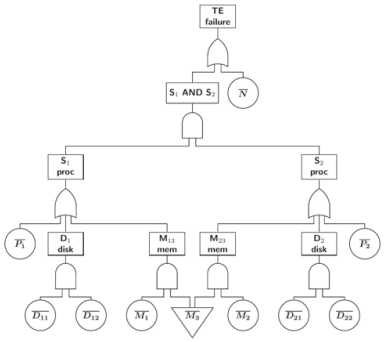

Figure 4: Esempio di Fault Tree Analysis.

2.1.1 Continous Time Markov Chain

Le Continuous-Time Markov Chains (CTMC) rappresentano uno strumento potente e flessibile nell'ambito dell'analisi di sistemi dinamici e processi stocastici. Questo framework matematico permette di modellare il comportamento di sistemi che evolvono nel tempo, con un'enfasi particolare sulle transizioni di stato che avvengono in maniera continua. È possibile analizzare un sistema individuando i diversi stati che lo caratterizzano, ad esempio, in prima analisi è possibile descrivere un qualsiasi sistema attraverso una CTMC a due stati: Up e Down.

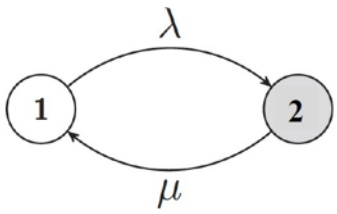

Figure 5: CTMC a due stati.

Modellando il sistema tramite la CTMC in Figura 5, i tassi (distiribuiti in accordo ad una distribuzione esponenziale) che caratterizzano le transizioni assumono uno specifico significato:

- λ: rappresenta il failure rate
- μ : rappresenta il repair rate

A differenza della FTA, avendo ora acquisito il concetto di "stato" è possibile rappresentare un'unità che può essere non disponibile per un certo tempo e poi riparata. L'availability A(t) può essere espressa come la probabilità che il sistema sia attivo in un dato istante di tempo, indipendentemente dal numero di guasti e ripristini che ha subito prima del tempo t, e rappresenta quindi una probabilità di stato. In una CTMC come quella in figura 5 l'availability può essere vista come la probabilità che il sistema si trovi nello stato 1. A partire dalla matrice di transizione nelle condizioni di regolarità, in una CTMC omogenea, le probabilità di stato tendono asintoticamente (t $\rightarrow \infty$) a valori costanti e indipendenti dallo stato iniziale. Tale vettore di probabilità è detto distribuzione limite e può essere calcolato risolvendo il sistema:

$$\begin{pmatrix} P_1(\infty) & P_2(\infty) \end{pmatrix} \begin{pmatrix} 1-\lambda dt & \lambda dt \\ \mu dt & 1-\mu dt \end{pmatrix} = \begin{pmatrix} P_1(\infty) & P_2(\infty) \end{pmatrix} \tag{3}$$

In particolare, essendo le due equazioni linearmente dipendenti è bene tenere in considerazione una ulteriore relazione, ovvero $P1(\infty) + P2(\infty) = 1$
Tramite questo approccio è dunque possibile modellare sistemi in continua evoluzione nel tempo in maniera molto più efficace rispetto alla FTA. Tuttavia, i modelli CTMC presentano restrizioni in termini di applicabilità ad alcuni problemi del mondo reale, dove i tempi di soggiorno negli stati non si adattano a una distribuzione esponenziale. Per aggirare queste tipologie di problemi si fa affidamento ad alcune tipologie di modelli come:

- semi-Markov models
- Markov Regenerative models

2.1.2 Stochastic Reward Network

Una Petri Networks (PN) è un grafo diretto bipartito i cui nodi sono divisi in due insiemi disgiunti chiamati posti e transizioni collegate da archi. Un posto può contenere uno o più token che specificano una determinata condizione della rete, detta marcatura. L'attivazione della transizione è un'azione atomica in cui un token viene rimosso da ciascun posto di input della transizione e viene aggiunto a ciascun posto di output della stessa. Formalmente, una PN è una quintupla (P, T, I, O, M) dove:

- $P = \{p_1, p_2, \ldots, p_{n_p}\}$ è l'insieme degli n_p posti
- $T = \{t_1, t_2, \ldots, t_{t_p}\}$ è l'insieme degli n_t transizioni
- I è la relazione di ingresso della transizione ed è rappresentata mediante archi diretti dai posti alle transizioni
- O è la relazione di uscita della transizione ed è rappresentata mediante archi diretti dalle transizioni ai posti
- $M = (m_1, m_2, \ldots, m_{n_p})$ è la marcatura iniziale: il generico m_i rappresenta il numero di token presenti nel posto p_i della marcatura M

Una marcatura è raggiungibile da un'altra marcatura se c'è una sequenza di scatti a partire dalla marcatura iniziale che risulta nella nuova marcatura. Una estensione delle Petri Networks (PN) sono le Stochastic Petri Networks. Agli elementi precedentemente evidenziati bisogna aggiungere un insieme L di N numeri reali non negativi che rappresentano i tassi di attivazione (firing rates) delle variabili aleatorie distribuite esponenzialmente associate a ciascuna transizione. Formalmente quindi una SPN è una sestupla (P, T , I , O, M, L):

$$L = \lambda_1(M), \lambda_2(M), ..., \lambda_N(M) \tag{4}$$

Si possono considerare altre tipologie di transizione, come quelle temporizzate, dove i tempi di accensione classici sono distribuiti esponenzialmente, e quelle immediate, in cui i tempi di accensione sono pari a zero. Questi modelli fanno riferimento alle Generalized Stochastic Petri Networks. Le Stochastic Reward Networks estendono ulteriormente la GSPN aggiungendo:

- **Guard Functions**: una transizione è considerata disabilitata a meno che la funzione di guardia non sia vera
- **Variable Arc Cardinality**: la cardinalità (o molteplicità) di un arco può essere espressa in funzione della marcatura di una SRN
- **General Marking Dependency**: i rate e le probabilità delle transizioni possono anche essere definiti come funzioni generali della marcatura di una SRN

In una SRN è possibile definire una **reward (rate) function** $X()$ che rappresenta un processo casuale non negativo indicativo delle condizioni del sistema. La definizione di $X()$ varia in base alla misura desiderata; per la valutazione dell'availability, al variare del tempo t, e quindi della marcatura, X(t) è definita come:

- 1 nel caso in cui il sistema sia funzionante al tempo t (up condition)
- 0 nel caso in cui il sistema sia non funzionante (down condition)

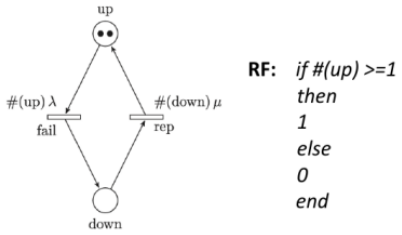

Figure 6: : Esempio di SRN modellante un sistema composto da due elementi riparabili con riparazioni indipendenti.

Per ogni marcatura della SRN al tempo t, viene valutata la reward rate function e il risultato viene assegnato come reward rate per quella marcatura. L'availability al tempo t si configura come:

$$A(t) = \Pr\{X(t) = 1\} = \mathbb{E}[X(t)] = \sum_{i \in S} r_i \cdot p_i(t) \tag{5}$$

dove:

- S è l'insieme dei marking, che può essere suddiviso in un subset di stati "up" (aventi reward rate $r_i = 1$) e un subset di stati "down" (aventi reward rate $r_i = 0$)
- $p_i(t)$ è la probabilità del sistema di trovarsi nello stato i all'istante t
- E[X(t)] è il reward rate atteso istantaneo

L'expected **steady-state reward rate** E[X], in maniera analoga, può essere calcolato come:

$$\mathbb{E}[X(t)] = \sum_{i} r_i \pi_i \tag{6}$$

dove

- r_i : reward rate nella marcatura tangibile i
- π_i: probabilità steady state (t $\to \infty$) di trovarsi nella marcatura i

L'elaborazione delle Continuous-Time Markov Chain (CTMC) che modellano sistemi complessi può presentare complicazioni dovute all'esplosione degli stati. Questo fenomeno è associato al fatto che la matrice generatrice infinitesimale di una CTMC può assumere dimensioni notevoli. La manipolazione o specifica manuale di una matrice così grande richiede tempo ed è suscettibile a errori umani.

L'utilizzo del formalismo delle reti di Petri può alleviare questo problema, poiché consente una specifica concisa del comportamento del sistema e può essere direttamente e automaticamente convertito in una CTMC. Tuttavia, questa astrazione potrebbe essere vista come problematica, poiché in alcune situazioni può risultare più complesso operare con una Stochastic Reward Net (SRN), e ottenere la corrispondente CTMC potrebbe non essere immediato.

2.1.3 RBD

Un RBD (Reliability Block Diagram) è una rappresentazione grafica della struttura logica di un sistema basata sull'analogia che viene assunta tra lo stato di corretto funzionamento del sistema e il passaggio da una corrente tra le due estremità di una rete di interruttori adeguata. Affinché sia possibile rappresentare un sistema attraverso un RBD questo deve avere una struttura monotona (o coerente). Un sitema è detto avere le una struttura monotona se presenta le seguenti caratteristiche:

- Ogni parte costituente del sistema può trovarsi solo in due stati: "funzionante" o "guasto".
- Il sistema stesso può avere solo due stati : "funzionante" o "guasto"
- Il sistema è certamente funzionante se ogni sua parte è funzionante.
- Il sistema è certamente guasto se ogni sua parte è guasta. Per un sistema monolitico è possibile creare un reliability block diagram (RBD) del sistema, descritto nello standard IEC 61078

Le tecniche di modellizzazione descritte sono principalmente destinate ad essere applicate a sistemi senza riparazioni e in cui l'ordine in cui si verificano i guasti non è rilevante. Tuttavia, l'assenza di riparazioni non è un'assunzione necessaria in quanto la supposizione predominante è quella dell'indipendenza statistica delle componenti.

La suddivisione in blocchi è guidata dall'idea che per utilizzarli dovremmo conoscere l'affidabilità (o disponibilità) di ciascun blocco. Successivamente, le connessioni logiche tra i blocchi devono essere definite, tenendo presente che

la struttura RBD non corrisponde necessariamente al modo in cui i componenti fisici sono collegati. Tramite l'RBD è possibile valutare l'affidabilità (o disponibilità) dell'intero sistema data l'affidabilità di ciascun blocco. Per calcolare l'affidabilità e/o la disponibilità di RBD in serie, si moltiplicano le affidabilità di tutti i componenti

$$\Pi_{i=1}^{N} R_i \tag{7}$$

Per calcolare l'affidabilità e/o l'availability di RBD in parallelo, possiamo invece procedere come segue:

$$1 - \Pi_{i=1}^{N}(1 - R_i) \tag{8}$$

Gli RBD (Reliability Block Diagram) costituiscono uno strumento semplice ma efficace, fornendo una metodologia chiara per derivare l'affidabilità complessiva di un sistema a partire da quella dei singoli componenti. Questo approccio facilita l'individuazione dei componenti critici nel garantire la disponibilità del sistema, agevolando l'identificazione di aree che potrebbero beneficiare di miglioramenti o l'implementazione di ridondanze.

Tuttavia, le semplificazioni effettuate, come l'assunzione di indipendenza tra i guasti dei componenti e la costanza nei tassi di guasto nel tempo, potrebbero introdurre una semplificazione eccessiva della realtà. Inoltre, questo approccio offre una visione limitata del processo di riparazione successivo a un guasto, limitando la comprensione della gestione delle eventuali interruzioni. Infine, poiché il formalismo opera a un livello di astrazione elevato, spesso si deve fare affidamento sui valori di affidabilità e disponibilità forniti dai produttori per ciascun componente, il che potrebbe comportare alcune limitazioni legate alla precisione dei dati di input.

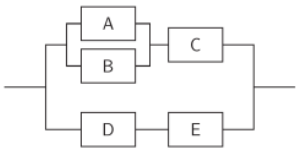

Figure 7: Esempio di RBD composto da sere e paralleli.

