.DIEM

UNIVERSITÀ DEGLI STUDI DI SALERNO

Dipartimento di Ingegneria dell'Informazione ed Elettrica e Matematica applicata (DIEM)

System Safety Engineering

Relazione Progetto d'esame

prof. Fabio Postiglione
prof. Mario Di Mauro

GRUPPO N° 1	
Studenti	***Matricole***
Barone Guglielmo	06227 01885
Carratù Arianna	06227 01696
Grimaldi Salvatore	06227 01742

Anno Accademico 2022 – 2023

Indice

1 Introduzione **3**

2 Network Function Virtualization (NFV) e Service Function Chain (SFC) **4**
2.1 NFV: architettura 4
2.2 SFC: caratteristiche e implementazione 5
2.2.1 Caratteristiche 6
2.2.2 Implementazione 6
2.3 IP Multimedia Subsystem (IMS) 7
2.4 Le sfide del paradigma NFV 8

3 Confronto tra metodi stocastici per valutare la disponibilità stazionaria **9**

4 Modello di disponibilità di un nodo SFC a 5-layer **13**
4.1 Descrizione della struttura 5-layer 13
4.2 Modello SRN 14
4.2.1 Disponibilità stazionaria 15
4.2.2 Confronto con eventuale modellazione tramite CTMC 15

5 Stima a massima verosimiglianza dei tassi di guasto e di riparazione del layer hardware **16**
5.1 Procedura di stima a massima verosimiglianza 16
5.2 Modello Esponenziale 17
5.3 Stima del tasso di riparazione μ_{PHY} 18
5.4 Stima del tasso di guasto λ_{PHY} 19

6 Analisi di disponibilità **21**
6.1 Disponibilità della serie $(N_1 - N_2 - N_3 - N_4)$ 21
6.2 Individuazione della configurazione *six nines* 22
6.2.1 Algoritmo availability_calculator 23
6.2.2 Risultati 23
6.2.3 Validazione dei risultati in TimeNET 25

7 Analisi di sensitività **28**
7.1 Risultati 28
7.2 Focus sulla procedura adottata 29
7.2.1 Algoritmo sensitivity_calculator 30
7.2.2 Estrazione dei risultati e plot in MATLAB 31

8 Conclusioni **33**

1 Introduzione

I nuovi paradigmi di rete come la **Network Function Virtualization (NFV)** permettono di definire sistemi di comunicazione, in applicazioni *safety-critical*, attraverso strutture dinamiche come le **Service Function Chain (SFC)**, ovvero reti di nodi virtualizzati collegati in serie. In questo progetto si prende in considerazione uno scenario in cui, per motivi di emergenza, si richiede di mettere in piedi una SFC in maniera rapida ed efficiente per trasferire dati clinici da una server farm ad un centro operativo. La struttura di rete in questione è formata da 4 nodi connessi in serie ($N_1 - N_2 - N_3 - N_4$), tutti realizzati secondo il cosiddetto modello a container 5-layer.
La relazione è strutturata nel modo seguente: il capitolo 2 fornisce una breve introduzione del contesto tecnologico e delle principali caratteristiche delle Service Function Chain, con particolare riferimento al paradigma NFV e al dominio IP Multimedia Subsystem (IMS). Il capitolo 3 è incentrato sulla descrizione dei principali metodi stocastici per valutare la disponibilità stazionaria di un sistema e presenta anche un confronto tra essi, evidenziandone pro e contro. Il capitolo 4 è dedicato alla presentazione del modello di disponibilità adottato per la rappresentazione di un singolo nodo SFC a 5-layer. In particolare, vengono descritte l'architettura 5-layer (container, docker, virtual machine, hypervisor, hardware) ed il modello SRN scelto per analizzarne l'evoluzione in presenza di eventi di fallimento e di riparazione. Il capitolo 5 si focalizza sulla procedura di stima a massima verosimiglianza eseguita per stimare due parametri fondamentali (λ_{PHY} e μ_{PHY}, rispettivamente il rate di fallimento e di riparazione dell'hardware) per poter effettuare la successiva analisi di disponibilità. Quest'ultima occupa interamente il capitolo 6 ed ha l'obiettivo di individuare la configurazione a minima ridondanza tale da garantire una disponibilità stazionaria almeno uguale a 0.999999, anche detta *six nines steady-state availability*. Il capitolo 7 è, invece, incentrato sull'analisi di sensitività condotta per valutare la robustezza del sistema ridondato ottenuto alle deviazioni rispetto ai valori nominali dei parametro λ_{DCK} e μ_{DCK}, che sono rispettivamente il rate di fallimento e di riparazione del layer docker. Infine, il capitolo 8 tira le somme del lavoro svolto, fornendo le opportune conclusioni.

2 Network Function Virtualization (NFV) e Service Function Chain (SFC)

L'incremento esponenziale di utilizzo dei dispositivi mobili negli ultimi 20 anni ha rivoluzionato il campo delle reti di telecomunicazioni, richiedendo un sostanziale cambio di tendenza da parte dei providers, che hanno iniziato a porre sempre più attenzione alla **reliability** (affidabilità) e alla **availability** (disponibilità). Brevi interruzioni di un'infrastruttura di rete, infatti, possono avere conseguenze drastiche, che vanno dalla perdita economica alla perdita di vite umane. I guasti o disservizi sono eventi aleatori da gestire e, associando ad essi un rischio, la probabilità di errore deve essere la più bassa possibile. Tuttavia, la riduzione del rischio di guasti comporta un notevole incremento dei costi da sostenere[15][12]. La sfida principale che coinvolge i providers, sia di rete sia di telecomunicazione, è quella di soddisfare tali requisiti mitigando i costi [4].
Nell'ambito della transizione verso le reti mobili di quinta generazione (5G), una delle componenti più importanti è il paradigma **Network Function Virtualization** (NFV), poiché consente di superare la verticalizzazione delle Network Functions (NF) su singoli nodi fisici, avvalendosi della virtualizzazione a livello software [7]. Un altro paradigma degno di nota è il Software Defined Networking (SDN), che permette di dividere il piano dei dati (inoltro dei pacchetti) dal piano di controllo (instradamento) [8]. In particolare, un controller SDN consente di gestire in modo centralizzato il traffico di rete attraverso l'invio delle tabelle di inoltro ai router. Un esempio di interoperabilità tra questi paradigmi è la **Service Function Chain** (SFC), ovvero un insieme ordinato di elementi di rete (ad esempio router, firewall, ecc.) collegati tra loro con l'obiettivo di fornire uno specifico macro-servizio [4].

2.1 NFV: architettura

Il paradigma NFV è nato nel 2012 in una conferenza promossa dall'European Telecommunications Standards Institute (ETSI) [2].
Prima della sua introduzione da parte dei providers, vi erano sole reti *non virtualizzate* in cui le NF, spesso indicate anche come nodi di rete o elementi di rete, erano implementate come una combinazione di software e hardware specifici del fornitore. [7].
Con l'adozione dell'NFV, le funzioni di rete complesse monolitiche in esecuzione su hardware specializzato vengono scomposte in unità funzionali più piccole e orchestrate dinamicamente su un'infrastruttura virtualizzata. Tali funzioni di rete includono sia l'elaborazione del piano di controllo sia l'elaborazione del piano dati, come il normale routing, il controllo degli accessi e la gestione della Quality of Services (QoS), e vengono eseguite come software in una Virtual Machine (VM), chiamata anche **Virtual Network Function** (VNF) [17].
Di seguito si presenta l'architettura di una moderna rete *virtualizzata*.
L'obiettivo del paradigma NFV è separare il software che definisce la VNF dall'hardware e dal software che crea un'infrastruttura di virtualizzazione delle funzioni di rete di hosting generica, detta **Virtual Function Virtualization Infrastructure** (NFVI), che esegue le VNF [17].
L'architettura di riferimento è definita dall'ETSI [17] con l'obiettivo di descrivere gli elementi necessari per realizzare le NFV in maniera standardizzata in modo da garantire l'*interoperabilità* tra diversi providers. Ciò consente di implementare diverse VNF sull'infrastruttura virtualizzata per supportare i servizi di rete *End-To-End* (E2E). Inoltre, tale infrastruttura virtualizzata può essere applicabile a diversi casi d'uso dell'operatore con il minimo sforzo di integrazione e il massimo riutilizzo [7]. In Figura 1 viene descritta l'architettura composta dai suoi blocchi: **Virtualized Network Functions** (VNFs), **NFV Infrastucture** (NVFI) e **NFV Management and Orchestration** (MANO). Di seguito, seguendo un approccio top-down, vengono descritte le componenti essenziali dell'architettura NFV partendo dai suoi tre principali blocchi:

