.DIEM

Dipartimento di Ingegneria dell'Informazione ed Elettrica e Matematica Applicata

Corso di Laurea in Ingegneria Informatica

Università degli Studi di Salerno

Corso di Laurea Magistrale in Ingegneria Informatica
Project Work per il corso di System Safety Engineering

Gruppo 2

Luigi Ferraioli *0622701853*

Dario Picone *0622701750*

Mario Petagna *0622701757*

Sommario

1. INTRODUZIONE

1.1 Service Function Chain (SFC)

La Service Function Chain (SFC) è un concetto e una tecnica utilizzata nelle reti di telecomunicazioni e nelle reti definite dal software per dirigere il traffico di rete attraverso una sequenza specifica di funzioni di rete, o “funzioni di servizio”, al fine di soddisfare i requisiti di servizio richiesti dalle applicazioni o dagli utenti finali.

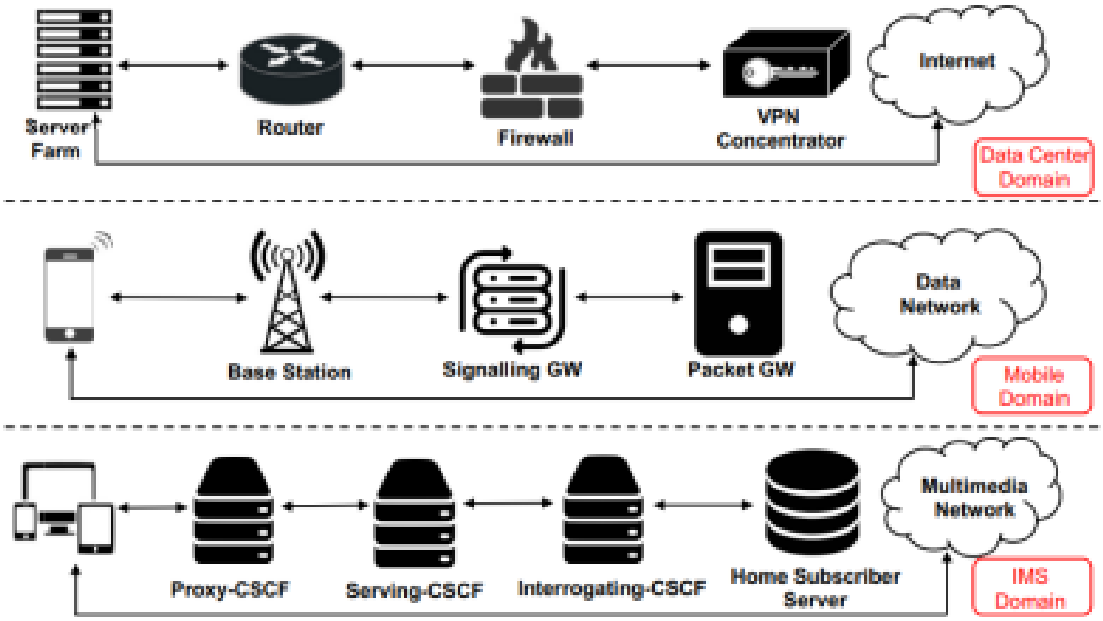


Figura 1, Service Function Chains

Una "funzione di servizio" può essere un'applicazione, un dispositivo o una funzionalità di rete che esegue un'azione specifica sul traffico di rete. Ad esempio, una funzione di servizio potrebbe essere un firewall, un bilanciatore di carico, un sistema di rilevamento delle intrusioni, un dispositivo di sicurezza, ecc. Ciascun dispositivo di rete richiede hardware dedicato e costoso per la sua implementazione. Al fine di risolvere il problema relativo al costo, le SFC introducono un approccio virtualizzato per le funzioni di servizio. In questo modo, al posto di avere funzioni di rete implementate su dispositivi hardware, le funzioni di servizio vengono virtualizzate e implementate su server o macchine virtuali.

1.2 Network Function Virtualization (NFV)

La Network Function Virtualization (NFV) è un concetto ed un paradigma che mira a trasformare le reti di telecomunicazioni tradizionali, spostando le funzioni di rete specializzate dai dispositivi hardware dedicati a software virtualizzati eseguiti su server standard.

Invece di utilizzare dispositivi hardware dedicati per eseguire funzioni di rete come firewall, router, switch, bilanciatori di carico, ottimizzatori di traffico e altre funzioni specializzate, la NFV consente di eseguire queste funzioni come software virtualizzati chiamati Virtual Network Functions (VNF) su server standard.

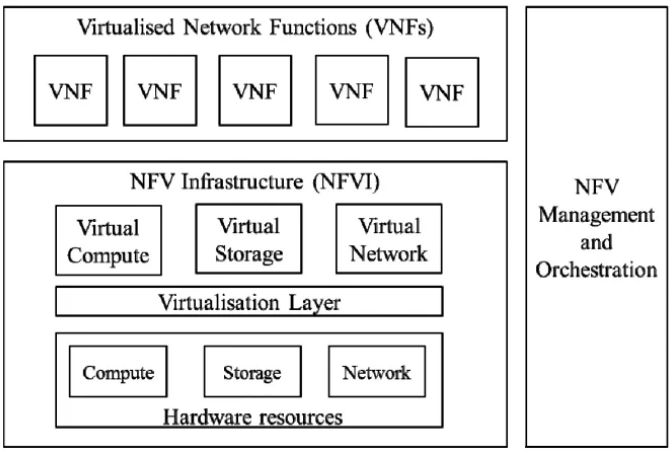


Figura 2: Modello dell'architettura NFV

Questo approccio offre una maggiore flessibilità, scalabilità ed efficienza per le reti.