2.2 Confronto tra le tecniche

	Pro	Contro
Fault Tree	il grafico evidenzia i componenti critici legati ai guasti del sistema	difficile catturare le cause comuni dei guasti
CTCM	facile da rappresentare attraverso il diagramma dello spazio degli stati	approccio monolitico che porta a un'esplosione dello spazio degli stati per i sistemi del mondo reale
Stochastic Reward Networks	comoda interfaccia "di alto livello" che nasconde i dettagli tecnici	difficoltà di accesso agli stati sottostanti

Table 1: Presentazione Pro e Contro delle varie tecniche

Capitolo 3

3 Analisi dei sistemi

3.1 Modelli di disponibilità dei sistemi S1-S2-S3

Ora procederemo a fornire la rappresentazione dei modelli di disponibilità associati ai tre sottosistemi del nostro sistema, come dettagliato nel Capitolo 1. Il sistema in questione è composto dai sottosistemi S1, S2 e S3, ciascuno dei quali sarà oggetto di un'analisi dettagliata nei prossimi passaggi

3.2 Sistema S1: stazione di monitoraggio

La stazione di monitoraggio è strutturata con due sottosistemi identici, manifestando quindi una naturale ridondanza. All'interno di ciascun sottosistema, le componenti significative per il modello di disponibilità includono un sensore e un elemento di storage. Ora procediamo con la modellizzazione e l'analisi di ciascuna di queste componenti in modo dettagliato.

3.2.1 CTMC: sensore

Il sensore è modellato come una Continuous-Time Markov Chain a due stati, caratterizzata da:

MTTF_sens	400h
MTTR_sens	8h

Table 2: Parametri che caratterizzano l'unità riparabile Sensore

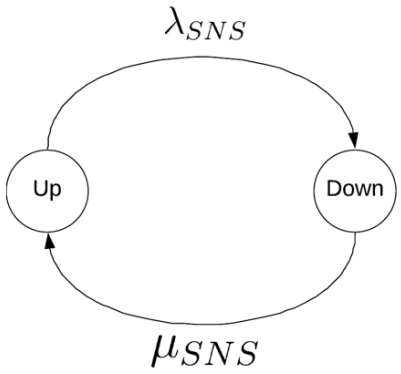

Figure 8: CTM che modella i due stati del sensore (risp. funzionante e non funzionante), ricordando che λ e μ hanno rispettivamente i valori di $1/MTTF$ e $1/MTTR$

3.2.2 CTMC: Storage

La Tabella seguente mostra i parametri che caratterizzano l'unità riparabile *Storage*. Lo Storage viene modellato come un CTMC (Continuos-Time Markov Chain). I parametri sono i seguenti:

Parametro	Descrizione	Valore
$1/\lambda\_stor$	mean time for storage failure	20000 hours
$1/\mu 1\_stor$	mean time for one disk repair	30 minutes
$1/\mu 2\_stor$	mean time for two disk repair	1 hour
$1/\chi\_stor$	mean time to copy the data	20 minutes
$1/\alpha\_rep$	mean time to technician summoned	30 minutes

Table 3: Parametri che caratterizzano l'unità riparabile Storage

Il sistema di storage è modellato come una configurazione RAID1 con due dischi rigidi, dettagliata nella Figura 9. In caso di guasto su uno dei dischi, il sistema attraversa diversi stati, tra cui U1, RP, U2, D1 e D2, con le relative transizioni.

Nello stato UP, entrambi i dischi sono completamente funzionanti. Gli stati

U1 e U2 riflettono le condizioni in cui solo uno dei dischi è operativo. Quando si verifica un guasto, il sistema passa da UP a U1. In questa situazione di guasto parziale, un tecnico viene inviato per la riparazione con un tempo medio di risposta di $1/\alpha_{rep}$. Successivamente, il sistema raggiunge lo stato RP, dove il disco guasto viene sostituito con un tempo medio di riparazione di $1/\mu_{1stor}$, e il sistema entra nello stato U2. Durante la fase di sostituzione, i dati devono essere copiati sul nuovo disco appena inserito con un tempo medio di copia di $1/\chi_{stor}$. Se durante l'operazione di copia si verifica un guasto sull'altro disco, il sistema passa allo stato D2. Nel caso in cui il guasto sull'altro disco avvenga prima dell'arrivo del tecnico, il sistema passa da U1 a D1 senza dischi rimanenti. Lo stato D2 viene quindi raggiunto da D1 quando il tecnico arriva con una frequenza α_{rep}. Infine, il sistema si ripristina dalla condizione D2 allo stato iniziale completamente funzionante (UP) con un tempo medio di riparazione di $1/\mu_{2stor}$. Questo modello descrive in modo dettagliato il comportamento del sistema di storage in presenza di guasti e le operazioni necessarie per il ripristino delle funzionalità.

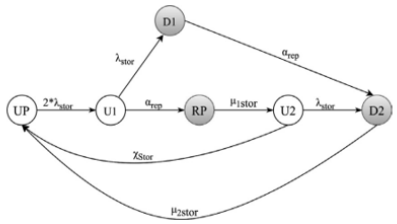

Figure 9: CTMC che modellizza la componente Storage

3.2.3 Rappresentazione dell'intero sistema S1 attraverso un Fault Tree

Le foglie del sottosistema S1 rappresentano le singole componenti discusse precedentemente. La modellazione del sottosistema tiene presente che esso è considerato funzionante solo se entrambe le sue componenti, sensore e storage, sono operative. Poichè sappiamo che l'intero sistema S1 è in realtà composto da due sottosistemi, è possibile reiterare l'approccio di un singolo sottositema per ottenere il Fault Tree rappresentante l'intero sistema. In questo caso, sappiamo che l'intero sistema S1 funziona se almeno uno dei due sottosistemi è funzionante, pertanto si otterrebbe la "failure" nel caso in cui entrambi i sottosistemi non funzionassero. La rappresentazione del sistema è visualizzabile attraverso la seguente figura:

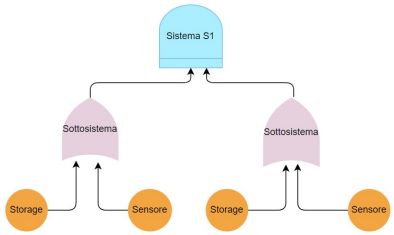

Figure 10: Rappresentazione Sistema S1

3.3 Modellazione sistema S2: nodo di rete SFC

Per la modellazione affidabilistica, il sistema SFC può essere interpretato seguendo un modello a 5 layer, conforme a quanto illustrato nella Figura 1.1. La rappresentazione accurata di questo modello è stata ottenuta mediante l'utilizzo di una SRN, illustrata nella Figura 11. I valori dei vari parametri sono dettagliati nella Tabella 4 al fine di fornire una descrizione completa delle caratteristiche del sistema.

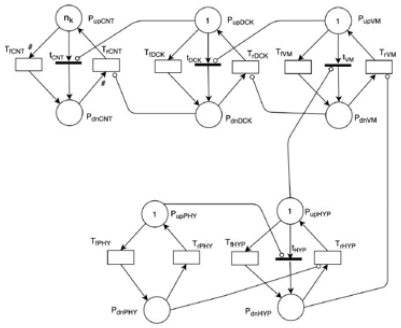

Figure 11: Stocastic Reward Net modellante S2

Parametro	Descrizione	Valore
$1/\lambda_{\mathrm{CNT}}$	Container MTTF (h)	500
$1/\lambda_{\mathrm{DKC}}$	Docker daemon MTTF (h)	1000
$1/\lambda_{\mathrm{VM}}$	Virtual machine MTTF (h)	2880
$1/\lambda_{\mathrm{HPV}}$	Hypervisor MTTF (h)	2880
$1/\lambda_{\mathrm{HW}}$	Hardware MTTF (h)	60000
$1/\mu_{\mathrm{CNT}}$	Container MTTR (s)	2
$1/\mu_{\mathrm{DKC}}$	Docker daemon MTTR (s)	5
$1/\mu_{\mathrm{VM}}$	Virtual Machine MTTR (h)	1
$1/\mu_{\mathrm{HPV}}$	Hypervisor MTTR (h)	2
$1/\mu_{\mathrm{CNT}}$	Hardware MTTR (h)	8

Table 4: Parametri relativi alla Stocastic Reward Net

La nostra rete è progettata per rappresentare le diverse componenti del sistema tra cui container, docker, virtual machine, hypervisor e hardware. Inizialmente il sistema funziona correttamente, ciò è intuibile dal fatto che i posti "up" sono marcati con almeno un gettone. In particolare, nel caso del posto P_{upCNT} , il cui numero di gettoni è indicato ocn n_k si hanno 3 gettoni. Ogni componente è soggetta a possibili guasti, modellati attraverso transizioni contrassegnate con il pedice "f". E' importante notare che, se una transizione di guasto coinvolge una componente posizionata al di sotto dei container, la perdita di una marca dal posto "up" disabilita l'arco inibitore collegato alla transizione immediata relativa alla componenete successiva. Ciò comporta l'attivazione della transizione immediata che porta tutte le componenti in uno stato "down" a cascata. Questo approccio consente di modellare il comportamento reale del sistema, in cui un guasto all'hardware impedisce il funzionamento di qualsiasi software che sia di virtualizzazione containerizzato. È interessante notare che la riparazione di una componente più in alto nella gerarchia non è possibile fintanto che le componenti più basse non vengono riparate. Anche in questo caso, la dipendenza

sequenziale nella riparazione delle componenti aggiunge un comportamento realistico al modello, riflettendo accuratamente la natura del sistema che stiamo analizzando. E' possibile notare, inoltre, che le transizioni T_{fCTN} e T_{rCTN} sono "marketing dependent". Il rate della transizione T_{fCNT} dipende dal numero di gettoni in P_{upCNT}, mentre quello di t_{rCNT} dipende dal numero di gettoni in P_{dnCNT}. Da un punto di vista pratico, questa attenzione nel modello indica che eventuali guasti e/o riparazioni ai container sono fra loro indipendenti. Per procedere con il calcolo della availability de sistema, considerato funzionante quando vi è almeno un gettone nel posto P_{upCNT} , la rete è accompagnata dalla seguente reward function:

Algorithm 1 Reward Function

if $\#P_{\text{upCNT}} \geq 1$ **then** ▷ Il valore #(P) fa riferimento al numero di gettoni all'interno del posto (P)
    1
else
    0
end if

3.4 Sistema 3 : Elaboratore

Il sistema S3 è progettato per rappresentare un elaboratore che include una CPU, un elemento di storage, un software di analisi dei dati, un sistema operativo e una parte hardware. Il corretto funzionamento del sistema è condizionato al funzionamento simultaneo di tutte le sue componenti. Pertanto, si è proceduto a una rappresentazione del sistema mediante un Diagramma a Blocchi di Affidabilità (RBD) in serie, come illustrato nella Figura 12.

Adottando un approccio multilivello, ciascuna componente è stata modellata individualmente come una Continuous-Time Markov Chain (CTMC).