1. **VNFs**: racchiude al suo interno una o più VNF, che rappresentano delle implementazioni software di NF che possono essere eseguite sull'infrastruttura di rete virtualizzata (VNFI) [7].

2. **VNFI**: è l'infrastruttura fisica che garantisce l'esecuzione delle VNF [7]. Si compone di un insieme di nodi distribuiti in diverse posizioni per supportare i requisiti di località e latenza nei vari casi d'uso. A sua volta, l'NFVI può essere classificato in tre domini [17]:

   (a) elaborazione e archiviazione;

   (b) hypervisor;

   (c) rete.

3. **MANO**: copre l'orchestrazione e la gestione del ciclo di vita delle risorse fisiche e/o software che supportano la virtualizzazione dell'infrastruttura e la gestione del ciclo di vita delle VNF. In aggiunta, si concentra su tutte le attività di gestione specifiche della virtualizzazione necessarie nel framework NFV [7]. Al suo interno, vi sono tre blocchi fondamentali:

   (a) NFV orchestrator: il suo obiettivo è quello di separare i VNF dall'infrastruttura, occupandosi della loro gestione [17].

   (b) VNF manager: è responsabile della gestione del ciclo di vita delle VNF, inclusa la creazione VNF, l'allocazione delle risorse, la migrazione e la cessazione [17].

   (c) Virtualized Infrastructure Manager (VIM): è responsabile del controllo e della gestione delle risorse di elaborazione, archiviazione e rete, nonché della loro virtualizzazione [17].

Come ultimo elemento dell'architettura presentata vi è un Operational Support System (OSS) che ha il ruolo di supportare funzioni gestionali e operative [17] operando da supervisore generale [5].

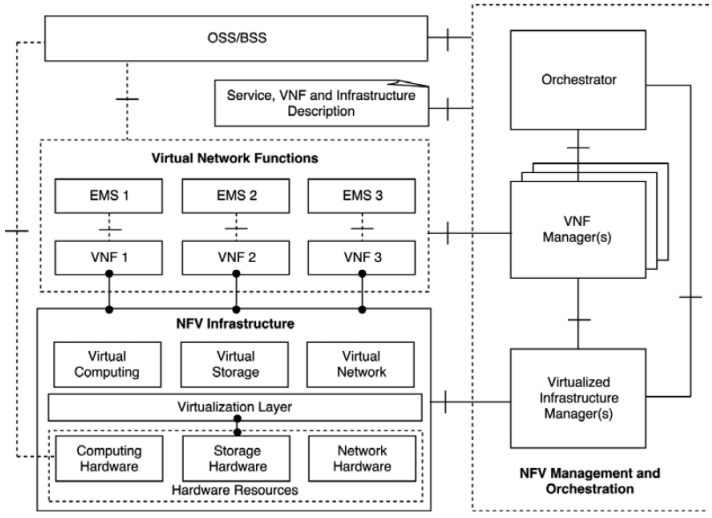

Figura 1: Architettura di riferimento per NFV [5]

2.2 SFC: caratteristiche e implementazione

Il framework NFV consente la costruzione e la gestione dinamica delle istanze VNF e delle relazioni tra di esse relative a dati, controllo, gestione, dipendenze e altri attributi [7]. Una composizione di VNF viene spesso definita Service Function Chain (SFC) ed è tipicamente governata dal Virtualized Infrastructure Manager (VIM), vale a dire l'elemento chiave dell'intera architettura NFV [6]. In Figura 2 vengono mostrati dei tipici esempi di SFC.

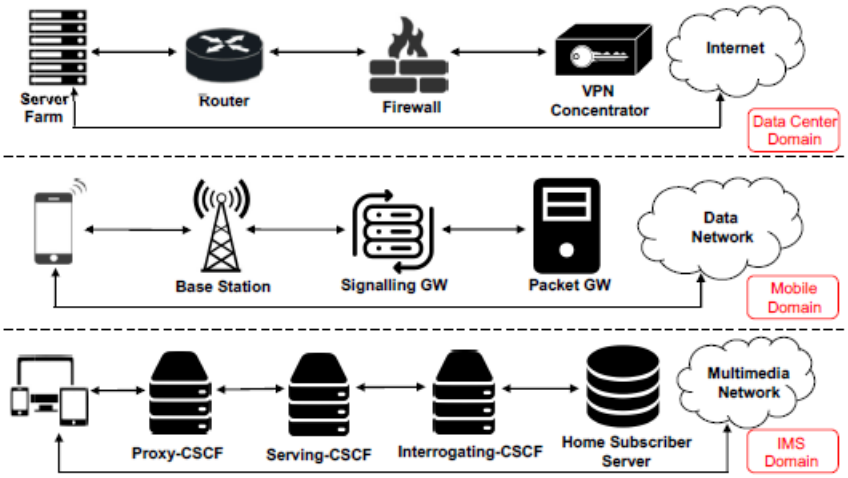

Figura 2: SFC utilizzate in 3 domini differenti: Data Center (in alto), Mobile (centro), IP Multimedia Subsystem (in basso) [11] [12]

2.2.1 Caratteristiche

Un **Network Service** (NS) può essere visto a livello architetturale come un *forwarding graph* (grafo di inoltro) di NF interconnesse che supportano l'infrastruttura di rete. Queste funzioni di rete possono essere implementate in una singola rete di operatori o interoperare tra diverse reti di operatori [7]. In Figura 3 viene rappresentato un grafo di inoltro tra due end points per un servizio di rete.

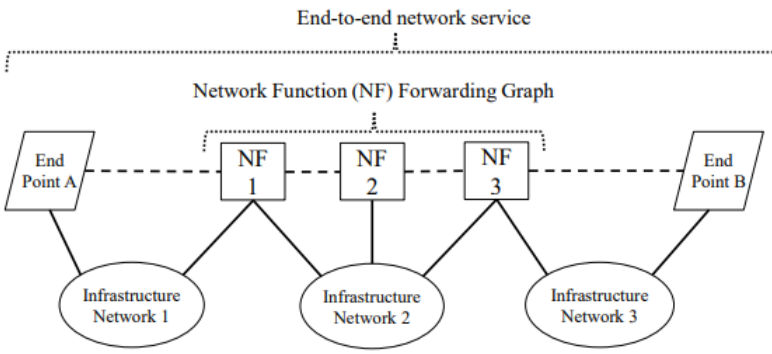

Figura 3: Rappresentazione del forwarding graph di un servizio di rete end-to-end [7]

Un grafo di inoltro di VNF si riferisce a una pipeline di VNF che il traffico dovrebbe attraversare [17]. Quindi, una SFC costituisce un insieme ordinato di funzioni responsabili di trattamenti specifici per singoli pacchetti che può operare a diversi livelli dello stack protocollare TCP/IP [4].
In altre parole, la pipeline può fornire un set di servizi **flessibile** e **configurabile** per il traffico di dati e con il grafo di inoltro VFN è possibile scegliere qualsiasi combinazione di servizi in base alle applicazioni e alle tipologie di reti (ad esempio, 4G, 5G, ecc) [17].