Le VNF possono essere implementate, gestite e scalate in modo dinamico, consentendo alle reti di adattarsi rapidamente alle richieste di servizio e alle esigenze di traffico. Le funzioni di rete possono essere facilmente allocate e distribuite su server virtuali in base al carico di lavoro e possono essere ridimensionate orizzontalmente o verticalmente per soddisfare le esigenze di traffico in modo efficiente.

La NFV offre diversi vantaggi, tra cui:

1. Flessibilità: Le VNF possono essere facilmente attivate, disattivate o riallocate, consentendo una rapida implementazione di nuove funzioni di rete o servizi.

2. Scalabilità: Le VNF possono essere scalate in base alle esigenze del traffico di rete, sia in termini di capacità che di prestazioni.

3. Efficienza: L'utilizzo di server standard e l'allocazione dinamica delle risorse consentono un utilizzo efficiente delle risorse hardware e una riduzione dei costi operativi.

4. Agilità: La virtualizzazione delle funzioni di rete semplifica l'implementazione di nuove funzioni o servizi di rete, consentendo alle organizzazioni di essere più agili e reattive alle esigenze del mercato.

2. PRESENTAZIONE DEL MODELLO DI DISPONIBILITÀ

2.1 Descrizione del nodo containerizzato

Il nodo proposto nella traccia, raffigurato in Figura 3, sfrutta la tecnologia a container. Il nodo containerizzato presenta cinque strati (dal basso verso l'alto):

1. **Hardware (PHY):** ingloba tutti i componenti fisici;
2. **Hypervisor (HYP):** è un processo che consente il mapping di istruzioni di più macchine virtuali su una macchina fisica.
3. **Virtual Machine (VM):** è il software che crea un ambiente virtuale in grado di emulare il comportamento di una macchina fisica.
4. **Docker (DCK):** è il software che gestisce i container.
5. **Container (CNT):** è un ambiente software in cui sono isolati processi e applicazioni. Nel caso in esame i container sono tre.

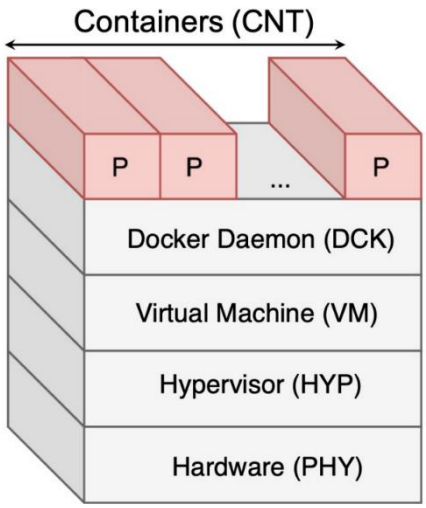


Figura 3: Nodo a 5 layer containerizzato

2.2 Modello SRN del nodo containerizzato

La SRN associata al nodo è rappresentata in Figura 4:

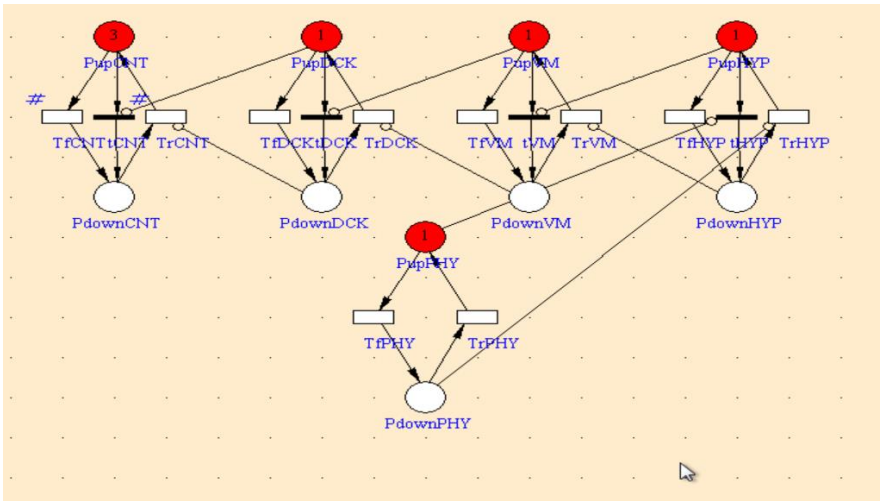


Figura 4: Stochastic Reward Network del nodo

Con questa rappresentazione si riesce a modellare:

- Attraverso i posti $\boldsymbol{P_{up}}$, le condizioni in cui i componenti sono funzionanti e attraverso i posti $\boldsymbol{P_{down}}$ le condizioni in cui i componenti sono guasti.
- Attraverso le transizioni timed $\boldsymbol{T_f}$ il passaggio da uno stato di corretto funzionamento ad uno stato di guasto (failure) e attraverso le transizioni timed $\boldsymbol{T_r}$ il passaggio da uno stato di guasto ad uno stato di corretto funzionamento (repair).
- Attraverso le transizioni istantanee $\boldsymbol{t}$ e gli archi inibitori ad esse collegati, il guasto immediato dovuto alla rottura degli strati sottostanti.
- Attraverso gli archi inibitori l'impossibilità di un componente di ripristinare il suo stato di corretto funzionamento qualora lo strato sottostante sia ancora guasto.

Le condizioni iniziali definite per questo modello sono state di 3 token per il P_{upCNT}, per modellare la presenza di 3 container, e di un token per tutti gli altri P_{up}.

2.3 Calcolo della disponibilità per il singolo nodo

Per calcolare la disponibilità del singolo nodo bisogna garantire l'esistenza della reward rate function X(t), un processo casuale non negativo che rappresenta le condizioni di stato del sistema. Per valutare la disponibilità, la reward rate function $X(t)$ varia in accordo alle varie misure e per il caso specifico, per le misure di disponibilità, essa vale $X(t) = 1$ se siamo in uno stato di corretto funzionamento e $X(t) = 0$ se siamo in uno stato di non funzionamento.