Figure 12: Enter Caption

3.4.1 CTMC: SO, HW, App

Il sistema operativo, l'hardware e App vengono modellati come Continuos-Time Markov Chain a due stati sono caratterizzati da:

MTTF_SO	600 h
MTTR_SO	120 s

Table 5: Parametri che caratterizzano l'unità riparabile del Sistema Operativo

MTTF_HW	30000 h
MTTR_HW	4h

Table 6: Parametri che caratterizzano l'unità riparabile dell'Hardware

MTTF_SW	500 h
MTTR_SW	60 s

Table 7: Parametri che caratterizzano l'unità riparabiledel Software

3.4.2 CTMC: CPU

Anche la CPU, come nel caso precedente, è stata modellata ocme una Continuos-Time Markov Chain. La differenza sta nel fatto che in questo caso è stato utilizzato un modello a tre stati. La figura 13 mostra il modello appena descritto:

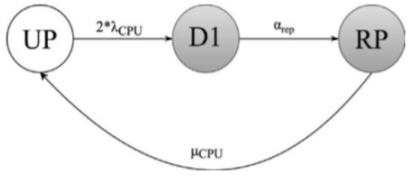

Figure 13: CTMC che modella la CPU

Parametro	Descrizione	Valore
$1/\lambda_{\text{CPU}}$	tempo medio per il guasto della CPU	2500 hours
$1/\mu_{\text{CPU}}$	tempo medio per la riparazione della CPU	30 minutes
$1/\alpha_{\text{rep}}$	tempo medio all'arrivo tecnico	30 minutes

Table 8: Parametri relativi alla CTMC CPU

Nel caso in cui si dovesse verificare un guasto il sistema si sposta nello stato D1 in cui viene invocato un riparatore, che risulterà disponibile entro un certo tempo. All'arrivo del riparatore il sistema si sposta nello stato RP. Da tale stato è possibile tornare in UP dopo un tempo dettato da $1/\mu_{CPU}$

Capitolo 4

4 Stima a massima verosimiglianza dei tassi di guasto e riparazione per il componente hardware (PHY)

Nel contesto del progetto attuale, con una particolare focalizzazione sull'architettura Service Function Chain (SFC) considerata come un singolo nodo all'interno di un sistema più ampio, con attenzione specifica allo strato hardware denominato PHY (Physical Layer), emerge la necessità di valutare accuratamente i tassi di guasto λ_{PHY} e di riparazione μ_{PHY}. Poiché questi parametri non sono stati forniti direttamente, è necessario valutarli separatamente, dato il loro ruolo cruciale nella determinazione dell'affidabilità e della disponibilità del sistema. La ricerca e la valutazione dei tassi di guasto e di riparazione di un componente può essere intrapresa attraverso due approcci differenti:

- da una parte, si può considerare l'implementazione di modelli teorici costruiti sulla base di ipotesi e informazioni provenienti da esperti del settore, al fine di creare una rappresentazione idealizzata del componente
- dall'altra parte, si può ricorrere alla raccolta di dati empirici in termini di misurazioni dirette, ottenuti tramite analisi e test sul campo del componente in questione.

Per il layer fisico dell'architettura SFC, essendoci stati già forniti i dati relativi alle misurazioni, saranno utilizzati questi ultimi per la valutazione e la conseguente stima dei tassi di guasto λ_{PHY} e di riparazione μ_{PHY} corrispondenti. Entrambi gli approcci offrono vantaggi distinti: da un lato, i modelli forniscono un quadro teorico utile per guidare l'analisi di affidabilità e della disponibilità o di un sistema o di un componente; dall'altro, per garantire l'accuratezza e la rilevanza di tali modelli è fondamentale che siano supportati e caratterizzati da dati concreti, sopratutto quando i parametri non sono noti a priori.
Si è scelto di adottare un approccio basato sulla stima a massima verosimiglianza (MLE) per determinare i valori di λ_{PHY} e μ_{PHY}, utilizzando i dataset denominati repairs e failures assegnati e tenendo in considerazione l'assunzione sui tempi di riparazione e di guasto esponenziali per il componente in esame: questo metodo non solo ci permette di sfruttare le informazioni disponibili nei dataset, ma fornisce anche uno strumento robusto e scientificamente fondato per ottenere stime affidabili e precise.

4.1 Stima a Massima Verosimiglianza (MLE)

Per la successiva analisi, abbiamo adottato un modello parametrico basato sull'assunzione che i tempi di guasto e di riparazione del componente PHY seguano una distribuzione esponenziale. Questa scelta è motivata dalla natura

accidentale dei guasti che si verificano nel componente, i quali non sono correlati a fenomeni di usura o invecchiamento.

Con l'adozione di un modello di distribuzione per la funzione di non affidabilità F(x), i cui parametri sono sconosciuti, abbiamo optato per una **strategia parametrica** basata sulla stima di tali parametri a partire dai dati disponibili. Sono disponibili diverse tecniche per le procedure di stima parametriche, che spaziano dalle analitiche come il Metodo dei Minimi Quadrati e la Massima Verosimiglianza, ai metodi grafici come i Probability Plots. Nel contesto del nostro progetto, abbiamo scelto la tecnica di stima a **Massima Verosimiglianza**. Formalmente:

Definizione 4.1. *Sia D un insieme di dati osservati e sia $M(\theta)$ un modello di probabilità caratterizzato dal vettore dei parametri $\theta = (\theta_1, \theta_2, ..., \theta_n)'$ definito sull'insieme k-dimensionale Θ, la funzione di θ*

$$L(\theta) = c \cdot [PrD; \theta] \tag{9}$$

*proporzionale tramite una quantità scalare c indipendente da θ alla probabilità di osservare i dati D nel modello $M(\theta)$, è chiamata **likelihood function***

Sia X una random variable con funzione di densità di probabilità $p(x; \theta)$ parametrizzata da θ che rappresenta il modello della popolazione, da cui si estrae un campione casuale composto da n random variables indipendenti e identicamente distribuite (i.i.d.) $X_1, ..., X_n$ i cui valori sono raccolti in un vettore $x = x_1, ..., x_n$, la likelihood function è definita da

$$L(\theta; x) = c \cdot p(x_1; \theta) ... \cdot p(x_n; \theta) = c\Pi_{i=1}^{n} p(x_i; \theta) \tag{10}$$

La likelihood function fornisce un potente strumento per fare inferenza sul parametro θ sconosciuto, dati i dati x e il modello della popolazione $M(\theta)$, in questo caso la $pdfp(x; \theta)$.

Lo stimatore a massima verosimiglianza di θ è il valore θ appartenente allo spazio dei parametri Θ che massimizza la funzione $L(\theta)$, definito formalmente come:

$$\hat{\theta} = argmax_{\Theta} L(\theta) \tag{11}$$

Il criterio di massimizzazione della likelihood function si basa sulla considerazione che la MLE rappresenta il valore del parametro incognito che massimizza la probabilità di accadimento dei dati osservati, dato il modello di probabilità $M(\theta)$. Supponendo che le derivate parziali di $L(\theta)$ esistano nello spazio Θ, la stima a massima verosimiglianza è data dal punto $\hat{\Theta}$ in cui tutte le derivate parziali prime devono essere zero, cioè dove il gradiente di $L(\theta)$ è uguale al vettore nullo k-dimensionale

$$\nabla L(\theta) = 0 \tag{12}$$

Pertanto, la stima ML è da ricercare tra i punti stazionari, cioè tra le soluzioni del sistema di k equazioni

$$\frac{\partial(\theta_i)}{\partial(\theta_i)} = 0, i = 1, ..., k \tag{13}$$

in cui la matrice Hessiana $H(\theta)$ è definita negativa, ovvero tutti i suoi autovalori sono negativi.

Essendo interessati a trovare il massimo, ogni quantità moltiplicativa all'interno di $L(\theta)$ che non contiene θ non è rilevante e può essere eliminata dai calcoli. Inoltre, poichè cercare il massimo di una funzione equivale a cercare il massimo della sua trasformazione monotona, di solito conviene operare sulla trasformazione logaritmica della verosimiglianza $l(\theta) = \log L(\theta)$, tipicamente denominata **log-likelihood**.

Tra i vantaggi principali che hanno portato ad adottare il metodo di stima a massima verosimiglianza è da evidenziare la sua estrema versatilità, poichè si adatta alla maggior parte dei modelli di distribuzione e dei sampling schemes presenti nelle applicazioni.

Ovviamente per n finito la funzione di distribuzione degli stimatori ML dipende sia dal tipo di campionamento che dalla distribuzione della v.a. . Per $n \to \infty$, gli stimatori ML mostrano alcune proprietà asintotiche utili:

1. sono **consistenti**, ovvero convergono al valore vero
2. sono **asintoticamente efficienti**, a varianza più piccola
3. sono **asintoticamente Normali**, ovvero la loro funzione di distribuzione converge ad una normale
4. presentano la **proprietà di invarianza**: sia $\eta = g(\theta)$ una funzione biunivoca di parametro θ, il metodo a massima verosimiglianza ha la proprietà che se $\hat{\theta}$ è l'MLE di θ, allora $\hat{\theta} = g(\hat{\theta})$ è l'MLE di η. Questa è una proprietà particolarmente rilevante ed utilizzata per la stima della reliability.

4.1.1 Il modello Esponenziale

Uno dei modelli matematici più conosciuti utilizzati per descrivere la reliability function di un dispositivo è il cosiddetto modello esponenziale, la cui reliability function è

$$R(x) = exp(-\frac{x}{\theta}) \tag{14}$$

dove θ è l'average lifetime, cioè il valore atteso del lifetime del dispositivo, a partire dal quale è possibile calcolare il valore di reliability $R(x)$ per ogni mission time x, e quindi $F(x)$ come

$$F(x) = 1 - R(x) = 1 - exp(-\frac{x}{\theta}) \tag{15}$$

Dalla definizione di pdf, la failure pdf per il modello esponenziale è

$$f(x) = \frac{\partial F(x)}{\partial x} = \frac{1}{\theta}\exp(-\frac{x}{\theta}) \tag{16}$$

da cui è possibile ricavare la failure rate function $h(x)$ come

$$h(x) = \frac{f(x)}{R(x)} = \frac{\theta^{-1}exp(-\frac{x}{\theta})}{exp(-\frac{x}{\theta})} = \frac{1}{\theta} \tag{17}$$

dimostrando che, nel caso dell'*exponentiable reliability*, il failure rate è uguale all'inverso dell'average lifetime, ed è quindi una costante che non dipenda da x_t, ovvero il tempo

Osservazione: Nel modello esponenziale il failure rate non dipende da quanto tempo il device ha operato.

Adottare il modello esponenziale equivale quindi ad assumere che il meccanismo di guasto dell'unità sia lo stesso del tempo $x = 0$ legandosi solo a fenomeni accidentali e non a fenomeni di usura, essendo in questo modello il rischio di guasto indipendente dall'età. Per questa proprietà, la distribuzione esponenziale è detta senza memoria implicando l'uguaglianza tra la distribuzione della vita residua Y e la distribuzione dell'intera vita X. Tutte le considerazioni condotte giustificano ampiamente le assunzioni fatte sul componente tecnologico i cui parametri tasso di guasto e di riparazione verranno stimati con la metodologia precedentemente presentata.

4.1.2 Modello Esponenziale MLE

Sia X una random variable che modella il lifetime di una determinata unità i cui failure events si presuppone siano indipendenti dall'età accumulata, X è distribuita esponenzialmente con pdf data da

$$f(x;\theta) = \theta^{-1}exp(\frac{-x}{\theta}) \tag{18}$$

dove $\theta = \mu = \frac{1}{\lambda}$ è il parametro da stimare

Sia $x = (x_1, x_2, ..., x_n)$ un campione completo di n osservazioni della random variable X, la likelihood function è

$$L(\theta, x) = \Pi_{i=1}^{n} f(x_1) = \theta^{-n} e^{-\frac{1}{\theta}\sum_i x_i} \tag{19}$$

e definendo $T^* = \sum_{i=1}^{n} x_i$ la statistica sufficiente per la stima di θ , allora la funzione di log-verosimiglianza può essere espressa come:

$$l(\theta) = \log L(\theta) = -n\log\theta - \frac{1}{\theta}\sum_i x_i = -n\log\theta - \frac{T^*}{\theta} \tag{20}$$

e da ciò si evince che si ha bisogno unicamente della somma dei campioni e non dei singoli campioni, cioè la statistica è sufficiente e minimale.

Quindi, per calcolare l'MLE dobbiamo risolvere

$$\frac{\partial l(\theta)}{\partial \theta} = 0 \rightarrow -\frac{n}{\theta} + \frac{1}{\theta^2} T^* = 0 \tag{21}$$

da cui

$$\hat{\theta} = \frac{T^*}{n} = \tilde{x} = \mu_{ML} \tag{22}$$

Pertanto, l'MLE è pari alla media campionaria delle osservazioni, cioè in linea con il fatto che rappresenta il valore atteso di X

> Una statistica sufficiente cattura tutte le informazioni contenute nel campione riguardo al parametro che si vuole stimare, senza perdere alcuna informazione utile.