2.2.2 Implementazione

In uno scenario che prevede l'uso di reti virtualizzate, l'SFC può essere sfruttato in combinazione con alcuni elementi chiave dell'infrastruttura di rete come:

1. il controllore SDN, utile per creare e gestire percorsi attraversando la SFC quando alcuni criteri di rete sono abbinati;

2. il Virtualized Infrastructure Manager, incaricato di gestire e distribuire le risorse virtuali necessarie per un corretto funzionamento delle VNF che compongono le SFC [4].

Inoltre, la flessibilità ottenuta consente agli operatori di rete di fornire, in modo rapido e flessibile, molteplici soluzioni adeguate per esigenze specifiche [4]. In Figura 4 si evince quanto detto, osservando che le funzionalità sono raccolte in due domini: **dominio delle funzioni di servizio** (*Service Function Domain*) e il **dominio di controllo** (*Control Domain*).

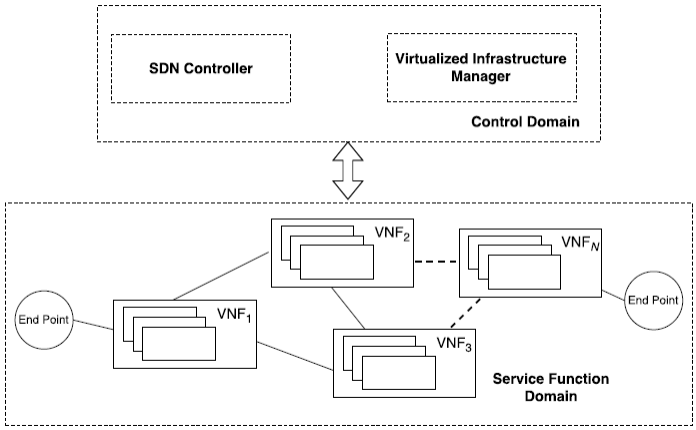

Figura 4: Implementazione della Service Function Chain usando un'architettura NFV [4]

Il primo dominio comprende l'insieme di VNF che costituiscono il servizio di rete specifico. Il secondo si riferisce ad una parte dell'infrastruttura di gestione incaricata di controllare e monitorare il lavoro delle VNF. Di seguito, si pone l'attenzione sulle Service Function Chains applicate al contesto delle reti IP Multimedia Subsystem. [4].

2.3 IP Multimedia Subsystem (IMS)

Con riferimento alla Figura 2, un esempio di SFC è l'**IP Multimedia Subsystem**. Il modello di riferimento IMS si basa sui server *Call Session Control Function* (CSCF), ovvero nodi di rete comunicanti tra loro. In Figura 5, vengono rappresentati i principali messaggi coinvolti in una comunicazione e i nodi della rete IMS. Il *Proxy CSCF* (P-CSCF) assume il ruolo di interfaccia tra un'apparecchiatura utente (*User Equipment*, UE) e il nucleo della rete IMS; il *Serving CSCF* (S-CSCF) si occupa principalmente dell'instradamento dei messaggi di segnalazione; l'*Interrogating CSCF* (I-CSCF) è un elemento facoltativo incaricato di inoltrare richieste o risposte al S-CSCF appropriato; l'*Home Subscriber Server* (HSS) fornisce alcune funzionalità di database come la profilazione degli utenti, l'autorizzazione, l'autenticazione e l'accounting. Una sessione IMS è fondamentalmente divisa in due fasi, come evidenziato in Figura 5: la prima riguarda una procedura di impostazione della chiamata (linee tratteggiate) incaricata di gestire le politiche di routing e la negoziazione delle risorse, e la seconda è relativa alla sessione multimediale (linea continua) tra le due UE [5].

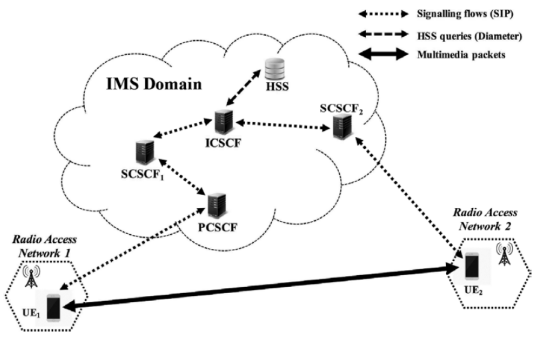

Figura 5: Una procedura di impostazione della chiamata (linee tratteggiate) e una sessione multimediale (linea continua) in un quadro IMS con due apparecchiature utente (UE) collegate a diverse reti di accesso radio [5]

2.4 Le sfide del paradigma NFV

Il paradigma NFV è stato sviluppato di recente e vi sono diverse sfide che devono essere affrontate da parte dei providers. In particolare, devono essere garantite le seguenti *carrier grades properties* [17]:

- **affidabilità e disponibilità**: per quanto concerne la disponibilità, essa deve essere intesa come la disponibilità del servizio End-To-End che include tutti gli elementi (VNF e componenti dell'infrastruttura).
- **efficienza**: il framework NFV dovrebbe adottare rigorosi Service-Level Agreement (SLA) per le NF offerte. Ad esempio, il SLA può specificare il ritardo medio, la larghezza di banda e la disponibilità per tutti i servizi forniti a un cliente. Per supportare la conformità ai SLA, la piattaforma dovrebbe monitorare attentamente le prestazioni per ciascun cliente e adattare dinamicamente le risorse per soddisfare i SLA.
- **scalabilità**: dovrebbe supportare un gran numero di VNF e adattarsi all'aumentare del numero di abbonati/applicazioni/volume di traffico. La capacità di offrire una selezione di NF per cliente potrebbe potenzialmente portare alla creazione di nuove offerte e, quindi, a nuovi modi per gli operatori di monetizzare le loro reti.

3 Confronto tra metodi stocastici per valutare la disponibilità stazionaria

Uno dei principali requisiti richiesti ad un sistema o servizio è la **disponibilità** (availability) dello stesso. Essa viene definita come la capacità di essere in uno stato tale da eseguire la funzionalità richiesta, sotto date condizioni, in un preciso istante di tempo, o dopo che è trascorso un tempo sufficiente (indipendentemente dal numero di guasti e riparazioni, facendo riferimento a *processi riparabili*). Dato che i guasti sono eventi aleatori, la disponibilità è, di fatto, una probabilità di stato, da cui l'adozione di modelli stocastici per la sua valutazione.
Non essendo possibile garantire la disponibilità per il 100% del tempo, si punta ad avere "un numero di 9 quanto più alto possibile", considerando un trade-off tra costi necessari e prestazioni garantite.
Si distinguono, inoltre, disponibilità istantanea e disponibilità stazionaria. La **disponibilità istantanea** ($A(t)$) è definita come la probabilità che il sistema sia in uno stato funzionante in un certo istante di tempo t. La **disponibilità stazionaria**, o availability steady-state (A_{SS}), è una misura fondamentale nell'analisi delle prestazioni di sistemi di rete, infrastrutture di telecomunicazione, servizi, e così via; essa viene definita come:

$$A_{SS} = \frac{MTTF}{MTTF + MTTR} \tag{1}$$

dove $MTTF$ è il *Mean Time To Failure* ed $MTTR$ è il *Mean Time To Repair*. Si osservi, inoltre, che la somma di $MTTF$ e $MTTR$ corrisponde al tempo di vita totale.
Un benchmark tipico, in particolare per i sistemi safety critical, codificato anche dagli standard, è rappresentato da una A_{SS} a cinque 9, che si traduce in un downtime di soli 5.256 minuti in un anno.