A questo punto è possibile calcolare il reward rate atteso istantaneo, definito come:

$$E[X(t)] = \sum_i r_i \pi_i(t)$$

In cui:

- $\boldsymbol{r_i}$ rappresenta il reward rate associato alla particolare marcatura i.
- $\boldsymbol{\pi_i(t)}$ rappresenta la probabilità di essere nella marcatura i in un generico istante di tempo t.

Il reward rate atteso a regime viene definito come:

$$E[X] = \sum_i r_i \pi_i$$

Si dimostra che la disponibilità istantanea è uguale al reward rate atteso instantaneo e, analogamente, la disponibilità a regime è uguale al reward rate atteso a regime.

$$A(t) = E[X(t)]$$

$$A = E(X)$$

2.4 Definizione della reward function

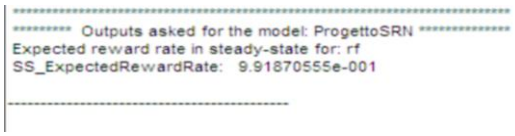
Come da specifiche, affinché ci sia il corretto funzionamento del sistema è necessario che almeno un container sia funzionante. Pertanto, la reward function risulta essere la seguente:

```
if(#(PupCNT)>=1)
1
else
0
end
```

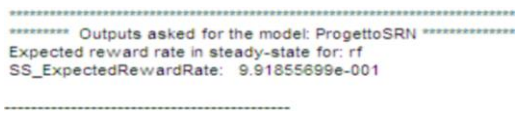
Dove per $\#P_{upCNT}$ si fa riferimento al numero di token presenti nel posto P_{upCNT}.

2.5 Risultato ottenuto

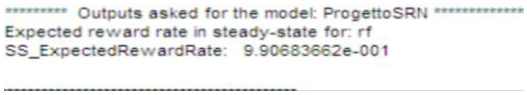
A questo punto, una volta definita la reward function, attraverso il tool SHARPE è possibile calcolare la steady state availability del singolo nodo. Di seguito viene riportato il risultato ottenuto per i vari nodi:



Per il nodo 1 e 2;



Per il nodo 3;



Per il nodo 4;

3. METODI STOCASTICI UTILIZZATI PER LA VALUTAZIONE DELLA DISPONIBILITÀ STAZIONARIA

La valutazione della disponibilità stazionaria di un sistema può essere effettuata utilizzando diversi metodi stocastici. Qui di seguito vengono illustrati alcuni dei metodi stocastici tipicamente utilizzati, evidenziando quelli che sono i pro e i contri delle varie tecniche.

3.1 Catene di Markov tempo continue (CTMC)

Le catene di Markov a tempo continuo (CTMC) sono un tipo di processo stocastico che descrive l'evoluzione di un sistema in cui le transizioni da uno stato all'altro sono governate da una distribuzione di probabilità esponenziale. Le CTMC sono utilizzate per modellare sistemi che evolvono nel tempo, in cui la probabilità di una transizione dipende solo dallo stato corrente e dal tempo trascorso in tale stato. Esse vengono spesso utilizzate per modellare sistemi complessi, come reti di computer, sistemi di telecomunicazioni, sistemi di produzione elettronica e impianti industriali e anche per modellare la durata di eventi come, ad esempio, il tempo tra le chiamate in un call center.

Per calcolare la disponibilità stazionaria di un sistema usando le CTMC, è necessario costruire un modello di catena di Markov che rappresenti il sistema, definendo gli stati del sistema e le transizioni tra di essi. Successivamente, è necessario assegnare una probabilità di transizione (o rate) a ciascuna transizione nello stato del sistema e risolvere le equazioni di bilancio di probabilità per il modello della catena di Markov.

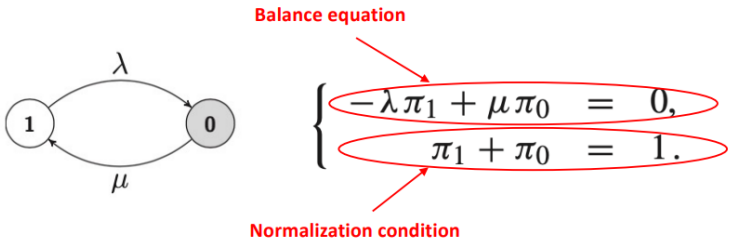


Figura 5: Catena di Markov tempo continuo

Un altro modo più complesso per il calcolo della disponibilità è la risoluzione delle equazioni differenziali. In questo modo, è possibile calcolare la probabilità che il sistema si trovi in uno stato particolare in un determinato momento, ovvero la disponibilità stazionaria del sistema.

Le catene di Markov rappresentano quindi uno strumento semplice ed immediato da utilizzare per andare a modellare problemi a bassa dimensionalità. Il contro di questo formalismo risulta essere il problema dell'esplosione dello spazio degli stati. Ovvero, se volessimo andare a considerare ulteriori stati intermedi nella rappresentazione del modello, andremmo inevitabilmente a complicare quelle che sono le equazioni di bilancio per poter calcolare la disponibilità stazionaria, rendendo il modello più pesante.

3.2 Stochastic Petri Nets e Stochastic Reward Nets (SRN)

Le reti di Petri stocastiche nascono come semplificazione delle CTMC e aiutano a risolvere il problema della rappresentatività. Formalmente una rete di Petri stocastica è definita come una sestupla formata dall'insieme dei posti, l'insieme delle transizioni, l'insieme delle relazioni input output rispetto agli stati, il set di token o marking iniziali e il set dei rate delle transizioni esponenziali.