4.1.3 MLE per dati censurati a destra

Il dataset contentente i tempi al guasto del componente presenta la particolarità di essere un campione censurato di tipo I, dove alcuni dati sui tempi di guasto non sono completamente osservati. La censura difatti si verifica quando l'informazione su un evento non è disponibile (in questo caso il guasto): se un componente viene sostituito prima di guastarsi, il tempo al guasto non è conosciuto e viene considerato censurato. La censura di tipo I è una forma di right censoring in cui la durata del test per ciascun unità del campione non può superare una quantità fissata τ. Pertanto, il test termina al tempo τ indipendentemente dal fatto che alcune unità siano ancora funzionanti a quel tempo, e l'informazione fornita da questo test è che la durata (non osservata) delle unità non guastate è maggiore di τ. Ritornando alla procedura di stima, con campioni censurati di tipo I, l'approccio è il seguente: definita la likelihood totale data da

$$L(\theta, t) = L(\theta) = c\Pi_i f(t_i) \cdot \Pi_i [F(t_j) - F(t_{j-1})^{d_j}] \cdot [F(t_j)]^{l_j} \cdot [1 - F(t_j)]^{r_j} \tag{23}$$

dove

- d_j è il numero di unità guastate nell'intervallo (t_{j-1}, t_j)
- l_j è il numero di unità guastate prima di t_j
- r_j è il numero di unità guastate dopo t_j

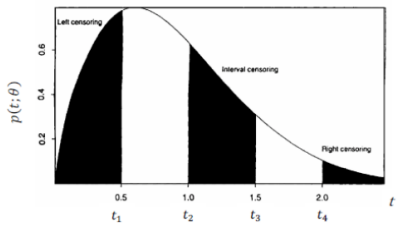

Figure 14: Esempio MLE dati censurati

Sia $x = (x_1, ..., x_m; \tau_i, ..., \tau_{n-m})$ il campione osservato degli m failure times x_i e dei $n-m$ survival times τ_j, la funzione di verosimiglianza risultante per censura a destra di tipo I, considerando unità la cui vita è modellata da una distribuzione esponenziale, è

$$L(\theta) = L(\theta, x) = \Pi_i^m [\theta^{-1} e^{-\frac{x_i}{\theta}}] \cdot \Pi_i^{n-m} e^{-\frac{\tau_j}{\theta}} = \theta^{-m} e^{\frac{\sum_{i=1}^m x_i + \sum_{j=1}^{n-m} \tau_j}{\theta}} \tag{24}$$

Introducendo la statistica sufficiente

$$T^* = \sum_{i=1}^{m} x_i + \sum_{j=1}^{n-m} \tau_j \tag{25}$$

la likelihood function è

$$L(\theta) = \theta^{-m} e^{-T^* \theta^{-1}} \tag{26}$$

per trovare l'MLE dobbiamo risolvere

$$\frac{d}{d\theta} l(\theta) = 0 \rightarrow -\frac{m}{\theta} + \frac{T^*}{\theta^2} = 0 \rightarrow (\hat{\theta}) = \frac{T^*}{m}$$

ottenendo lo stimatore risultante $\hat{\theta}$.

> Si noti che la quantità T^* può essere interpretata come il tempo totale di test delle n unità.

Dai risultati ottenuti possiamo dire che lo stimatore ML per il parametro θ si ottiene, indipendentemente dal tipo di campionamento, dividendo il tempo di prova complessivo T^* accumulato da tutte le n unità per il numero di esemplari che hanno fallito m ($<$ n). Tuttavia, sia il tipo di censura che di campionamento vanno ad influenzare le proprietà statistiche degli stimatori risultanti.

4.2 Analisi dei dataset

4.2.1 Analisi *repairs_gr1*

Il seguente dataset contiene i tempi di riparazione non censurati per il componente PHY, che risulta essere fondamnetale per la procedura di stima finalizzata all'ottenimento del tasso di riparazione μ_{PHY}. A tal scopo è stato effettuato uno script Matlab:

```
dati = dlmread('repairs_gr1.txt');

% Calcola il numero di campioni
numero_di_campioni = length(dati);
% Calcola la media
media = mean(dati);

```

```
% Calcola la deviazione standard
deviazione_standard = std(dati);

% Calcola il minimo
minimo = min(dati);

percentile_25 = prctile(dati, 25);

mediana = median(dati);

percentile_75 = prctile(dati, 75);

% Calcola il massimo
massimo = max(dati);

% Visualizza i risultati
fprintf('Numero di campioni: %d\n', numero_di_campioni);
fprintf('Media: %.2f\n', media);
fprintf('Deviazione standard: %.2f\n', deviazione_standard);
fprintf('Minimo: %.2f\n', minimo);
fprintf('25%% percentile: %.2f\n', percentile_25);

fprintf('Mediana: %.2f\n', mediana);
fprintf('75%% percentile: %.2f\n', percentile_75);

fprintf('Massimo: %.2f\n', massimo);
```

Listing 1: Script Matlab Analisi dei dati

L'esecuzione di tale script ha portato ai seguenti risultati:

- **Numero di campioni:** 70
- **Media:** 7.86 ore
- **Deviazione Standard:** 7.75 ore
- **Minimo:** 0.07 ore
- **25° percentile**: 2.59 ore
- **Mediana:** 5.25 ore
- **75° Percentile:** 10.29 ore
- **Massimo:** 34.62 ore

Da questi risultati siamo in grado, di poter stabilire il tempo necessario per riparare il componente che stiamo analizzando; la media, infatti, indica un tempo di riparazione considerato accettabile, ma la Deviazione Standard fa pensare a una variazione non trascurabile nei tempi di riparazione dei diversi casi.

4.2.2 Analisi *Failurel_gr1*

Il dataset "failureI_gr1" presenta un insieme di tempi al guasto del componente in esame con la particolarità di includere dei dati censurati. I dati censurati sono indicati nel file "censuring_gr1", in cui 1 indica il dato censurato nella riga corrispondente al dataset "failureI_gr1". Con l'utilizzo dello script MATLAB mostrato di seguito:

```
% Leggi i dati da un file di testo
dati = dlmread('failureI_gr1.txt');
dati_censoring = dlmread('censoring_gr1.txt');
dati_censoring = dati_censoring(dati_censoring == 1.0000000e+00);

% Calcola il numero di campioni
numero_di_campioni = length(dati);
% Calcola la media
media = mean(dati);

% Calcola la deviazione standard
deviazione_standard = std(dati);

% Calcola il minimo
minimo = min(dati);

% Calcola il 25%% percentile
percentile_25 = prctile(dati, 25);

% Calcola la mediana (50%% percentile)
mediana = median(dati);

% Calcola il 75%% percentile
percentile_75 = prctile(dati, 75);

% Calcola il massimo
massimo = max(dati);

% Visualizza i risultati
fprintf('Numero di campioni: %d\n', numero_di_campioni);
fprintf('Media: %.2f\n', media);
fprintf('Deviazione standard: %.2f\n', deviazione_standard);
fprintf('Minimo: %.2f\n', minimo);
fprintf('25%% percentile: %.2f\n', percentile_25);
fprintf('Mediana: %.2f\n', mediana);
fprintf('75%% percentile: %.2f\n', percentile_75);
fprintf('Massimo: %.2f\n', massimo);

% Calcola il numero di campioni per censoring
```

```
numero_di_campioni_censoring = sum(dati_censoring);

% Visualizza il numero di campioni per censoring
fprintf('Numero di campioni per censoring: %d\n',
    numero_di_campioni_censoring);
```

Listing 2: Script Matlab Analisi dei dati

precedente abbiamo ottenuto i seguenti risultati:

- **Numero di campioni:** 80
- **Media:** 27352.34 ore
- **Deviazione Standard:** 19341.82 ore
- **Minimo:** 161.13 ore
- **25° percentile:** 9669.69 ore
- **Mediana:** 24840.66 ore
- **75° Percentile:** 42953.53 ore
- **Massimo:** 60000.00 ore
- **Numero campioni censurati:** 10

Dato il valore ottenuto dalla Deviazione Standard possiamo concludere che il dataset dei tempi di guasto mostra una vasta gamma di variazioni. Il dato censurato serve ad indicare situazioni in cui il guasto non si è verificato entro la fine del periodo di osservazione. Ciò ovviamente richiede una particolare attenzione nell'analisi statica in quanto i dati censurati influenzano la comprensione della durata della vita del componente stesso.

4.3 Stima del tasso di riparazione

Il seguente script Matlab:

```
close all;
clc;

% Carica il file
repairTimes = load("repairs_gr1.txt");

% Calcolo del numero di campioni
nsamps = length(repairTimes);

% Stima del parametro mu_PHY

%% Metodo 1
```

```
MTTR_mean = sum(repairTimes) / nsamps;
fprintf('Metodo 1\nMTTR est. : %f [h]\n', MTTR_mean);
fprintf('Repair rate  est. : %f [1/h]\n\n', 1/MTTR_mean);

%% Metodo 2
MTTR_mle = mle(repairTimes, 'Distribution', 'Exponential');
fprintf('Metodo 2\nMTTR est. : %f [h]\n', MTTR_mle(1));
fprintf('Repair rate  est. : %f [1/h]\n\n', 1/MTTR_mle(1));

%% Metodo 3
[MTTR_exp, MTTR_ci] = expfit(repairTimes);
fprintf('Metodo 3\nMTTR est. : %f [h] \nMTTR est. 95%% confidence
    interval : \nlower bound: %f [h]\nupper_bound: %f [h]\n', MTTR_exp,
    MTTR_ci(1), MTTR_ci(2));
fprintf('Repair rate  est. : %f [1/h]\n\n', 1/MTTR_exp);

%% Metodo 4
options = optimset('Display', 'off', 'MaxIter', 10000);
theta0 = 2;
[theta, fval, exitflag, output, grad, hessian] = fminunc(@log_lik,
    theta0, options, repairTimes);
fprintf('Metodo 4\nMTTR est. : %f [h]\n', exp(theta));
fprintf('Repair rate  est. : %f [1/h]\n\n', 1/exp(theta));

% Plot
figure
histogram(repairTimes, 'Normalization', 'pdf');
title('Repair Times - Complete Sample', 'FontSize', 24);
xlabel('Repair Times', 'FontSize', 20);
ylabel('pdf', 'FontSize', 20);

% Personalizza la scala dell'asse x
ax = gca;
ax.FontSize = 14;

% Imposta la scala dell'asse y da 0 a 1
ax.YLim = [0 0.13];
ax.XLim = [0 40];

hold on;

% Genera il vettore xax
xax = linspace(0, ax.XLim(2), 100);

% Plot con scala logaritmica sull'asse y
plot(xax, exppdf(xax, MTTR_exp), 'r:', 'LineWidth', 2);
```

```
57 legend('Hist', 'Est.', 'FontSize', 20);

59 % Imposta ticks sull'asse y ogni 0.01
yticks(0:0.01:0.13);
```

Listing 3: Script per la stima del tasso di riparazione

ci permette di analizzare la procedura di stima del tasso di riparazione del componente PHY. Il primo passo è stato quello di caricare il file *repairsgr1* che contiene l'insieme di osservazioni temporali. Il processo di stima viene articolato in 4 metodologie distinti:

1. *Stima attraverso la Media:* calcoliamo l'MTTR come media aritmetica dei tempi di riparazione e il tasso di riparazione μ_{PHY} che sappiamo essere il suo inverso
2. *Maximum Likelihood Estimation:* tale approccio segue la metodologia della Maximum Likelihood Estimation. In aiuto viene fornita la funzione matlab ***mle()***
3. *Exp Distribution Fit Function:* Utilizzando la funzione MATLAB expfit() con i tempi di riparazione in input per ottenere la stima della media di dati distribuiti esponenzialmente, accompagnata dall'intervallo di confidenza del 95% per la stima.
4. *Log-likelihood Estimation*

Tali procedure sono servite per ottenere un set di dati comparabile e per poter confermare i vari risultati ottenuti. Il plot all'interno dello script, ha lo scopo di combinare all'istogramma dei tempi di riparazione analizzati la funzione di densità di probabilità stimata.