Si vogliono confrontare, dunque, i metodi stocastici tipicamente adoperati per la valutazione della disponibilità stazionaria, evidenziandone i vantaggi e gli svantaggi.
Secondo Trivedi in [15] i formalismi di modellazione possono essere classificati in:

- **non a spazio di stato**: queste tecniche sono concise ed efficienti e consentono all'utente di catturare le relazioni tra i componenti e le condizioni che portano al guasto di un sistema [16]. Tra di esse vi sono il **Reliability Block Diagram (RBD)**, il **Fault-Tree (FT)** e i Reliability Graphs. Una limitazione di questi modelli è l'assunzione che i componenti che costituiscono il sistema debbano essere statisticamente indipendenti, in quanto non esiste il concetto di stato [12].

- **a spazio di stato**: in tali tecniche l'evoluzione dell'affidabilità e della disponibilità è funzione esplicita dello stato [16].I modelli a spazio di stato stocastici si distinguono in Markoviani e Non-Markoviani. In particolare, i Markoviani si dividono a loro volta in tempo-discreto e tempo-continuo, e questi ultimi in omogenei e non. Tra quelli omegenei vi sono le Continous Time Markov Chain (CTMC) e le Stochastic Reward Net (SRN), caso particolare delle reti di Petri stocastiche.

- **multi-livello**: combinano la potenza di modellazione dei modelli a spazio di stato con l'efficienza dei modelli che non usano lo stato [12]. Ad esempio, in [5] viene adoperato un formalismo gerarchico a tre livelli dove al I livello vi è un *Reliability Block Diagram* (RBD), al II livello un *Fault-Tree* (FT) e, infine, una *Continue Time Markov Chain* (CTMC) che modella alcune foglie del FT.
  Un approccio simile è stato utilizzato per la modellazione della SFC oggetto della relazione. In particolare, si è adoperato al top-level un RBD e al low level delle SRNs.

Di seguito si confrontano i metodi stocastici di cui ci si avvale tipicamente per la valutazione della A_{SS}:

(i) **Fault Tree** (FT): un modello di Albero dei Guasti (o Fault Tree) è una struttura logica che aiuta nell'analisi di un sistema dal punto di vista di come l'affidabilità dei singoli componenti influisce sull'affidabilità di un sistema complessivo. Gli alberi dei guasti sono rappresentati come una struttura ad albero con la radice dell'albero che rappresenta un evento indesiderato (o *Top Event*), come un errore di sistema [16], da cui l'appellativo di "formalismo pessimistico". In altre parole, è un formalismo grafico che consente di illustrare le cause dei guasti di un sistema avvalendosi della logica booleana (in

particolare delle porte logiche AND e OR). Quindi, l'obiettivo è la definizione del Top Event in funzione di tutti i nodi dell'albero. Ogni evento di fallimento di base ha una certa probabilità di verificarsi, che può essere calcolata utilizzando dati storici o stime date dai fornitori dei componenti. Si utilizzano queste probabilità per calcolare la probabilità totale di fallimento del sistema e, in seguito, si calcola la A_{SS} come la sua probabilità complementare.
Il suo principale vantaggio consiste nell'evidenziare immediatamente i componenti critici correlati ai guasti del sistema. Invece, lo svantaggio nell'utilizzo di tale formalismo si ha nella difficoltà di catturare i cosiddetti *guasti di modo comune*. Inoltre, non è possibile rappresentare l'azione di riparazione di un componente.
Tale tecnica prende anche il nome di Fault Tree Analisys (FTA). Un esempio di FT è mostrato in Figura 6.

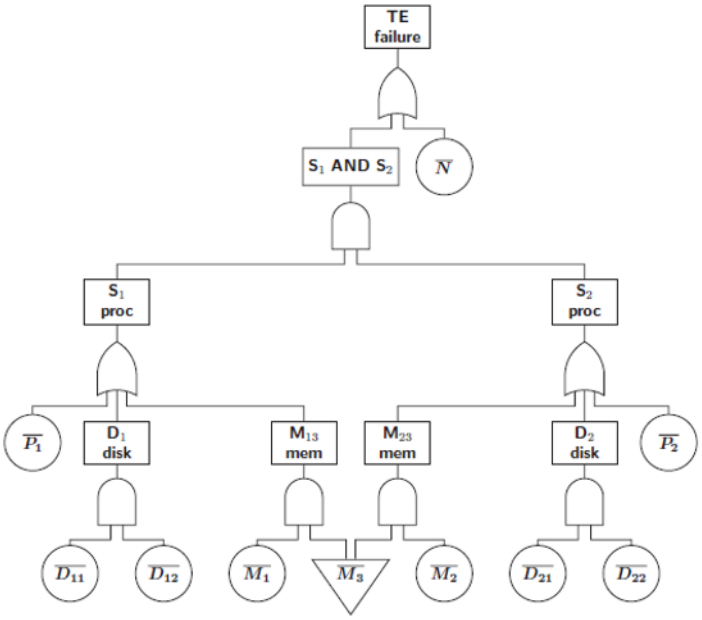

Figura 6: Esempio di FT che modella i guasti di un sistema [15]

(ii) **Continue Time Markov Chain** (CTMC): strumento grafico che offre una rappresentazione del sistema basata su stati, sfruttando il formalismo delle catene di Markov [12]. La modellazione del sistema risulta molto semplice da rappresentare mediante il diagramma a spazio di stato, e avviene attraverso la definizione dei rate (o tassi) di transizione da uno stato ad uno altro in accordo ad una distribuzione esponenziale. Ad esempio, in un sistema a due stati (funzionante e non funzionante) è possibile dare un significato fisico a questi rate: il rate di transizione dallo stato funzionante a quello non funzionante prende il nome di *failure rate* (λ), mentre quello dallo stato non funzionante a funzionante si denota con *repair rate* (μ), consentendo in tal modo di definire anche l'azione di riparazione, a differenza del formalismo precedente. Le CTMC consentono di calcolare la disponibilità stazionaria risolvendo il sistema costituito dalle equazioni di bilancio (per cui per ogni stato il flusso di probabilità entrante deve eguagliare quello uscente) e dalla condizione di normalizzazione delle probabilità, essendo la probabilità di essere nello stato funzionante pari proprio alla A_{SS}.
Tuttavia, si tratta di un approccio monolitico che può degenerare in un'esplosione dello spazio di stato per sistemi reali complessi con un conseguente incremento del numero di equazioni di bilancio necessarie a ricavare la disponibilità stazionaria. Un esempio di CTMC è mostrato in Figura 7.