Le reti stocastiche con reward risultano essere un ulteriore passo in avanti rispetto alle reti di Petri stocastiche per via dell'aggiunta del concetto di funzione di guardia. Una transizione viene definita disabilitata finché la funzione di guardia è impostata al valore TRUE. Inoltre, la cardinalità di un arco, i rate e le probabilità delle transizioni in una SRN possono essere espressi in funzione della distribuzione dei marking. Le SRN, d'altra parte, sono utilizzate per modellare sistemi che evolvono in base a ricompense, in cui le transizioni tra gli stati dipendono dalla ricompensa accumulata. Le SRN consentono di modellare il comportamento di sistemi in cui l'obiettivo è massimizzare una ricompensa specifica.

Le operazioni da seguire per la valutazione della disponibilità in una SRN sono state già riportate nel capitolo 2.

3.3 Riassunto dei pro e dei contro

Quindi dal punto di vista numerico con le catene di Markov si riesce ad avere il controllo completo di quello che sta accadendo mentre col formalismo delle SRN bisogna calcolare da un lato in quale distribuzione ci si trova e da un altro lato la probabilità di stare in un determinato marking. In sintesi, il formalismo delle SRN risulta essere molto efficace dal punto di vista rappresentativo mentre per il controllo risulta essere sicuramente peggiore rispetto al formalismo delle catene di Markov.

Entrambe le tecniche sono utilizzate per descrivere il comportamento di sistemi dinamici, pertanto le modellazioni sono intercambiabili. È possibile rappresentare una CTMC come una SRN assegnando ricompense appropriate alle transizioni tra gli stati. Tuttavia, è importante notare che le SRN possono rappresentare anche altri aspetti del sistema, come i costi o le penalità associate alle transizioni. Mentre è possibile modellare una CTMC utilizzando una SRN, il viceversa può essere più complicato a causa di particolari strutture come gli archi inibitori o le transizioni immediate. A conferma di questo, nelle SRN è consentito l'utilizzo delle transizioni istantanee, mentre nelle CTMC tale formalismo non risulta ammissibile.

3.4 Scelta del modello di disponibilità

Al fine di modellare il nodo rappresentato nel capitolo 2 in Figura 3, si è scelto di utilizzare il formalismo delle Stochastic Reward Networks per via della sua efficacia dal punto di vista rappresentativo. La rappresentazione del nodo sarebbe risultata essere molto complessa nel caso in cui si fosse scelto il formalismo delle catene di Markov a tempo continuo dato che il numero considerevole di stati da analizzare avrebbe portato inevitabilmente ad una perdita di efficacia nella rappresentazione visiva.

4. STIMA DEI PARAMETRI λ_{PHY} e μ_{PHY}

Per poter calcolare la disponibilità stazionaria dei quattro nodi in serie, sono stati forniti i valori relativi a tutti gli MTTF e gli MTTR dei componenti.

RECIPROCO DEI VALORI DEI PARAMETRI

1/Parametro	Valore
$1/\lambda_{CNT}[h]$	800
$1/\lambda_{DCK}[h]$	1500
$1/\lambda_{VM}[h]$	2000
$1/\lambda_{HYP}[h]$	3000
$1/\lambda_{PHY}[h]$	da stimare
$1/\mu_{CNT_i-N_1}[s]$	10
$1/\mu_{CNT_i-N_2}[s]$	10
$1/\mu_{CNT_i-N_3}[s]$	100
$1/\mu_{CNT_i-N_4}[h]$	2
$1/\mu_{DCK}[s]$	10
$1/\mu_{VM}[h]$	3
$1/\mu_{HYP}[h]$	3
$1/\mu_{PHY}[h]$	da stimare

Figura 6: Tabella dei reciproci dei valori dei parametri

Dalla tabella è possibile notare che i valori relativi allo strato fisico siano da stimare mediante tecniche statistiche. Per effettuare tale stima, vengono forniti i dataset contenenti rispettivamente un campione completo di tempi di riparazione e un campione censurato, secondo time censoring, di tempi di guasto "repairs_gr2.mat" e "failures_gr2.mat", quest'ultimo accompagnato da un dataset "censoring_gr2.mat" contenente gli indicatori che ci consentono di comprendere quali dati del dataset relativo ai failures siano censurati.

4.1 Analisi dei dataset

Sono stati analizzati i dataset. Per il dataset relativo ai tempi di riparazione sono presenti 183 campioni (campione completo) che presentano una distribuzione qualitativa dei tempi riportata in figura:

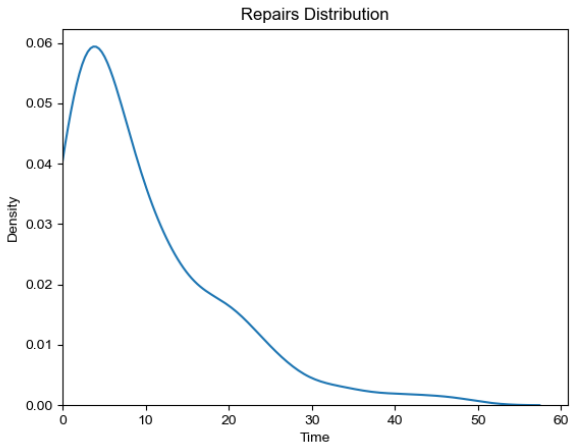
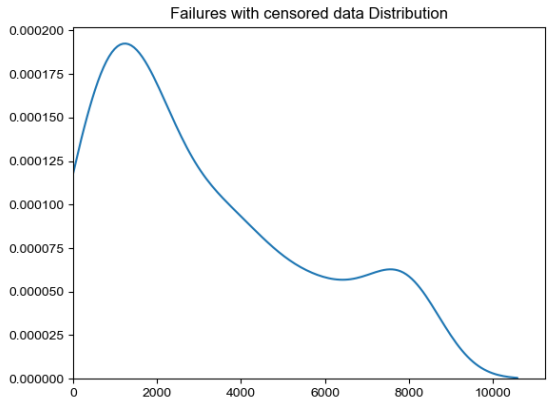
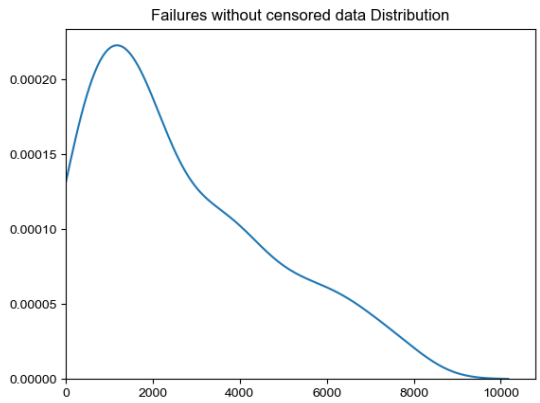