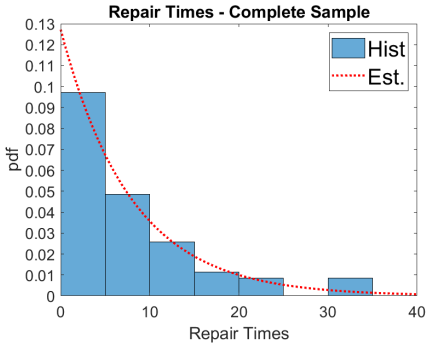

Figure 15: Istogramma dei tempi di riparazione e pdf stimata

Il seguente grafico ha lo scopo di mostrare un confronto tra la distribuzione dei tempi e la stima della funzione di densità di probabilità. I rettangoli blu rappresentano l'istogramma che mostra la frequenza relativa ai tempi di riparazione

all'interno di intervalli ben precisi, mentre la stima della pdf è visualizzabile attraverso la linea tratteggiata che si presume che segua una distribuzione di tipo esponenziale. Analizzando ora la seguente immagine possiamo notare che la pdf segue abbastanza bene la distribuzione dei tempi di riparazione, potendo quindi concludere che il modello esponenziale è un buon modello per i dati osservati. Ovviamente tale modello non si sposa al 100% con l'istogramma, infatti dall'immagine è possibile vedere delle discrepanze,sorattutto nell'intervallo in cui all'interno del nostro dataset non sono presenti valori (intervallo 25-30) ma in linea di massima possiamo confermare che la pdf stimata approssima bene l'andamento dei dati osservati.

4.4 Stima del tasso di guasto

Il seguente script implementa la procedura di stima del tasso di guasto del componente PHY a partire da un campione censurato a destra.

1. *Stima del tempo medio al guasto:*
   - Approccio corretto: calcolo del tempo medio al guasto utilizzando i soli dati non censurati, fornendo così una stima non distorta dell'MTTF
   - Approccio errato: calcolo del tempo medio al guasto considerando tutti i campioni inclusi quelli censurati, portando invece ad una stima distorta a causa della censura dei dati
2. *Maximum Likelihood Estimation* con censura condotta attraverso la MATLAB function mle() con una configurazione che tiene conto della censura a destra, offrendo una stima del tasso di guasto che incorpora tali caratteristiche
3. *Log-likelihood estimation* con censura a completamento dell'analisi offrendo una valutazione completa e approfondita del tasso di guasto

```
% Inizializzazione dell'ambiente
close all
clc
clearvars

% Carica il file
failureTimes = load("failureI_gr1.txt");
censoringData = load("censoring_gr1.txt");

% Seleziona solo i tempi di guasto non censurati
nonCensoredFailureTimes = failureTimes(censoringData == 0);

% Calcola il numero di campioni censurati e non censurati
nsamps_nocens = length(nonCensoredFailureTimes);
nsamps_tot = length(failureTimes);
```

```
16
% Stampa del numero di campioni
18 fprintf('Campioni totali: %d\n', nsamps_tot);
fprintf('Campioni non censurati: %d\n', nsamps_nocens);
20 fprintf('Campioni censurati: %d\n', nsamps_tot - nsamps_nocens);

22 % Stima del parametro lambda_PHY
%% Metodo 1
24 MTTF_mean = sum(failureTimes) / nsamps_nocens;
MTTF_wr_mean = sum(failureTimes) / nsamps_tot;
26 fprintf('\nMetodo 1\nMTTF est.: %f [h]\n', MTTF_mean);
fprintf('Failure rate est.: %f [1/h]\n\n', 1/MTTF_mean);
28 fprintf('MTTF wrong est.: %f [h]\n', MTTF_wr_mean);
fprintf('Failure rate wrong est.: %f [1/h]\n\n', 1/MTTF_wr_mean);
30
%% Metodo 2
32 % mle() function with right censoring
MTTF_mle = mle(failureTimes, 'Distribution', 'Exponential', 'Censoring'
    ,censoringData);
34 fprintf('Metodo 2\nMTTF est.: %f [h]\n', MTTF_mle);
fprintf('Failure rate est.: %f [1/h]\n\n', 1/MTTF_mle);
36
%% Metodo 3
38 options = optimset('Display', 'off', 'MaxIter', 10000);
theta0 = 10;
40 [theta, fval, exitflag, output, grad, hessian] = fminunc('log_lik_cens'
    , theta0, options, censoringData, failureTimes);
exp(theta);
42 fprintf('Metodo 3\nMTTF est.: %f [h]\n', exp(theta));
fprintf('Failure rate est.: %f [1/h]\n\n', 1/exp(theta));
44
%% Plot
46 figure
histogram(failureTimes(censoringData == 0), 'Normalization', 'pdf');
48 title('Failure times: type I right censoring', 'FontSize', 24);
xlabel('failureTimes (uncensored)', 'FontSize', 20);
50 ylabel('pdf', 'FontSize', 20);
ax = gca;
52 ax.FontSize = 16;
hold on;
54 xax = linspace(0, ax.XLim(2), 100);
plot(xax, exppdf(xax, MTTF_mean), 'r:', 'LineWidth', 2);
56 plot(xax, exppdf(xax, MTTF_wr_mean), 'k:', 'LineWidth', 2);
legend('hist', 'est.', 'wrong est. with stopping times', 'FontSize',
    20);
```

Listing 4: Analisi dei tempi di guasto con censura

Dopo la fase di stima, si procede con un'analisi dei risultato generando un plot il quale mira a confrontare i tempi al guasto osservati mediante un istogramma con le funzioni di densità di probabilità stimate, sia in maniera corretta che errata permettendo di visualizzare le discrepanze tra i due approcci e l'impatto della censura sulla stima del parametro.

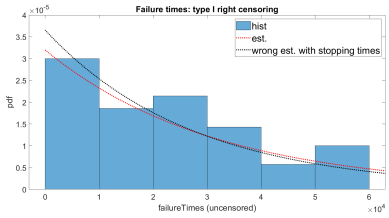

Figure 16: Istogramma dei tempi al guasto e pdf sia correttamente che errata

Nel seguente plot i rettangoli in blu rappresentano l'istogramma dei tempi di guasto non censurati, mentre la linea rossa tratteggiata riporta la stima corretta della pdf, infine la linea nera tratteggiata riporta la stima errata della pdf in cui la discrepanza riportata mostra l'effettivo effetto della censura, infatto vediamo di come inizialmente tende a sovrastimare il tasso di guasto fino a circa il valore di 30.000, successivamente a questo valore, invece, vediamo una sottostima del guasto. Dai risultati ottenuti vediamo come effettivamente la procedura di stima corretta risulta essere più affidabile rispetto al caso in esame.

Capitolo 5

5 Analisi di disponibilità

5.1 Approfondimento sull'Availability

Procediamo con un'analisi più approfondita della disponibilità e della sua ottimizzazione, concentrandoci in particolare sui sistemi critici per la sicurezza.

L'obiettivo primario di progettare sistemi con un'elevata disponibilità è di importanza capitale. La continuità della disponibilità rappresenta un requisito imprescindibile per garantire il corretto funzionamento dei sistemi moderni. La nostra analisi si focalizzerà su strategie e approcci volti a massimizzare la disponibilità in contesti critici per la sicurezza, contribuendo così a garantire la robustezza e l'affidabilità di tali sistemi nell'era della trasformazione digitale.

5.1.1 Ottimizzazione della Configurazione di Ridondanza

Per conseguire un elevato livello di disponibilità, è essenziale considerare l'implementazione della ridondanza nei sistemi. La sfida principale consiste nell'identificare una configurazione che soddisfi un obiettivo di disponibilità pari a 0.999999, noto come "six nines", minimizzando al contempo il costo totale c_{tot}. La ridondanza implica la replicazione dell'intero sistema, che sia S1, S2 o S3 (anche se dimostrato che la ridondanza di componente,rispetto a quella di sistema, conduce a risultati migliori), ognuno con costi normalizzati rispettivamente a $c_1 = 0.2$, $c_2 = 0.3$,$c_3 = 1$. Per affrontare questa sfida, abbiamo implementato un algoritmo di ottimizzazione in Python che modella la configurazione del sistema e identifica il set ottimale di componenti. Questa configurazione, minimizzando la somma complessiva dei costi dei componenti ridondati, introduce la giusta quantità di ridondanza nei sottosistemi esistenti per raggiungere il livello desiderato di disponibilità. La classe Subsystem gestisce le proprietà individuali di ciascun sottosistema, come nome, costo e disponibilità. Le classi SerieSubsystem e ParallelSubsystem estendono questa funzionalità per gestire sistemi in serie o in parallelo, calcolando la disponibilità complessiva e il costo totale associato.

La classe *AvailabilityOptimizer* contiene l'algoritmo di ottimizzazione che, attraverso un approccio ricorsivo, esplora le possibili configurazioni di ridondanza per ogni sottosistema e determina la combinazione che raggiunge il target di disponibilità desiderato al minimo costo. La funzione *getOptimalConfiguration* seleziona la configurazione più economica tra quelle che soddisfano i requisiti di disponibilità.

5.1.2 L'Algoritmo di Ottimizzazione per la Disponibilità

La strategia di ottimizzazione adottata è rappresentata dall'algoritmo implementato in Python, il quale si basa su una ricerca esaustiva delle configurazioni

di ridondanza. Dovendo affrontare la complessità del problema, abbiamo adottato un approccio ricorsivo che esamina ogni possibile combinazione di sistemi ridondanti.Durante l'esecuzione, valutiamo la disponibilità complessiva di ciascuna combinazione, confrontandola con il nostro obiettivo di raggiungere six nines.

Nel corso dell'algoritmo, facciamo uso delle strutture dati Subsystem, SerieSubsystem e ParallelSubsystem, che rappresentano i vari componenti e le loro interazioni. Questi oggetti semplificano non solo il calcolo della disponibilità aggregata ma anche la somma dei costi associati alle diverse configurazioni di ridondanza. Un aspetto particolarmente interessante dell'algoritmo è la sua flessibilità nel modificare la ricerca in base ai risultati intermedi, consentendo una significativa riduzione dello spazio di ricerca e dei tempi di esecuzione.

Il processo di ottimizzazione è stato articolato in fasi distinte, ciascuna mirante a:

- *Generazione delle Configurazioni:* Partendo da una configurazione base senza ridondanza, l'algoritmo genera tutte le possibili combinazioni di ridondanza per i sottosistemi, rispettando il vincolo massimo di replicazione imposto dalla variabile $MAX_{REDUNDANCY}$.

- *Valutazione delle Configurazioni:* Ogni configurazione generata viene valutata in termini di disponibilità e costo totale. Questo consente di filtrare le soluzioni non idonee che non soddisfano il criterio di availability desiderato o che superano il limite di costo

- *Selezione della Configurazione Ottimale:* Tra le configurazioni idonee, l'algoritmo identifica quella con il costo minore, utilizzando la funzione getOptimalConfiguration. Questo passaggio è cruciale per garantire che l'investimento in ridondanza sia giustificato dal miglioramento sostanziale nella disponibilità del sistema.

Proseguendo l'analisi dell'availability nei sistemi critici per la sicurezza, esaminiamo il codice Python fornito, essenziale per questa ottimizzazione.