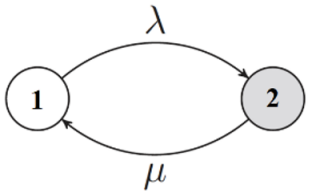

Figura 7: Esempio di un sistema a due stati modellato con il formalismo CTMC [12]

(iii) **Stochastic Reward Network** (SRN): formalismo introdotto dal gruppo di ricerca del prof. Trivedi [15], come estensione delle Generalized Stochastic Petri Networks (GSPN), che permette una rappresentazione dello stato del sistema in termini di posti e distribuzioni di "token" (dette *marking*), associando alle transizioni sia un meccanismo di timing stocastico (esponenziale) sia tempi di attivazione pari a 0 (transizioni immediate).
Le SRN possiedono i notevoli vantaggi di offrire un'interfaccia compatta di "alto livello" che nasconde i dettagli tecnici e, di poter modellare anche guasti di modo comune con le transizioni immediate. Inoltre, grazie all'introduzione delle funzioni di guardia che abilitano le transizioni, della cardinalità dell'arco variabile (in funzione della marcatura della SRN) e di rate e probabilità di transizioni che possono essere definiti anche come funzioni del marking della SRN, si riescono a modellare in maniera alquanto semplice anche sistemi reali complessi caratterizzati da situazioni del tipo *shared repair* o *indipendent repair*. Il contro di tale metodo è dato dalla difficoltà nell'accedere alla specifica distribuzione di stato sottostante.
Una SRN è caratterizzata da una *reward function X(t)*, la quale è un processo casuale non-negativo che rappresenta le condizioni del sistema. X(t) varia in accordo alla misura desiderata (es.: availability – il caso di nostro interesse –, dependability, performance). Per valutare la disponibilità, X(t) si definisce uguale a 1 in caso di sistema funzionante (*up* condition) in t, 0 altrimenti (*down* condition). La disponibilità istantanea si può esprimere come:

$$A(t) = \Pr\{X(t) = 1\} = E(X(t)) = \sum_{i \in S} r_i \cdot p_i(t) \tag{2}$$

dove: S è l'insieme dei marking, il quale 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, che si calcola come la somma su tutti i marking del prodotto tra il reward rate del marking i-esimo e la probabilità $p_i(t)$ di trovardi nel marking i-esimo.
Di fatto, si può dimostrare che il reward rate atteso istantaneo è uguale alla disponibilità istantanea $A(t)$, pertanto la disponibilità stazionaria è uguale al reward rate atteso stazionario, che è a sua volta uguale al limite per $t \to \infty$ del reward rate atteso istantaneo:

$$A = \lim_{t \to +\infty} A(t) = \sum_{i \in S} r_i \cdot p_i \tag{3}$$

dove p_i rappresenta la probabilità steady-state data da $p_i = \lim_{t \to +\infty} p_i(t)$. Un esempio di SRN è mostrato in Figura 8.

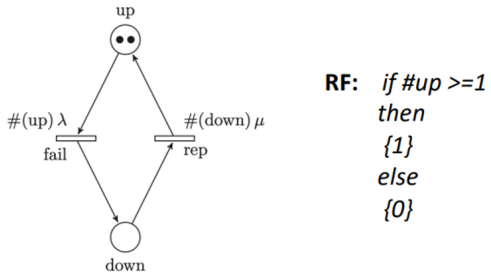

Figura 8: Esempio di un sistema a due componenti riparabili (independent repair) modellato con il formalismo SRN con relativa reward function [15]

(iv) **Reliability Block Diagram** (RBD): è un formalismo di alto livello per la modellazione della struttura logica di un sistema *monotono* i cui componenti, assunti statisticamente indipendenti, sono associati ciascuno ad un blocco. Se i componenti sono riparabili e tutti i loro processi di guasto e riparazione sono statisticamente indipendenti, il metodo RBD è applicabile anche per valutare la disponibilità. In particolare, per la valutazione di quest'ultima si prendono in considerazione come i componenti del sistema sono configurati (in serie o in parallelo) e la probabilità di funzionamento di ciascun componente. Nel caso di componenti in serie, tutti devono funzionare affinché l'intero sistema funzioni. In un RBD, questa configurazione viene rappresentata da una serie di blocchi collegati in sequenza. Per calcolare l'affidabilità (disponibilità) di un sistema in serie, si moltiplicano le affidabilità (disponibilità) di tutti i componenti. Per componenti configurati in parallelo, il sistema continua a funzionare finché almeno uno dei componenti funziona. In un RBD, questa configurazione è rappresentata da blocchi paralleli. Per calcolare l'affidabilità (disponibilità) di un sistema parallelo, si può utilizzare la formula: $1-\prod_i(1-R_i)$, dove R_i è l'affidabilità (disponibilità) di ciascun componente. Le strutture logiche in serie e quelle in parallelo sono spesso gli elementi di base di strutture più complesse. Un sistema composto da sottosistemi interconnessi in serie o strutture parallele è denominato *sistema serie-parallelo* e viene risolto attraversa il metodo della riduzione. I vantaggi nell'utilizzo di tale formalismo sono la semplicità con cui è possibile ricavare l'affidabilità (disponibilità) dell'intero sistema, a partire da quella dei singoli componenti e la possibilità di identificare quali componenti sono critici per la disponibilità del sistema e, quindi, dove potrebbero essere necessari miglioramenti o ridondanze. Gli svantaggi sono l'eccessiva semplificazione, in quanto si assume che i guasti dei componenti siano indipendenti e che i tassi di guasto siano costanti nel tempo, oltre che la mancanza di dettagli sul processo di riparazione dopo un guasto. Infine, essendo un formalismo di alto livello bisogna, spesso, affidarsi a valori di affidabilità (disponibilità) per ciascun componente forniti dai produttori. Un esempio di RBD è mostrato in Figura 9.

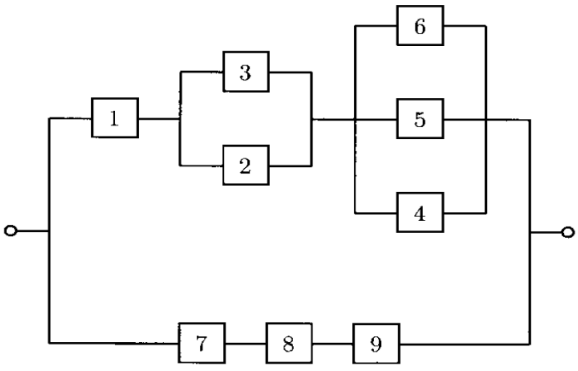

Figura 9: Esempio di RBD

4 Modello di disponibilità di un nodo SFC a 5-layer

L'obiettivo del progetto, data una SFC formata da 4 nodi connessi in serie ($N_1 - N_2 - N_3 - N_4$), consiste nell'individuazione della configurazione della SFC stessa che presenti la minima ridondanza tale da garantire una disponibilità stazionaria di almeno 0.999999, detta *six nines steady-state availability*. Per raggiungere tale scopo, pertanto, è imprescindibile fornire innanzitutto un modello di disponibilità di un singolo nodo della serie.

4.1 Descrizione della struttura 5-layer

Si assume che un nodo sia realizzato secondo il modello 5-layer rappresentato in Fig. 10.

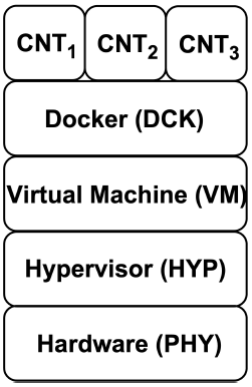

Figura 10: Astrazione di un nodo 5-layer dotato di 3 container

Si può osservare come la struttura a 5 strati si componga di:

- **Container (CNT)**: la funzionalità di rete softwarizzata che deve essere fornita. È rilevante evidenziare che il container level è caratterizzato dalla presenza di un numero di container uguale a 3. Spesso, infatti, per questioni di disponibilità, su un unico docker si mandano in esecuzione più istanze (repliche) di un singolo container.
- **Docker daemon (DCK)**: il container manager più diffuso [1]. Esso gestisce l'intero ciclo di vita di uno o più container;
- **Virtual Machine (VM)**: l'ambiente virtualizzato che ospita la struttura dei container;
- **Hypervisor (HYP)**: il layer di interfaccia tra l'hardware e i layer che vi sono collocati al di sopra;
- **Hardware (PHY)**: comprende tutti i componenti fisici, come processore, memoria, alimentatore, ecc.