Figura 7: Distribuzione dei tempi di riparazione

Per quanto riguarda i failures, i dati in possesso risultano essere 200 di cui 17 censurati con censura time censoring (tipo 1) all'istante di 8000 ore. L'andamento qualitativo con e senza censura dei tempi di fallimento è il seguente:





Dall'andamento presentato sia dai tempi di fallimento e riparazione, è possibile intuire che il modello più adatto per la stima dei parametri è quello esponenziale. Tale assunzione ci permette di eseguire un'analisi a massima verosimiglianza. Inoltre, è fondamentale che questi tempi afferiscano ad un modello esponenziale per garantire che il modello SRN presentato sia corretto.

4.2 Algoritmo per la stima dei parametri

Per trovare la stima a massima verosimiglianza del parametro λ_{PHY} è necessario seguire questo algoritmo:

Sia $\underline{x} = (x_1, x_2, \ldots, x_m; \tau_1, \tau_2, \tau_{n-m})$ il vettore di osservazioni di m campioni falliti al tempo x_i ed n-m tempi di sopravvivenza τ_j. La stima a massima verosimiglianza associata è:

$$L(\theta;\underline{x}) = \left\{\prod_{i=1}^{m} \theta^{-1} \exp(-x_i/\theta)\right\}\left\{\prod_{j=1}^{n-m} \exp(-\tau_j/\theta)\right\}$$

$$= \theta^{-m} \exp\left\{-\theta^{-1}\left[\sum_{i=1}^{m} x_i + \sum_{j=1}^{n-m} \tau_j\right]\right\}$$

Assegnato il tempo totale di test:

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

Si riscrive la L come:

$$L(\theta;\underline{x}) = \theta^{-m} \exp\left\{-T^{*}\theta^{-1}\right\},$$

Si può calcolare la log-verosimiglianza $L(\theta;\underline{x}) = \log L(\theta;\underline{x})$, dato che massimizzare una funzione è equivalente a massimizzarne il suo logaritmo. Infine, si deriva la funzione ottenuta e la si eguaglia a zero. La stima a massima verosimiglianza di θ viene indicata con $\hat{\theta}$:

$$\hat{\theta} = \frac{T^{*}}{m} = \frac{\sum_{i=1}^{m} x_i + \sum_{j=1}^{n-m} \tau_j}{m},$$

Tale valore ricavato rappresenta la stima del MTTF associato a PHY. È possibile ricavare, grazie alla proprietà di invarianza, la stima a massima verosimiglianza λ_{phy} come il reciproco dell'MTTF.

Per stimare invece il valore di μ_{phy} è possibile semplificare le formule precedenti data l'assenza di valori censurati fino ad ottenere:

$$\hat{\theta} = \frac{T^{*}}{n} = \frac{\sum_{i=1}^{n} x_i}{n}.$$

4.3 Risultati ottenuti

I risultati dei parametri ottenuti a seguito dell'algoritmo di stima sono:

1/ $\lambda_{PHY} =$ ***3363.548283875574***

1/ $\mu_{PHY} =$ ***9.7081920533544***

5. RAGGIUNGIMENTO DEI SIX NINES

5.1 Modello RBD e calcolo della disponibilità stazionaria

Una volta modellati i singoli componenti attraverso la rappresentazione SRN presentata nel capitolo 2, si è scelto di utilizzare un modello RBD al fine di rappresentare il sistema in esame. L'utilizzo di un RBD permette di avere una rappresentazione più compatta ed efficace sfruttando l'indipendenza stocastica dei singoli nodi.

Il sistema in esame è stato rappresentato attraverso quattro nodi in serie (N1, N2, N3, N4). La rappresentazione dell'RBD è mostrata in Figura 8:

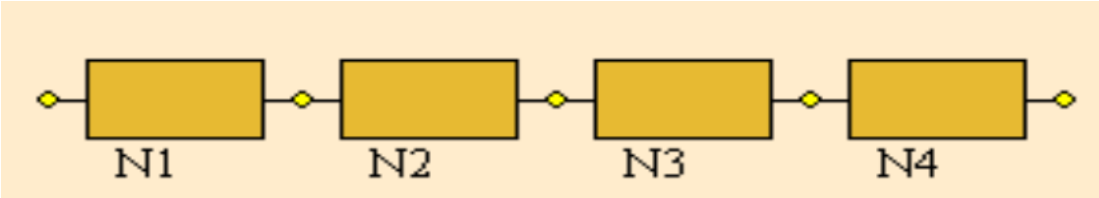


Figura 8: RBD del sistema

Il calcolo della disponibilità stazionaria di un sistema composto da più nodi in serie è espresso attraverso la seguente formula:

$$A_{SS} = \prod_{i=1}^{n} A_i$$

dove A_i rappresenta l'availability per lo specifico nodo i.