5.1.3 Descrizione delle Classi Python

Classe *Subsystem*
Metodi utilizzati:

```
class Subsystem:
    # Rappresenta un sottosistema con nome, costo e disponibilita'.
    def __init__(self, name, cost, availability: float):
        # Costruttore per la classe Subsystem
        self.name = name
        self.cost = cost
        self.availability = availability

    def __str__(self):
```

```
        return self.name

    def __repr__(self):
        return self.name

    def setName(self, name):
        # Setta il nome del sottosistema
        if isinstance(name, str):
            self.name = name

    def setCost(self, cost):
        # Setta il costo del sottosistema
        if isinstance(cost, int) or isinstance(cost, float):
            self.cost = cost

    def setAvailability(self, availability):
        # Setta la metrica di disponibilita' del sottosistema come
            float
        if isinstance(availability, int) or isinstance(availability,
            float):
            self.availability = availability

    def getName(self):
        return self.name

    def getCost(self):
        return self.cost

    def getAvailability(self):
        return self.availability

class SerieSubsystem(Subsystem):
    # Classe che eredita le caratteristiche della superclasse Subsystem
    # ma con una funzionalita' extra: l'aggiunta di un indice di serie
    def __init__(self, name, subsystems: [Subsystem]):
        # Costruttore per la classe SerieSubsystem
        self.subsystems = subsystems
        temp_cost = 0
        temp_availability = 1
        for subsystem in subsystems:
            temp_cost += subsystem.cost
            temp_availability *= subsystem.availability
        super().__init__(name, temp_cost, temp_availability)

    def addSubsystem(self, subsystem):
        self.subsystems.append(subsystem)
```

```
        self.cost += subsystem.cost
        self.availability *= subsystem.availability

    def removeSubsystem(self, subsystem):
        self.subsystems.remove(subsystem)
        self.cost -= subsystem.cost
        self.availability /= subsystem.availability

    def getSubsystems(self) -> [Subsystem]:
        return self.subsystems

    def getAvailability(self):
        self.availability = 1
        for subsystem in self.subsystems:
            self.availability *= subsystem.getAvailability()
        return self.availability

class ParallelSubsystem(Subsystem):
    # Classe che modella un sottosistema parallelo
    def __init__(self, name, subsystems: [Subsystem]):
        self.subsystems = subsystems
        temp_cost = 0
        temp_availability = 0
        for subsystem in subsystems:
            temp_cost += subsystem.cost
            temp_availability = 1 - (1 - subsystem.availability)
        super().__init__(name, temp_cost, temp_availability)

    def addSubsystem(self, subsystem):
        self.subsystems.append(subsystem)
        self.cost += subsystem.cost
        self.availability = 1 - (1 - self.availability) * (1 -
            subsystem.availability)

    def removeSubsystem(self, subsystem):
        self.subsystems.remove(subsystem)
        self.cost -= subsystem.cost
        self.availability = 1 - (1 - self.availability) / (1 -
            subsystem.availability)

    def getSubsystems(self) -> [Subsystem]:
        return self.subsystems

    def getAvailability(self):
        self.availability = 1 - (1 - self.subsystems[0].getAvailability
            ()) ** len(self.subsystems)
```

```
97          return self.availability
```

Listing 5: Descrizione del tuo codice

La presenza delle classi *SubSystem, SerieSubsystem e ParallelSubsystem* permettono di modellare le configurazioni in serie e in parallelo in modo da facilitare il calcolo della disponibilità. Dalla teoria sappiamo che nel caso della configurazione in serie, la disponibilità totale è il prodotto delle singole disponibilità dei singoli sottosistemi. Nella configurazione in parallelo, invece, la disponibilità viene calcolata considerando la ridondanza dei sottosistemi.

5.1.4 Descrizione della classe *AvailabilityOptimizer*

L'obiettivo di questa classe è quello di fornire degli algoritmi per ottimizzare l'Availability del sistema. Sono stati implementati degli algoritmi statici in modo da trovare la migliore configurazione di sottosistemi per raggiungere la disponibilità desiderata.

```
1  from Subsystem import *

3  class AvailabilityOptimizer:
       counter=0
5      @staticmethod
       def availabilityOptimizationAlgorithmRec(subsystem,
          desired_availability=0.999999, max_redundancy=3):
7          # metodo che implementa un algoritmo ricorsivo per ottimizzare
              la disponibilita' del sottosistema .
           #La funzione ricorsiva trova la configurazione ottimale del
              sottosistema per ottenere la disponibilita' desiderata.
9          #Esplora iterativamente le differenti combinazioni di
              ridondanza di ogni sottosistema. I parametri che analizza
              sono:
           #l'instanza del sottosistema serie da ottimizzare, l'
              availability desiderata e il massimo numero di ridondanza
              consentito per ogni sottosistema.
11         #Ritorna una lista di configutrazioni, ognuna e' una lista di
              tuple con un sottosistema e il suo costo di ridondanza.

13         def recursiveOptimization(current_index, current_configuration,
               current_availability):
               if current_index == len(subsystem.subsystems):
15                 if current_availability >= desired_availability:
                       opt_list.append(AvailabilityOptimizer.
                          _buildConfiguration(current_configuration.copy()
                          ))
17                 return
               for redundancy in range(1, max_redundancy + 1):
```

```
19                  new_availability = 1 - (1 - subsystem.subsystems[
                        current_index].availability) ** redundancy
                    if current_index == 0:
21                      updated_availability = new_availability
                    else:
23                      updated_availability = current_availability *
                            new_availability

25                  current_configuration.append((subsystem.subsystems[
                        current_index], redundancy))
                    recursiveOptimization(current_index + 1,
                        current_configuration, updated_availability)
27                  current_configuration.pop()
            opt_list = []
29          recursiveOptimization(0, [], 1)
            return opt_list
31
        @staticmethod
33      def getOptimalConfiguration (opt_list : [(Subsystem, int)]) -> [(
            Subsystem, int)]: # type: ignore
            #Questo metodoo statico determina la configurazione con il piu'
                basso costo da una lista di potenziali configurazioni.
35          #Questo metodo compara configurazioni differenti basate sul
                loro costo e ritorna quello con il costo migliore.
            #In questo contesto i parametri che prende sono una lista di
                configurazioni potenziali e ritorna una tupla contenente la
                configurazione
37          #del sottosistema e le loro ridondanze rispettive.
            if len (opt_list) == 0:
39              return None
            else:
41              best_configuration = opt_list [0]
                for configuration in opt_list:
43                  if configuration.getCost() < best_configuration.getCost
                        ():
                        best_configuration = configuration
45              return best_configuration

47      @staticmethod
        def _buildConfiguration (configuration) -> SerieSubsystem:
49          #Metodo statico per costruitre un istanza di SerieSubsystem da
                una data configurazione.
            #Questo metodo costruisce un SerieSubsystem da una
                configurazione che specifica la ridondanza per ogni
                Subsystem nella serie.
51          #In questo contesto i parametri che prende sono una lista di
```

```
            configurazioni(tuples) ognuna contenenti un Sottosistema e
               il suo indice di ridondanza.
        #Ritorna un istanza di SerieSubsystem rappresentando la
           configurazione specifica.

        serie_subsystem = SerieSubsystem ("" , [])
        name = ""
        AvailabilityOptimizer.counter += 1
        for subsystem, redundancy in configuration :
            parallel_subsystem = ParallelSubsystem("", [])
            for i in range(redundancy):
                parallel_subsystem.addSubsystem(Subsystem(str(subsystem
                   ), subsystem.getCost(), subsystem.getAvailability())
                   )
                parallel_subsystem.setName(str(subsystem))
            name += str(subsystem) + "_"
            serie_subsystem.addSubsystem(parallel_subsystem)
        name += str(AvailabilityOptimizer.counter)
        serie_subsystem.setName(name)
        return serie_subsystem

    @staticmethod
    def availabilityOptimizationAlgorithm(subsystem : SerieSubsystem ,
       desired_availability = 0.999999, max_redundancy = 3) -> [(
       Subsystem , int)]:
    #Ottimizza la ridondnza di un sistema serie consistente esattamente
        di tre elementi chiave per raggiungere una availability
       desiderata.
    #Questo metodo statico calcola la ridondanza ottimale per ognuno
       dei tre sottosistemi in una configurazione serie per trovare l'
       availability desiderata.
    #La funzione itera lungo possibili libelli di ridondanza per ognuno
        dei sottosistemi e calcola l'availability totale del sistema.
       Il metodo e' specificatamente designatoo
    #per sistemi composti di esattamente tre elementi ed e' meno
       generico rispetto ad un approccio ricorsiva che gestisce i
       sisteemi con un numero arbitrario
    #di serie e parallelo. Notiamo che questo metodo e' applicabile
       solamente ad instanze di SerieSubsystem con tre elementi.
    #I parametri che il metodo prende sono una lista di configurazioni
       di sottosistemi, che nel caso specifico sono tre, poi una
       disponibilita' desiderata, per il
    #sistema serie come float, la quale e' impostata di default a
       0.999999.Infine viene ritornata una lista di tuple , ognuna
       contenenete un sottosistema ed il
    #il suo livello di ridondanza , per raggiungere la disponibilita'
```

```
        desiderata.
        opt_list = []
        for i in range(1, max_redundancy + 1):
            for j in range(1, max_redundancy + 1):
                for k in range(1, max_redundancy + 1):
                    avail_i = 1 - (1 - subsystem.subsystems[0].
                        availability) ** i
                    avail_j = 1 - (1 - subsystem.subsystems[1].
                        availability) ** j
                    avail_k = 1 - (1 - subsystem.subsystems[2].
                        availability) ** k
                    if avail_i * avail_j * avail_k >=
                        desired_availability:
                        opt_list.append([(subsystem.subsystems[0], i),
                            (subsystem.subsystems[1], j), (subsystem.
                            subsystems[2], k)])
        return opt_list
```

Listing 6: Descrizione del tuo codice

I metodi statici hanno il compito di evidenziare un approccio che sia funzionale e riutilizzabile. La funzione *availabilityOptimizationAlgorithmRec* esplora in profondità tutte le combinazioni di ridondanza.

5.1.5 Esecuzione del *main*

```
from Subsystem import  Subsystem, SerieSubsystem
from Availability import AvailabilityOptimizer


if __name__ == "__main__":
    DESIRED_AVAILABILITY = 0.999999
    MAX_REDUNDANCY = 3
    S1 = Subsystem("S1", 0.2, 9.99610403e-001)
    S2 = Subsystem("S2", 0.8, 9.98426217e-001)
    S3 = Subsystem("S3", 1, 9.98845783e-001)


    net = SerieSubsystem("S1_S2_S3", [S1, S2, S3])
    opt_list = AvailabilityOptimizer.
        availabilityOptimizationAlgorithmRec(net,DESIRED_AVAILABILITY ,
        MAX_REDUNDANCY)
    with open("lista_configurazioni_ottimali.txt", "w") as f:
        f.write("Configurazioni possibili per l'availability desiderata
            = " + str(DESIRED_AVAILABILITY) + " e massima ridondanza =
           " + str(MAX_REDUNDANCY) + "\n")
```

```
        for opt in opt_list:
            f.write("\nNome:" + str(opt))
            f.write("\nConfigurazione: [")
            for elem in opt.getSubsystems():
                f.write("(" + str(elem.getName()) + ", " + str(len(elem
                    .getSubsystems())) + "), ")
            f.write("]")
            f.write("\nCosto: " + str(opt.getCost()) + "\n
                Disponibilita' stazionaria: " + str(opt.getAvailability
                ()) + "\n")
    print("\nConfigurazioni possibili per l'availability desiderata
        scritte nel file lista_configurazioni_ottimali.txt:\n")
    optimal_configuration = AvailabilityOptimizer.
        getOptimalConfiguration(opt_list)
    print("\nConfigurazione con il costo minore\n Nome: " + str(
        optimal_configuration))
    print(" Configurazione: [", end="")
    for elem in optimal_configuration.getSubsystems():
        print("(" + str(elem.getName()) + ", " + str(len(elem.
            getSubsystems())) + "),", end="")
    print("]")
    print(" Costo: " + str(optimal_configuration.getCost()) +"\n
        Disponibilita' stazionaria: " + str(optimal_configuration.
        getAvailability()), end="\n\n")
```

Listing 7: Descrizione del tuo codice

Dall'analisi ottenuta attraverso l'esecuzione del seguente algoritmo, abbiamo identificato la soluzione presente nella figura sottostante.