In sostanza, sull'hardware si colloca un hypervisor; sull'hypervisor si colloca una virtual machine; sulla virtual machine c'è uno strato docker; e su quest'ultimo viene ospitata la funzionalità vera e propria sottoforma di container.

La scelta di adoperare il modello 5-layer per rappresentare ad alto livello un singolo nodo di una SFC è giustificata dal fatto che negli ultimi anni la tecnologia di virtualizzazione dei container venga sempre più utilizzata per fornire servizi di rete e di telecomunicazione. Infatti, implementare un'intera macchina virtuale (VM) per ospitare un singolo servizio di rete virtualizzato è costoso e comporta un inutile spreco di risorse. Un'alternativa valida è rappresentata proprio dai container, che sono pacchetti di codice software contententi il codice di un'applicazione, le sue librerie e altre dipendenze. Il vantaggio principale dei container rispetto alle macchine virtuali è che essi sono molto più "leggeri" poiché non richiedono un proprio sistema operativo dedicato. Alla luce di ciò, è possibile ospitare diversi container su una singola infrastruttura virtualizzata, a patto che sia garantito un grado sufficiente di isolamento [3].

4.2 Modello SRN

Si assume che i layer e i container del nodo siano rappresentati da modelli indipendenti Continuous-Time Markov Chain a due stati (funzionante/non funzionante), caratterizzati da 2 parametri: rate di guasto λ e rate di riparazione μ. Di fatto, la modellazione del nodo in termini di disponibilità stazionaria può essere agevolmente realizzata con la SRN illustrata in Fig. 11, di cui si riporta di seguito una descrizione dettagliata.

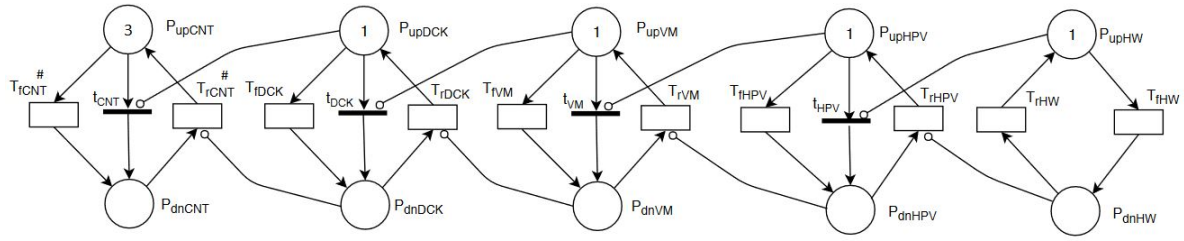

Figura 11: Modello SRN di un nodo SFC a 5-layer che include: container (CNT), docker daemon (DCK), virtual machine (VM), hypervisor (HYP), e hardware (PHY)

I posti P_{upCNT} $[P_{dnCNT}]$, P_{upDCK} $[P_{dnDCK}]$, P_{upVM} $[P_{dnVM}]$, P_{upHYP} $[P_{dnHYP}]$, P_{upPHY} $[P_{dnPHY}]$ si riferiscono rispettivamente alle condizioni di corretto funzionamento [failure] di container, docker daemon, virtual machine, hypervisor e hardware. Si osservi che ciascun posto contiene un singolo token (rappresentato dal numero 1), ad eccezione di P_{upCNT}, che ne contiene 3. Le transizioni T_{fCNT} $[T_{rCNT}]$, T_{fDCK} $[T_{rDCK}]$, T_{fVM} $[T_{rVM}]$, T_{fHYP} $[T_{rHYP}]$, T_{fPHY} $[T_{rPHY}]$, invece, indicano le attività di fallimento [riparazione] che caratterizzano rispettivamente container, docker daemon, virtual machine, hypervisor e hardware. Tali transizioni (rappresentate come rettangoli vuoti) sono dette "temporizzate" e, come precedentemente evidenziato, sono caratterizzate da tempi distribuiti esponenzialmente. Le transizioni t_{CNT}, t_{DCK}, t_{VM}, t_{HYP} (rappresentate come rettangoli neri e sottili), invece, sono dette "immediate" e indicano azioni con tempi di attivazione praticamente nulli.

Al fine di analizzare l'evoluzione di tale SRN, in cui gli eventi di fallimento e quelli di riparazione si verificano rispettivamente secondo i rate λ e μ, è utile porsi in una condizione iniziale di funzionamento in cui 3 token sono collocati in corrispondenza del posto P_{upCNT}, ed un solo token è presente in tutti gli altri posti "up". In caso di fallimento di un singolo container (si pensi, ad esempio, ad un reboot non controllato dello stesso), la transizione T_{fCNT} viene attivata ed un token viene trasferito da P_{upCNT} in P_{dnCNT}. Naturalmente, come conseguenza, 2 token rimangono in P_{upCNT}. Quando, invece, il container viene riparato, T_{rCNT} si attiva ed il token ritorna in P_{upCNT}. Ovviamente, il meccanismo appena descritto si ripete nel caso di fallimento di molteplici istanze di container, determinando lo spostamento verso P_{dnCNT} del numero pertinente di token. Si consideri ora il caso di fallimento del layer docker: la transizione T_{fDCK} si attiva ed il token che prima si trovava in P_{upDCK} si sposta in P_{dnDCK}. È interessante osservare che, quando il livello del docker fallisce, tutti i container diventano inattivi, dato che necessitano di un docker sottostante per poter essere in esecuzione. Tale criticità è modellata attraverso un arco inibitorio (raffigurato come un segmento che collega il posto P_{upDCK} alla transizione immediata t_{CNT} con un piccolo cerchio vuoto vicino a quest'ultima) che, in caso di fallimento del docker, forza l'attivazione proprio di t_{CNT}. Una volta che il docker viene riparato, T_{rDCK} si attiva, il token collocato in P_{dnDCK} passa in P_{upDCK}, quindi l'arco inibitorio che va da P_{dnDCK} a T_{rCNT} viene disabilitato, e i container si possono ristabilire secondo temporizzazione esponenziale. Comportamenti analoghi riguardano i fallimenti/riparazioni di virtual machine, hypervisor e hardware.

Inoltre, le transizioni T_{fCNT} e T_{rCNT} sono *marking dependent*: il rate di T_{fCNT} dipende dal numero di token contenuti nel posto P_{upCNT}, mentre il rate di T_{rCNT} dipende dal numero di token contenuti in P_{dnCNT}. Dal punto di vista pratico, ciò significa che i container sono tra loro indipendenti in termini di fallimenti/riparazioni.

Si evidenzia che l'unico layer sprovvisto di una transizione immediata è l'hardware. Infatti, dato che esso è l'ultimo layer della struttura a 5 strati rappresentata, non è necessario prendere in considerazione ulteriori dipendenze. Si può osservare che il sistema è **a cascata**: la logica risultante è tale per cui la caduta dell'hardware innesca una reazione a catena che comporta l'immediato crash anche di hypervisor, virtual machine, docker e di tutti i container.

4.2.1 Disponibilità stazionaria

Per il corretto funzionamento del nodo, c'è necessità che almeno uno dei 3 container sia funzionante ("up"). Per tale ragione, dato un marking i, il reward corrispondente r_i si può definire come:

$$r_i = \begin{cases} 1 & \text{se } \#P_{upCNT} \geq 1 \\ 0 & \text{altrimenti} \end{cases} \tag{4}$$

dove "#" si riferisce al numero di token.

Facendo il limite per $t \to \infty$ del valore atteso del reward rate istantaneo, è possibile ricavare la disponibilità stazionaria (steady-state availability) del nodo (vedi Eq. 3).