Invece, per un sistema composto da nodi in parallelo si ha:

$$A_{SS} = 1 - \prod_{i=1}^{n} [1 - A_i]$$

Di conseguenza, la disponibilità stazionaria del sistema in serie è un valore pari a ***0.966703941979336.***

5.2 Configurazione minima della SFC per il raggiungimento dei six nines

Ai fini di ottenere la disponibilità stazionaria pari ad almeno six nines è stato utilizzato un approccio euristico basato su tentativi che ha permesso di ottenere il risultato desiderato. I tentativi si sono concentrati sull'andare a ridondare ogni singolo componente del sistema, monitorando il valore della disponibilità stazionaria ad ogni incremento. Si è deciso di andare ad attuare una ridondanza dei componenti e non del sistema poiché, nel caso in esame le prove effettuate hanno mostrato un affidabilità migliore rispetto alla ridondanza a livello di sistema. Vengono riportati i risultati ottenuti per alcuni dei tentativi effettuati, il risultato relativo alla configurazione finale e la rappresentazione della configurazione finale.

A_{ss} (2-2-2-2) = 0.999714730817930

A_{ss} (3-3-3-3) = 0.999997576674452

A_{ss} (4-4-4-3) = ***0.999999178261502.***

I numeri riportati tra parentesi fanno riferimento al numero di componenti ridondati per ogni nodo del sistema. In questo caso la configurazione finale in figura, presenta 4 componenti per il primo, il secondo e il terzo nodo e 3 componenti per il quarto nodo ed è stata individuata come la configurazione minima per raggiungere i six nines.

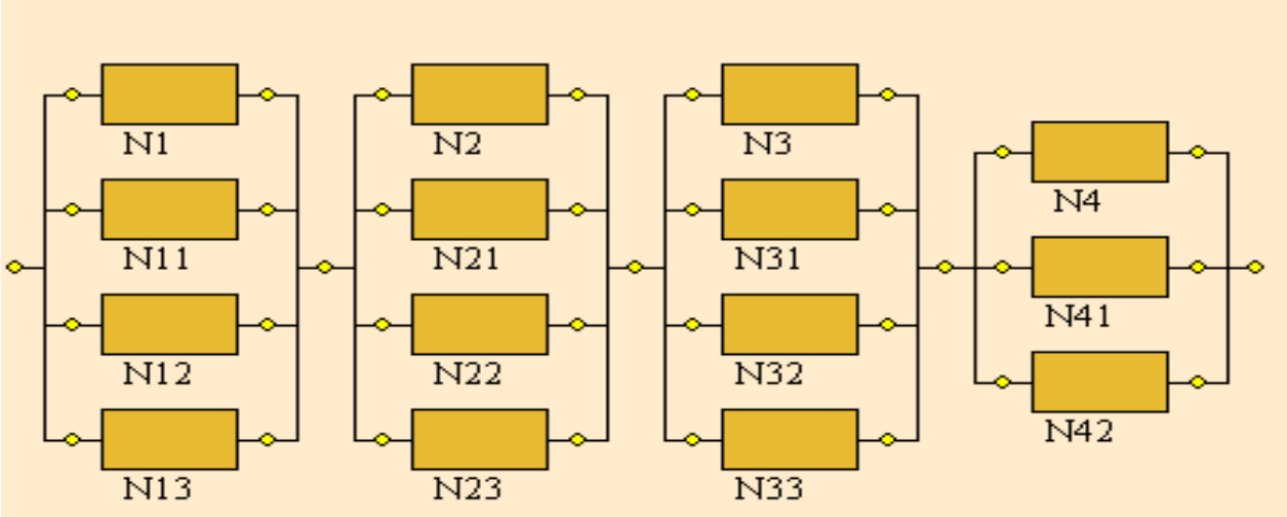


Figura 9: RBD con ridondanza minima

5.3 Ricerca della configurazione minima tramite algoritmo genetico

Per avere una conferma dei risultati ottenuti, è stato ideato un semplice algoritmo genetico capace di trovare una delle configurazioni minime per il problema in esame. Gli algoritmi genetici sono un tipo di algoritmo di ottimizzazione ispirato alla teoria dell'evoluzione. Si basano sulla selezione naturale, la riproduzione e la mutazione per generare soluzioni ottimali per problemi complessi.

L'algoritmo genetico inizia da un insieme di soluzioni iniziali, chiamate "popolazione". Le soluzioni sono valutate attraverso una funzione di fitness, che misura quanto bene ogni soluzione risolve il problema. L'algoritmo genetico procede attraverso una serie di iterazioni chiamate "mutazioni". In ogni mutazione, le soluzioni migliori vengono selezionate per la riproduzione. Le soluzioni selezionate sono quindi "riprodotte" tra loro, creando nuove soluzioni figlie. Le nuove soluzioni figlie vengono quindi valutate attraverso la funzione di fitness e le migliori vengono selezionate per la generazione successiva.