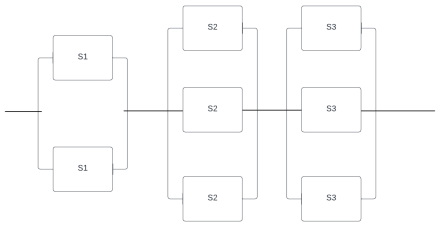

Figure 17: Configurazione miglior ridondanza ottenuta per garantire i six nines

Un quadro completo delle possibili combinazioni è presente nel file *.txt* del file python. Tra le proposte, abbiamo scelto la configurazione che presenta queste caratteristiche:
Nome: $S1\_S2\_S3$
Configurazione: [(S1, 2),(S2, 3),(S3, 3)]
Costo: 5.800000000000001

Disponibilita' stazionaria: 0.9999998427785765

5.2 Conclusioni

L'esecuzione dell'algoritmo risulta essere efficace per gli obiettivi da raggiungere. Infatti tale soluzione ha mostrato come migliorare l'affidabilità del sistema con l'aggiunta di ridondanze, senza però rischiare di eccedere nei costi.

Capitolo 6

6 Analisi di sensitività

Il seguente capitolo si pone come obiettivo quello di mostrare e applicare il concetto di Analisi di Sensitività. L'analisi di sensitività è una tecnica utilizzata nell'ambito della modellazione e dell'analisi dei sistemi per valutare come le variazioni nei parametri di un modello influenzano gli output o le prestazioni del sistema. In altre parole, l'analisi di sensitività si occupa di comprendere come cambiamenti nei dati in ingresso, nei parametri del modello o in altre variabili influenzino i risultati del modello o del sistema.

Nel nostro caso specifico, è stata condotta un'analisi di sensitività centrata su due parametri principali: λ_{DCK} e μ_{DCK}. L'obiettivo è quello di valutare l'impatto della variazione di questi due parametri sulla disponibilità stazionaria del sistema e successivamente è stato identificato il valore critico di ciascun parametro. Volendo precisare, diciamo che viene considerato valore critico quel punto oltre il quale la disponibilità stazionaria del sistema scende oltre la soglia fissata che sappiamo essere una soglia a six nines.

6.1 Procedura utilizzata

Al fine di rendere tale approccio riutilizzabile si è deciso di implementare delle funzioni Python che interaggissero in modo diretto con il software Sharpe. Il primo passaggio effetuato è stato quello di costruire il sistema all'interno di script python. Tale processo è stato effettuato recuperando gli script visti nel paragrafo 5 effettuando le dovute modifiche per il numero di sistemi ridondanti che abbiamo ritenuto necessari nelle conclusioni del capitolo precedente. In particolare, l'intero processo si articola nelle seguenti fasi:

1. costruzione del sistema fissata la configurazione con elementi ridondanti della rete e definizione della soglia di disponibilità stazionaria
2. configurazione e avvio della procedura di analisi
3. generazione e salvataggio del grafico finale a partire dai risultati dell'analisi

6.1.1 Costruzione del sistema

A partire dalle classi Subsystem, SerieSubsystem e ParallelSubsystem già presentate nel Capitolo 5, si è costruito l'intero sistema inserendo ciascun sottosistema (S1, S2 e S3) con il grado di ridondanza ottenuto dall'analisi di disponibilità. Ciò è implementato dalle seguenti righe di codice

```
1 from Subsystem import *
  from build import *
3
  if __name__ == "__main__":
```

```
    #costruzione del sistema ridondante con S1, S2, S3 ognuno con il
       proprio grado di ridondanza, ottenuto nell'analisi precedente.
    S1_RED_NUM = 2 #grado di ridondanza del sistema S1
    S2_RED_NUM = 3 #grado di ridondanza del sistema S2
    S3_RED_NUM = 3 #grado di ridondanza del sistema S3

    #creazione dei sottosistemi
    S1 = Subsystem("S1", 0.2, 9.99610403e-001)
    S2 = Subsystem("S2", 0.8, 9.98426217e-001)
    S3 = Subsystem("S3", 1, 9.98845783e-001)

    STEADY_STATE_DESIRED_AVAILABILITY = 0.999999 # soglia di
       disponibilita' stazionaria

    # creazione del sistema complessivo tramite  l invocazione  della
       funzione build_system che prende in input un dict di coppie key:
       value,
    # dove key rappresenta il sottosistema da inserire e value il
       numero di repliche
    system = build_system ({S1: S1_RED_NUM, S2 : S2_RED_NUM , S3:
       S3_RED_NUM})
    #Ottenuto quindi  l oggetto  rappresentante  l intero  sistema, e'
       possibile passare
    # alla definizione della configurazione che dovra' essere poi
       passata  a l l analyzer .
```

Listing 8: Descrizione del tuo codice

In particolare nel seguente codice viene prima definito il grado di ridondanza di ciascun sottosistema, successivamente avviene la creazione di ogni sottosistema, ognuno con il proprio nome, costo e disponibilità stazionaria. Infine nella parte finale si ha l'effettiva creazione del sistema complessivo tramite l'invocazione della funzione buildsystem che prende in input un dizionario di coppie chiave-valore, dove chiave rappresenta il sottosistema da inserire e value il numero di repliche. L'ultima riga definisce la disponibilità stazionaria. Di seguito l'implementazione della funzione build-system

```
from Subsystem import *
from analyzer import *
#Inizializza un oggetto system di tipo SerieSubsystem, che rappresenta
   un insieme di sottosistemi collegati in serie. Successivamente,
# itera attraverso ogni coppia chiave-valore nel dict configuration e,
   per ogni sottosistema, crea un ParallelSubsystem popolato con copie
   multiple del
#sottosistema originale, basato sul valore associato nel dict. Allo
   stesso tempo, aggiorna il nome del sistema con una concatenazione
   dei nomi dei sottosistemi, restituendo, infine,  l oggetto  system
```

```
    costruito, che ora rappresenta la struttura complessiva del sistema
    basato sulla configurazione fornita
def build_system (configuration : {Subsystem, int}):
    system = SerieSubsystem ('', [])
    name = ""
    for i, elem in enumerate (configuration.items()):
        key, value = elem
        temp = ParallelSubsystem (key.getName(), [])
        name += str (key.getName())
        for j in range (value):
            temp.addSubsystem (key)
            system.addSubsystem (temp)
        system.setName(temp)
    return system
```

Listing 9: Descrizione del tuo codice

In prima battuta, inizializza un oggetto system di tipo SerieSubsystem, che rappresenta un insieme di sottosistemi collegati in serie. Successivamente, itera attraverso ogni coppia chiave-valore nel dict configuration e, per ogni sottosistema, crea un ParallelSubsystem popolato con copie multiple del sottosistema originale, basato sul valore associato nel dict. Allo stesso tempo, aggiorna il nome del sistema con una concatenazione dei nomi dei sottosistemi, restituendo, infine, l'oggetto system costruito, che ora rappresenta la struttura complessiva del sistema basato sulla configurazione fornita. In prima battuta, inizializza un oggetto system di tipo SerieSubsystem, che rappresenta un insieme di sottosistemi collegati in serie. Successivamente, itera attraverso ogni coppia chiave-valore nel dict configuration e, per ogni sottosistema, crea un ParallelSubsystem popolato con copie multiple del sottosistema originale, basato sul valore associato nel dict. Allo stesso tempo, aggiorna il nome del sistema con una concatenazione dei nomi dei sottosistemi, restituendo, infine, l'oggetto system costruito, che ora rappresenta la struttura complessiva del sistema basato sulla configurazione fornita.

6.1.2 Configurazione e avvio della procedura di analisi

Ottenuto quindi l'oggetto rappresentante l'intero sistema, è possibile passare alla definizione della configurazione che dovrà essere poi passata all'analyzer. In particolare, è necessario specificare:

- RATE-TYPE, che può assumere i valori 'failure' o 'repair'
- TRANSITION-NAME, che indica il nome della transizione temporizzata sotto analisi (nel nostro caso, 'lambdaDCK' e 'muDCK')
- TRANSITION-VALUE-LOWER-BOUND, che indica il più piccolo valore (in ore), in termini di MTTF (o MTTR), da associare alla transizione

- TRANSITION-VALUE-UPPER-BOUND, che indica il più grande valore (in ore), in termini di MTTF (o MTTR), da associare alla transizione
- l TRANSITION-VALUE-STEP, che indica il passo con cui spostarsi dal lower bound all'upper bound

```
#TRANSITION_NAME indica il nome della transizione temporizzata sotto analisi
TRANSITION_NAME = 'lambda_DCK'
#TRANSITION_VALUE_LOWER_BOUND indica il valore minimo in ore del valore minimo di MTTF o MTTR da associare alla transizione.
TRANSITION_VALUE_LOWER_BOUND = 1
#TRANSITION_VALUE_UPPER_BOUND indica il valore massimo in ore del valore minimo di MTTF o MTTR da associare alla transizione
TRANSITION_VALUE_UPPER_BOUND = 1500
#indica il passo con cui spostarsi dal lower bound  a l l upper bound
TRANSITION_VALUE_STEP = 0.1
configuration = {'TRANSITION_NAME': TRANSITION_NAME, '
   TRANSITION_VALUE_LOWER_BOUND': TRANSITION_VALUE_LOWER_BOUND, '
   TRANSITION_VALUE_UPPER_BOUND': TRANSITION_VALUE_UPPER_BOUND,'
   TRANSITION_VALUE_STEP': TRANSITION_VALUE_STEP, '
   STEADY_STATE_DESIRED_AVAILABILITY':
   STEADY_STATE_DESIRED_AVAILABILITY}
#Terminata la fase di configurazione,    stata invocata la funzione
    sensitivity_analyzer che prende in input come parametri: il sistema
# complessivo, il sottosistema sotto analisi, il file contenente
    l  opportuno  codice sharpe per il calcolo della
# disponibilit  stazionaria del sottosistema in esame e la
   configurazione precedentemente definita. Al termine della sua
   esecuzione,
# tale funzione restituir  una lista di tuple del tipo (
   transition_value, new_availability, boolean_value). In
   particolare, boolean_value    uguale a
# True per le coppie (transitizon_value, new_availability) in cui
   new_availability    al di sotto di quella fissata, False
   altrimenti.
out = sensitivity_analyzer (system, S2, '
   S2_steady_state_avail_config',configuration)
TRANSITION_NAME = 'mu_DCK'
TRANSITION_VALUE_LOWER_BOUND = 1/3600
TRANSITION_VALUE_UPPER_BOUND = 20000/3600
TRANSITION_VALUE_STEP = 0.5/3600
configuration = {'TRANSITION_NAME': TRANSITION_NAME, '
   TRANSITION_VALUE_LOWER_BOUND': TRANSITION_VALUE_LOWER_BOUND, '
```

```
        TRANSITION_VALUE_UPPER_BOUND': TRANSITION_VALUE_UPPER_BOUND,'
        TRANSITION_VALUE_STEP': TRANSITION_VALUE_STEP, '
        STEADY_STATE_DESIRED_AVAILABILITY':
        STEADY_STATE_DESIRED_AVAILABILITY}
21  out = sensitivity_analyzer(system, S2, '
        S2_steady_state_avail_config',configuration)
```

Listing 10: Configurazione

Come mostrato all'interno del codice, la configurazione utilizzata per l'analisi di sensitività rispetto al failure rate è

RATE-TYPE	failure
TRANSITION-NAME	lambdaCDK
TRANSITION-NAME-LOWER-BOUND	1
TRANSITION-NAME-UPPER-BOUND	1500
TRANSITION-VALUE-STEP	0.1

Table 9: Parametri che caratterizzano l'unità riparabile Sensore

mentre, quella utilizzata per l'analisi di sensitività rispetto al repair rate è

RATE-TYPE	repair
TRANSITION-NAME	muCDK
TRANSITION-NAME-LOWER-BOUND	1/3600
TRANSITION-NAME-UPPER-BOUND	20000/3600
TRANSITION-VALUE-STEP	0.5/3600

Table 10: Parametri che caratterizzano l'unità riparabile Sensore

Terminata la fase di configurazione, è stata invocata la funzione sensitivity-analyzer che prende in input come parametri: il sistema complessivo, il sottosistema sotto analisi, il file contenente l'opportuno codice sharpe per il calcolo della disponibilità stazionaria del sottosistema in esame e la configurazione precedentemente definita. Al termine della sua esecuzione, tale funzione restituirà una lista di tuple del tipo (transition_value, new_availability, boolean_value).