4.2.2 Confronto con eventuale modellazione tramite CTMC

Sulla base del confronto riportato nel Captilo 3 tra SRN e CTMC, si può affermare che, per il caso in esame, utilizzare una Markov Chain invece di una SRN sarebbe decisamente più complicato per varie ragioni:

(i) si dovrebbe collegare ciascuno stato ad uno stato "0", in cui nulla funziona nel sistema, attraverso degli archi caratterizzati da failure rate compatibili con le transizioni immediate, quindi, con dei λ molto alti al fine di far tendere gli esponenziali praticamente subito a zero;

(ii) il sistema di equazioni risultante (sia quello costituito dalle equazioni di bilancio sia quello costituito dalle equazioni differenziali, se interessati anche al transitorio) sarebbe molto complicato da risolvere, avendo un elevato numero di stati, vista la complessità del sistema.

5 Stima a massima verosimiglianza dei tassi di guasto e di riparazione del layer hardware

Per quanto concerne lo strato hardware (PHY) del nodo a 5-layer in esame non sono stati direttamente forniti i valori dei tassi di guasto λ_{PHY} e di riparazione μ_{PHY}.
Queste informazioni, infatti, possono essere ottenute da due fonti [12] [15]:

1. **modelli**: sulla base di ipotesi e del confronto con esperti del dominio, viene considerata una rappresentazione idealizzata (modello) del componente utile a definire i parametri necessari alla sua caratterizzazione, talvolta suggeriti dagli stessi fornitori.

2. **misurazioni**: raccolta dei dati necessari dall'osservazione fisica del componente mediante prove sul campo. Nel caso in esame (physical layer del nodo) tali dati sono già a disposizione per poter essere utilizzati in MATLAB;

Quindi, da un lato i modelli forniscono un utile strumento per guidare l'analisi di un sistema/componente in termini di affidabilità/disponibilità; d'altra parte, i modelli devono essere caratterizzati con dei dati qualora i loro parametri non siano noti[12].
Si è, dunque, provveduto alla stima dei tassi con un approccio a massima verosimiglianza, servendosi dei data set assegnati:

- un campione completo (di 177 valori, tutti esattamente noti) di tempi di riparazione per la stima di μ_{PHY};

- un campione censurato (time censoring o censura a destra di tipo I) di 200 tempi di guasto per la stima di λ_{PHY}, per il quale vi è solo una parte di dati esatti (177 valori), essendo i restanti 23 dati censurati a destra, ossia superiori ad una certa quantità fissata a priori (80 000 ore nel caso in esame) che rappresenta la durata massima del test eseguito. I campioni censurati si possono avere, infatti, a causa dell'interruzione dell'esperimento ad un tempo prefissato.

5.1 Procedura di stima a massima verosimiglianza

Per la stima dei due tassi si è nelle condizioni di poter utilizzare un approccio parametrico che prevede l'assunzione di un modello di distribuzione che, nel caso in esame, è quello Esponenziale per i tempi di guasto e di riparazione, trattandosi di un'unità tecnologica i cui guasti sono accidentali (non dovuti all'invecchiamento).
Sono disponibili diverse tecniche per le procedure di stima parametrica, sia analitiche (ad esempio: Least Squares, Maximum Likelihood (ML)) che grafiche (ad esempio: Probability Plots).
La tecnica di seguito adoperata è quella a massima verosimiglianza, che definisce la funzione di verosimiglianza *L(θ)* come proporzionale alla probabilità di ottenere proprio i dati osservati con il modello ipotizzato M(θ). Formalmente:

$$L(\theta) = c \cdot Pr[D; \theta] \tag{5}$$

dove D è il data set, θ è il vettore dei parametri da stimare e c è una costante di proporzionalità.
Lo *stimatore a massima verosimiglianza* (MLE) di θ è il valore $\hat{\theta}$ appartenente allo spazio dei parametri Θ che massimizza la funzione L(θ). Il criterio di massimizzazione della funzione di verosimiglianza, quindi, si basa sulla considerazione che la stima ML rappresenta il valore del parametro sconosciuto che massimizza la probabilità di occorrenza dei dati osservati, dato il modello di probabilità M(θ) [12]. Formalmente:

$$\hat{\theta} = \arg\max_{\Theta} L(\theta) \tag{6}$$

Siccome si sta guardando al massimo di una funzione, è equivalente guardare al massimo di una sua trasformazione monotona. Infatti, di solito è conveniente operare sulla trasformazione logaritmica (essendo le funzioni L(θ) spesso esponenziali) della verosimiglianza, detta *log-verosimiglianza* (log-likelihood).

Lo stimatore ML è una funzione di n campioni variabili aleatorie (v.a.) X_1,X_2,...,X_n. Essendo $\hat{\theta}$ una v. a., la conoscenza della sua funzione di distribuzione consente di valutare l'accuratezza della procedura di stima, per esempio utilizzando i suoi intervalli di confidenza (più è stretto, maggiore è la precisione).
La funzione di distribuzione degli stimatori ML, per *n finito*, dipende ovviamente dalla distribuzione della v.

a. X e dal tipo di campionamento. Pertanto, dovrebbe essere identificato (ove possibile) caso per caso.

Gli stimatori ML mostrano alcune utili proprietà asintotiche (per $n \to \infty$):

1. **consistenza**: convergono al valore vero del parametro, ossia $\hat{\theta} \to \theta$;
2. **asintoticamente efficiente**: sono a minima varianza;
3. **asintoticamente Normale**: la loro distribuzione converge ad una Normale;
4. **proprietà di invarianza**: sia η= g(θ) una trasformazione biiettiva del parametro θ. Il metodo ML ha la proprietà che, se $\hat{\theta}$ è un MLE di θ, allora $\hat{\eta}$= g($\hat{\theta}$) è il MLE di η. Per esempio, questa proprietà viene utilizzata per la stima dell'affidabilità. Quando la funzione affidabilità R(x; θ) è biunivoca in θ, lo stimatore ML dell'affidabilità al tempo di missione x è semplicemente $\hat{R}$(x) =R (x; $\hat{\theta}$).

Il metodo ML viene applicato anche in situazioni diverse dal caso di test indipendenti su n individui differenti, o comunque connesse al caso di n v. a. i.i.d. .

5.2 Modello Esponenziale

Sia X una v. a. che modella il tempo di vita di una unità (componente hardware nel caso in questione) in cui gli eventi di failure sono assunti indipendenti dall'età accumulata. Quindi, X è una Esponenziale con pdf data da

$$f(x;\theta) = \theta^{-1} \exp\left(-\frac{x}{\theta}\right), \quad x \geq 0, \quad \theta > 0 \tag{7}$$

dove θ è la vita media ed è il parametro che deve essere stimato. Dato un campione completo casuale di n osservazioni $\underline{x} = (x_1, x_2, \ldots, x_n)$ della v. a. X. La funzione di verosimiglianza è pari a

$$L(\theta; \underline{x}) = \prod_{i=1}^{n} \theta^{-1} \exp\left(-\frac{x_i}{\theta}\right) = \theta^{-n} \exp\left(-\theta^{-1} \sum_{i=1}^{n} x_i\right) \tag{8}$$

definendo $T^* = \sum_{i=1}^{n} x_i$, abbiamo una statistica sufficiente per la stima di θ. Si evince che si ha bisogno unicamente della somma dei campioni e non dei singoli campioni, cioè la statistica è sufficiente e minimale. Allora la funzione log-verosimiglianza può essere espressa come:

$$\log L(\theta; x) = -n \log \theta - \theta^{-1} T^* \tag{9}$$

Quindi, per calcolare la stima a massima verosimiglianza (MLE) dobbiamo imporre che

$$(\log L(\theta; x))' = l'(\theta) = 0 \tag{10}$$

ciò equivale a

$$\frac{\partial \log L(\theta)}{\partial \theta} = -n\theta^{-1} + T^* \theta^{-2} = 0 \tag{11}$$