Questo processo di selezione, riproduzione e mutazione viene ripetuto per un certo numero di generazioni, fino a quando non viene trovata una soluzione ottimale o si raggiunge un limite di tempo o di iterazioni. Di seguito viene riportato il codice Python che permette di risolvere il problema in esame:

```python
import random

"""PRAMETERS"""
availiability_treshold=0.999999
cost_penalty_regulizer = 1
"""
seed=0
random.seed(seed)
"""
population=20
iterations=50000


"""STARTING POINT"""
#notazione: la serie è indicata come una lista in cui i numeri all interno corrispondono al numero di blocchi per quella posizione
model = [1,1,1,1]
avliabilities=[0.99187055,0.99187055,0.991855699,0.990683662]


"""MUTATION ENGINE"""
def mutation(model):
    selector = random.randint(0, 2*(len(model)))
    change_mode = int(selector/len(model)-1)
    block_selected = selector%len(model)
    if change_mode == 0: #RIDONDA
        model[block_selected] = model[block_selected]+1
    elif change_mode == 1: #RIMUOVI
        if model[block_selected]>1:
            model[block_selected] = model[block_selected]-1
    return model


```

```
"""AVALIABILITY EVALUATION"""
def evaluate_availiability(model, availiabilities):
    partial_av=[]
    block_selector=0
    for block in model:
        partial_av.append(1-pow((1-availiabilities[block_selector]),model[block_selector]))
        block_selector=block_selector+1
    av=1
    for part in partial_av:
        av *= part
    return av

"""FITNESS EVALUATION"""
def fitness_function(model, total_availiabilty,cost_penalty_regulizer):
    tot_block= block_count(model)
    fit_value = total_availiabilty-tot_block/cost_penalty_regulizer
    return fit_value

"""BLOCK CALCULATION"""
def block_count(model):
    tot_block=0
    for block in range(0,len(model)-1):
        tot_block = tot_block + model[block]
    return tot_block


#MAIN
winner = [99,99,99,99]
for i in range(1,iterations):
    conditon = True
    best_model = [1,1,1,1]
    best_fitness = fitness_function(model,evaluate_availiability(model, avliabilities),cost_penalty_regulizer)
    while (conditon):
        for sample in range(1,population):
            sample_model = mutation(best_model)
            sample_availability = evaluate_availiability(sample_model,avliabilities)
            sample_fitness = fitness_function(sample_model,sample_availability,cost_penalty_regulizer)
            if sample_fitness > best_fitness:
                best_model= sample_model
                best_fitness= sample_fitness
            if(sample_availability >= availiability_treshold):
                conditon= False
                if block_count(winner)>block_count(sample_model):
                    print("Nuovo miglior modello", sample_model,"con availability",sample_availability)
                    winner=sample_model
                break
```

I risultati ottenuti sono in accordo a quelli ottenuti mediante l'algoritmo euristico utilizzato precedentemente.

6. ANALISI DI SENSITIVITÀ

Attraverso l'analisi di sensitività riportata in questo capitolo è possibile andare ad analizzare l'andamento dei parametri λ_{DCK} e μ_{DCK} per poter individuare quelli che sono i valori critici al di sotto dei quali non è possibile raggiungere il valore di six nines per la disponibilità stazionaria. Tale analisi viene effettuata sul sistema ridondato ed è mostrata qui di seguito.

6.1 Analisi di sensitività al variare dei valori del parametro λ_{DCK}

Mediante uno script Python è stato possibile creare un grafico che mostrasse l'andamento della disponibilità stazionaria al variare del parametro $1/\lambda_{DCK}$. I punti segnati sul grafico sono stati calcolati precedentemente col tool MATLAB e presentano tali valori:

Disponibilità Stazionaria	$1/\lambda_{DCK}$
0.999999178296708	1500
0.999999155050243	1250
0.999999057656182	750
0.999998925702479	500

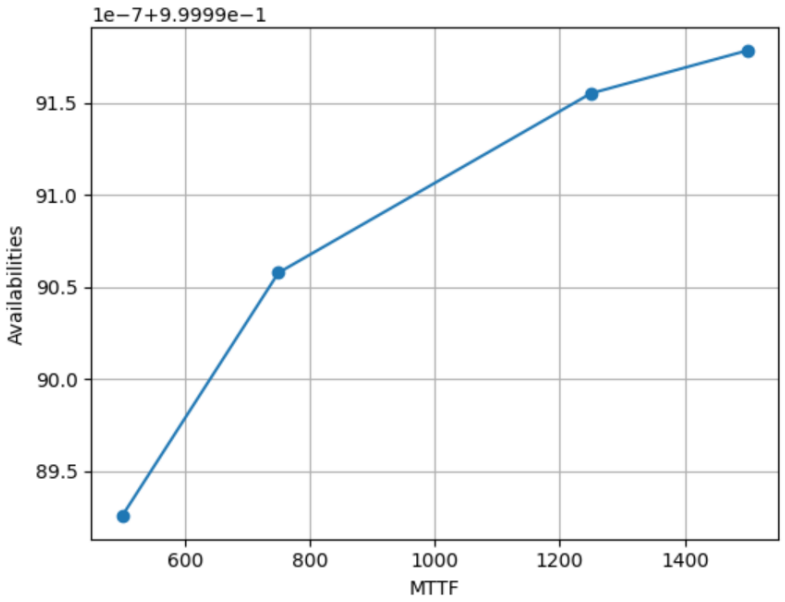


Figura 10: Analisi sensitività $1/\lambda_{DCK}$

Al fine di attuare un'analisi più accurata per trovare il valore critico di $1/\lambda_{DCK}$, si è fatto uno zoom sulla parte relativa all'intersezione col punto 0.999999. È stato possibile quindi individuare un valore critico di $1/\lambda_{DCK}$ pari a 641.3 h.