In particolare, boolean_value è uguale a True per le coppie (transition_value, new_availability) in cui new_availability è al di sotto di quella fissata, False altrimenti. L'implementazione della funzione *sensitivity_analyzer* è di seguito riportata:

```
import os
from sys import stdout
from asyncio import *
import re
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.ticker import FormatStrFormatter
from Subsystem import *
import shutil
import subprocess
def sensitivity_analyzer (system, subsystem, filename, configuration):
    filename_temp = filename + '_temp' #creazione file temporaneo
    shutil.copy (filename, filename_temp) #copia dei dati del file
       originale in quello temporaneo
    out_points = []  #inizializzazione lista vuota per la
       memorizzazione dei risultati.
    #esecuzione di un ciclo for con passo di TRANSITION_STEP da
       TRANSITION_VALUE_LOWER_BOUND a TRANSITION_VALUE_UPPER_BOUND.
    for transition_value in np.arange (configuration ['
       TRANSITION_VALUE_LOWER_BOUND'], configuration ['
       TRANSITION_VALUE_UPPER_BOUND'], configuration ['
       TRANSITION_VALUE_STEP']):
        _update_file (filename_temp, configuration['TRANSITION_NAME'],
           transition_value) #aggiorniampo il file sharpe temporaneo
           con il nuovo valore di MTTF/MTTR
        _compute_availability (subsystem, filename_temp) #calcoliamo la
            nuova disponibilita' del sistema
        new_availability = system.getAvailability () #otteniamo la
           nuova disponibilita' del sistema
        if new_availability < configuration ['
           STEADY_STATE_DESIRED_AVAILABILITY']: #determiniamo se la
           nuova disponibilita' del sistema    superiore o inferiore
           alla soglia desiderata
            out_points.append ((transition_value, new_availability,
               True)) #se inferiore alla soglia desiderata viene
               impostato il flag a True
        else:
            out_points.append ((transition_value, new_availability,
               False)) #altrimenti al valore False
    os.remove (filename_temp) # rimoviamo il file temporaneo
    return out_points #restituzione della lista contenente i risultati
```

```
        delle analsi.
```

Listing 11: Implementazione sensivity_analyzer

Infine, viene rimosso file temporaneo con .remove(filename_temp) e restituita la lista out_points contenente i risultati dell'analisi. Come anticipato, la funzione appena descritta fa uso delle procedure _update_file e _compute_availability. La prima, di seguito riportata:

```
def _update_file (filename, transition_name, transition_value):
    with open (filename, 'r') as f: #apre il file in modalita' lettura
       e legge tutte le sue linee memorizzandole nella variabile "lines
       "
        lines  = f.readlines()
    for i, line in enumerate (lines): #effettua un'iterazione su ogni
       riga, controlla se ciascuna inziia con "transition_name".
        if line.startswith (transition_name): #In caso affermativo
           sostisuisce quella riga on una nuova stringa che contine il
           nome della transizione seguito da 1/ ed il nuovo
            #valore di transition_value, ovvero MTTR/MMTF.
            lines[i] = str(transition_name) + '1/' + str (
               transition_value) + '\n'
    with open (filename, 'w') as f: #infine lo stesso file viene aperto
        in modalita' di scrittura per scrivere tutte le linee, adesso
       aggionrate.
        f.writelines(lines)
```

Listing 12: Implementazione funzione udate_file

opera nel seguente modo: come prima azione viene aperto il file in modalità lettura e vengono lette tutte le sue linee, memorizzandole nella variabile "lines". Successivamente avviene un'iterazione su ogni riga del file, e per ognuna di queste controlla se inizia con un transition_name. In caso affermativo viene sostituita quella riga con una nuova riga che contiene il nome della transizione seguita da *1/* e il nuovo transition_value(MMTF/MMTR). Infine viene riaperto lo stesso file in modalità scrittura e vengono scritte tutte le linee, aggiornate, nel file. Il secondo metodo viene riportatao di seguito:

```
def _compute_availability  (subsystem, filename):
    file_path = filename  # Sostituisci questo con il nome effettivo
       del file
    absolute_path = os.path.abspath(file_path)
    print("Il percorso completo del file e':", absolute_path)
    process = subprocess.run([absolute_path], capture_output=True) #
       viene lanciato lo script SHARPE, usando il file precedentemente
       aggiornato con il nuovo valore di MTTF/MTTR
    output = process.stdout
```

```
    #stdout.decode ('utf-8') #poiche' l'output catturato e' in formato
       binario questo deve essere decodificato in una stringa UTF-8 per
        poter essere utilizzato.
    match = re.search (r"SS_ExpectedrewardRate: \s*(\d\.e-)+", output)
       #Successivamente vengono utilizzati dei regex per cercare una
       stringa specifica nell'output in particolar modo quella
       indicante il nuovo valore di disponibilita' calcolato.
    value = match.group (1) if match else None #Viene quindi estretto
       il valore numerico
    subsystem.setAvailability (float(value)) #e viene usato per
       aggiornare la disponibilita' del sottosistema.
```

Listing 13: Implementazione funzione _compute_availability

effettua le seguenti operazioni. Inizialmente, viene lanciato lo script sharpe utilizzando il file precedentemente aggiornato con il nuovo valore di MTTF/MTTR, al fine di ottenere il nuovo valore di disponibilità del sottosistema sotto analisi. L'output catturato, che è in formato binario, viene decodificato in una stringa UTF-8 per poter essere utilizzato. La funzione poi utilizza le espressioni regolari (regex) per cercare una stringa specifica nell'output, ovvero, quella che indica il nuovo valore di disponibilità stazionaria calcolato. Il valore numerico viene quindi estratto e usato per aggiornare la disponibilità del sottosistema, attraverso il metodo setAvailability

6.1.3 Generazione del grafico a partire dai risultati dell'analisi

Terminata la fase di analisi, al fine di mostrare l'andamento della disponibilità stazionaria al variare del parametro in esame, è stata implementata la funzione plot, di seguito riportata:

```
#la funzione per plottare i grafici oltre a tracciare il grafico sulla
   base dei valori di output dell'analisi , inserisce sia una retta
   rappresentativa del valore nimnale del parametro considerato, sia un
    simbolo,
#ovvero nel nostro caso una stella, rappresentante il punto che ha come
    ascissa il valore critico del parametro a partire dal quale la
   disponiblita' stazionaria scende
#al di sotto di quella fissat e ecome orfinata il valroe della
   disponibilita' stesso. Al fine diottenere tale punto critico la
   funzione plot si avvale della procedura
#find_critical_index che trova l'indice del valore creitico nie
   risutlati dell'analisi di sensibilita'.
def plot (out, rate_type, nominal_value , title, x_label = "X-axis",
   y_label = "Y_axis", x_scale = "linear", y_scale = "linear"):
    x, y, z = zip(*out)
    x,y = list (x), list (y)
    plt.figure (figsize = (10, 6))
```

```
    plt.plot (x,y)
    plt.plot (np.ones_like(x) * nominal_value , y , linestyle = '--',
        color = 'yellow', label = 'Nominal value')
    critical_index = find_critical_index (z, rate_type)
    if critical_index is not None:
        x_critical, y_critical = x [critical_index], y [critical_index]
        print (x_critical, y_critical)
        plt.scatter (x_critical, y_critical, marker = '*', s= 300, c= '
            red', label = 'Critical value = ' + str (x_critical) + "[h]"
            )
        plt.ylim (y_critical - 0.0000010 , 0.999999)
    else:
        plt.ylim(0.999997740, 0.999999900)
        plt.title(title)
        plt.xlabel(x_label)
        plt.ylabel(y_label)
        ax = plt.gca()
        ax.set_xlabel(f"${x_label}$")
        ay = plt.gca()
        ay.set_ylabel(f"${y_label}$")
        plt.grid(True)
        plt.xscale(x_scale)
        plt.yscale(y_scale)
        plt.gca().yaxis.set_major_formatter(FormatStrFormatter('%.7f'))
        plt.legend()
        plt.savefig('Results' + os.sep + title + '.png')
def find_critical_index(lst, rate_type):
 if rate_type == 'failure':
    if lst[-1] == True:
        return len(lst)- 1
    for i, elem in enumerate(lst):
        if elem is False:
            return i- 1 if i > 0 else None
 elif rate_type == 'repair':
    for i, elem in enumerate(lst):
        if elem is True:
            return i
 return None
```

Listing 14: Funzione per effettuare i plot

In particolare, oltre a tracciare il grafico sulla base del valori di output dell'analisi, inserisce sia una retta rappresentativa del valore nominale del parametro considerato, sia un simbolo (nel nostro caso, una stella) rappresentante il punto avente come ascissa il valore critico del parametro a partire dal quale la disponibilità stazionaria scende al di sotto di quella fissata, e come ordinata il valore di disponibilità stesso. Al fine di ottenere tale punto critico, la funzione plot si

avvale della procedura find_critical_index che trova l'indice del valore critico nei risultati dell'analisi di sensibilità.

```
def find_critical_index(lst, rate_type):
 if rate_type == 'failure':
    if lst[-1] == True:
        return len(lst)- 1
    for i, elem in enumerate(lst):
        if elem is False:
            return i- 1 if i > 0 else None
 elif rate_type == 'repair':
    for i, elem in enumerate(lst):
        if elem is True:
            return i
 return None
```

Listing 15: Implementazione _find_critical_index

6.2 Risultati

I risultati dell'analisi di sensitività effettuata sono illustrati nelle Figure 17 e 18 e dimostrano che il sistema mantiene una disponibilità superiore ai six nine (0.999999) anche quando i valori di $\lambda_{DCK} e \mu_{DCK}$ sono rispettivamente molto inferiori e superiori ai loro valori nominali.

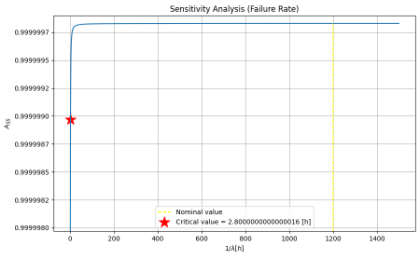

Figure 18: Analisi di sensitività sul parametro λ_{DCK}.

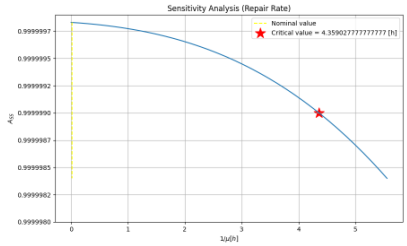

Figure 19: Analisi di sensitività sul parametro μ_{DCK}.

Specificamente, i dati rivelano che il reciproco del failure rate, $1/\mu_{DCK}$, può essere ridotto a circa 2.80 ore contro le 1200 ore nominali, e il sistema conserva ancora il rispetto del requisito di disponibilità stabilito. Analogamente, per il repair rate, il reciproco $1/\mu_{DCK}$, può essere esteso fino a circa 4.35 ore rispetto ai 30 secondi nominali, senza compromettere il livello di disponibilità richiesto. Questa tolleranza nel failure rate e nel repair rate suggerisce che il sistema docker ha un'ampia capacità di gestire guasti e malfunzionamenti senza influire negativamente sulla disponibilità complessiva. La ragione dietro questa robustezza può essere attribuita a diversi fattori, tra cui l'implementazione di misure di ridondanza. Infatti, in entrambi i casi, l'ampio margine è giustificato dal fatto che, nonostante la configurazione individuata sia quella a ridondanza minima, si ha una significativa superiorità della disponibilità steady-state del sistema rispetto al requisito dei six nines, essendo circa uguale a 0.9999998.

List of Figures