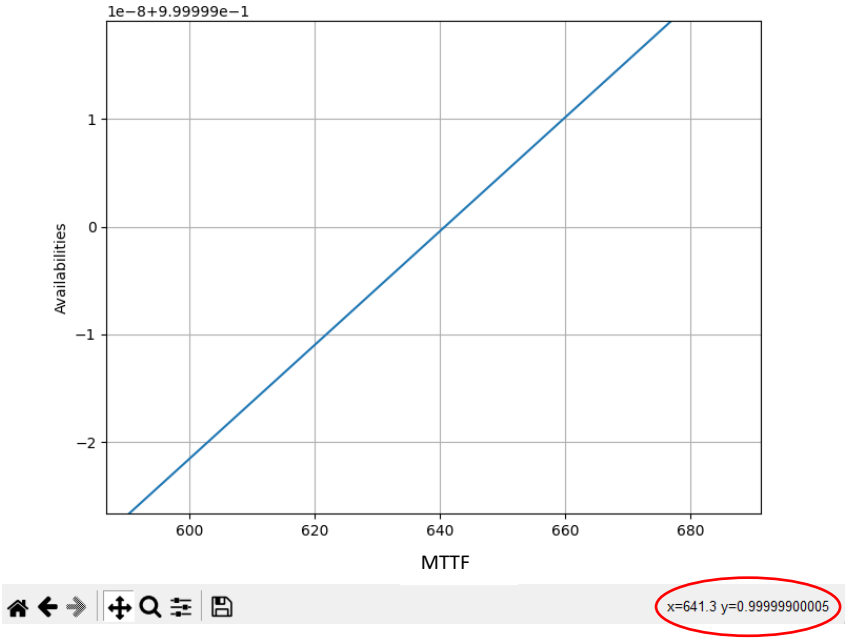


Figura 11: Valore critico $1/\lambda_{DCK}$

Considerando che il valore fornito di questo parametro è di 1500 h, da questa analisi possiamo notare che si ha un buon margine dell'MTTF relativo al nodo di Docker pari a circa 858.7 h.

6.2 Analisi di sensitività al variare dei valori del parametro μ_{DCK}

Allo stesso modo, utilizzando lo stesso script Python è stato creato il grafico che mostra l'andamento della disponibilità stazionaria al variare del parametro 1/ μ_{DCK}. I punti segnati sul grafico sono stati calcolati precedentemente col tool MATLAB e presentano tali valori:

Disponibilità Stazionaria	$1/\mu_{DCK}$
0.999999176976224	20
0.999999166325682	100
0.999999040553661	1000
0.999998965509832	1500

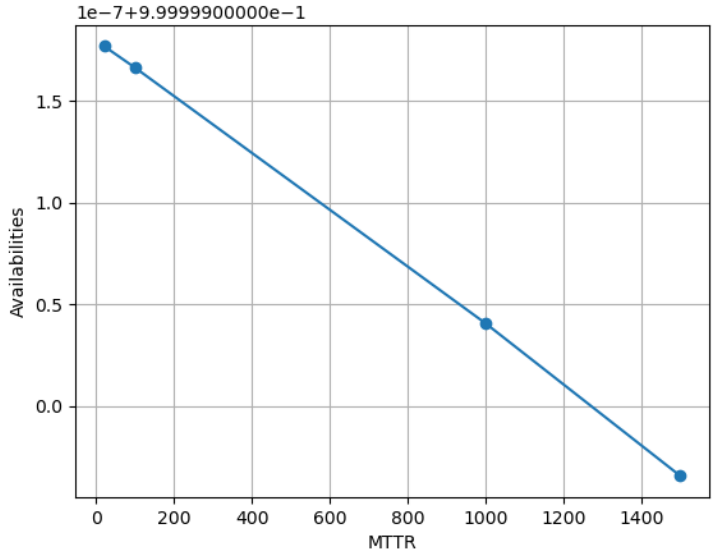


Figura 12: Analisi sensitività 1/ μ_{DCK}

Al fine di attuare un'analisi più accurata per trovare il valore critico di 1/ μ_{DCK}, si è fatto uno zoom sulla parte relativa all'intersezione col punto 0.999999. È stato possibile quindi individuare il valore critico di 1/ μ_{DCK} pari a 1270.5 s.

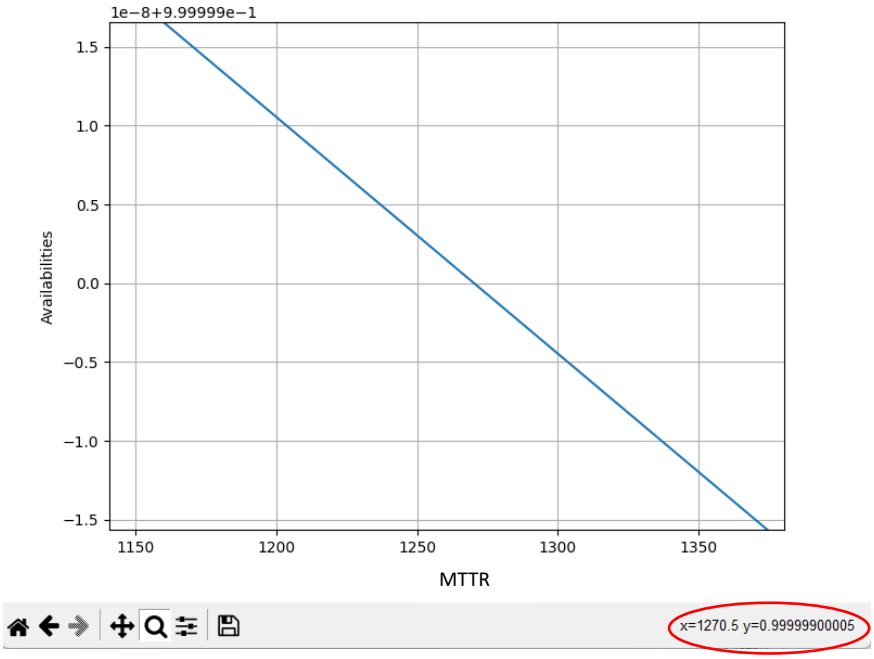


Figura 13: Valore critico 1/ μ_{DCK}

Anche in questo caso, considerando che il valore fornito di questo parametro è di 10 s, da questa analisi possiamo notare che si ha un enorme margine dell'MTTR per il nodo di Docker di circa 1260.5 s.