e moltiplicando ambo i membri per θ (possibile in quanto $\theta > 0$ per ipotesi) abbiamo che la stima ML del parametro incognito è pari a

$$\hat{\theta} = \frac{T^*}{n} = \frac{\sum_{i=1}^{n} x_i}{n} \tag{12}$$

Il MLE è, quindi, pari alla media campionaria delle osservazioni, essendo in linea con il fatto che rappresenta il "valore atteso" di X.
Nel caso in cui si ha a disposizione un campione censurato a destra di tipo I (come per i tempi di guasto del componente hardware) con un unico tempo di censura τ, indicando con m il numero di unità realmente fallite entro il tempo τ, la statistica sufficiente che contiene i tempi di osservazione è

$$T^* = \sum_{i=1}^{m} x_i + (n-m)\tau \tag{13}$$

e lo stimatore ML è

$$\hat{\theta} = \frac{T^*}{m} \tag{14}$$

Osservando i risultati ottenuti per le differenti tipologie di dati, si può affermare che lo stimatore ML per il parametro θ è ottenuto, indipendentemente dal tipo di campionamento, dividendo il tempo di prova complessivo T^*, accumulato da tutte le n unità sotto test, per il numero di provini che sono falliti $m(\leq n)$. Tuttavia, il tipo di campionamento (e censura) influenza le proprietà statistiche degli stimatori risultanti.

5.3 Stima del tasso di riparazione μ_{PHY}

Si riporta di seguito la procedura di stima ML del tasso di riparazione μ_{PHY} effettuata con il seguente script MATLAB a partire dai dati a disposizione nel file "repairs_gr1.m", notando che si sono utilizzati più metodi equivalenti tra loro (le funzioni MATLAB *mean()*, *mle()* e *expfit()* con gli opportuni parametri) al fine di verificare la correttezza del risultato ottenuto:

Listato 5.1. Procedura di stima a massima verosimiglianza del tasso di riparazione a partire da un campione completo.

```
% Set seed
rng(2023);

% Carica il file .mat
data = load('repairs_gr1.mat');

% Determina il nome della variabile contenuta nel file
varName = fieldnames(data);

% Estrae i tempi di riparazione
repairTimes = data.(varName{1});

% Calcola il numero di campioni
nsamps = length(repairTimes);

% Stima del parametro mu utilizzando il metodo di massima verosimiglianza
%mu_hat = 1 / mean(repairTimes);
mu_hat = 1 / (sum(repairTimes) / nsamps);

% Exp distribution fit function
[muhat,muci] = expfit(repairTimes);

% mle() function
muml = mle(repairTimes,'Distribution','Exponential');

% Stampa il tasso di riparazione stimato
fprintf('Il tasso di riparazione stimato mu: %f\n', mu_hat);
fprintf('Il MTTR stimato: %f\n', 1/mu_hat);

% Istogramma vs pdf stimata
figure
histogram(repairTimes,'Normalization', 'pdf');
title('RepairTimes: complete sample', 'FontSize', 24)
xlabel('repairTimes', 'FontSize', 20)
ylabel('pdf', 'FontSize', 20)
ax = gca;
ax.FontSize = 16;
hold on;
xax = linspace(0, ax.XLim(2), 100);
plot(xax, exppdf(xax,muhat), 'r:', 'LineWidth', 2)
legend('hist','est.', 'FontSize', 20)
```

Output:
Il tasso di riparazione stimato μ è 0.195339 h^{-1}
Il MTTR stimato è 5.119315 h

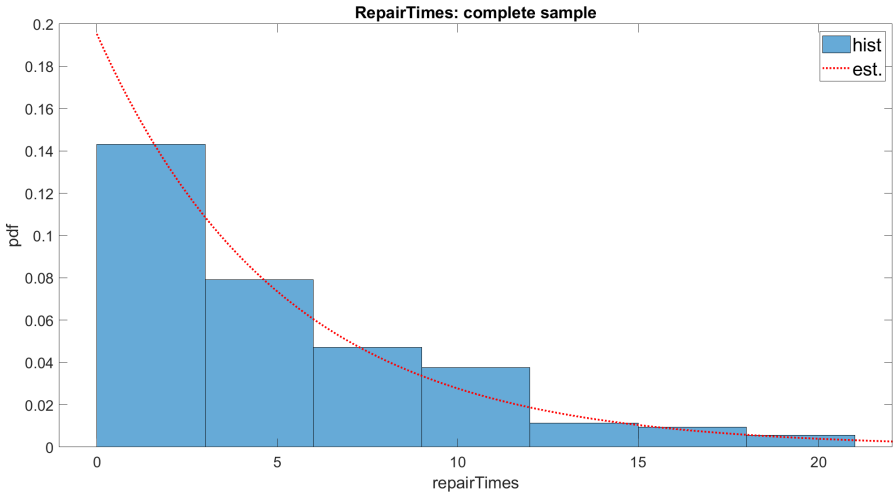

Figura 12: Pdf stimata vs istogramma dei tempi di riparazione

Osservando la Figura 12 si evince che la pdf stimata a partire dai dati forniti approssima bene l'andamento dell'istogramma dei tempi di riparazione, così come ci si aspettava.

5.4 Stima del tasso di guasto λ_{PHY}

Si riporta di seguito la procedura di stima ML del tasso di guasto λ_{PHY} effettuata con il seguente script MATLAB a partire dai dati a disposizione nel file "failures_gr1.m" censurati in accordo a quanto riportato nel file "censoring_gr1", notando che si sono utilizzati più metodi equivalenti tra loro (le funzioni MATLAB *sum()* fratto il numero di campioni non censurati e *mle()* con gli opportuni parametri) al fine di verificare la correttezza del risultato ottenuto:

Listato 5.2. Procedura di stima a massima verosimiglianza del tasso di guasto a partire da un campione censurato a destra (time censoring).

```
% Carica i file .mat
failureData = load('failures_gr1.mat');
censoringData = load('censoring_gr1.mat');

% Determina il nome delle variabili contenute nei file
failureVarName = fieldnames(failureData);
censoringVarName = fieldnames(censoringData);

% Estrae i tempi di guasto e i dati di censura
failureTimes = failureData.(failureVarName{1});
censoring = censoringData.(censoringVarName{1});

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

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

% Stima del parametro lambda utilizzando il metodo di massima verosimiglianza
lambda_hat = 1 / (sum(failureTimes) / nsamps_nocens);
lambda_wr=1/(sum(failureTimes)/nsamps_tot);

% mle() function with right censoring
mumlrc = mle(failureTimes,'Distribution','Exponential','Censoring',censoring);

% Stampa il tasso di guasto stimato
fprintf('Il tasso di guasto stimato lambda: %f\n', lambda_hat);
fprintf('Il MTTF stimato: %f\n', 1/lambda_hat);

```

```
% Istogramma vs pdf stimata e pdf stimata in modo errato con anche i tempi di stop
figure
histogram(failureTimes(find(censoring==0)),'Normalization', 'pdf');
title('Failure times: type I right censoring', 'FontSize', 24)
xlabel('failureTimes (uncensored)', 'FontSize', 20)
ylabel('pdf', 'FontSize', 20)
ax = gca;
ax.FontSize = 16;
hold on;
xax = linspace(0, ax.XLim(2), 100);
plot(xax, exppdf(xax,1/lambda_hat), 'r:', 'LineWidth', 2)
plot(xax, exppdf(xax,1/lambda_wr), 'k:', 'LineWidth', 2)
legend('hist','est.','wrong est. with stopping times', 'FontSize', 20)
```

Output:
Il tasso di guasto stimato λ è 0.000026 h^{-1}
Il MTTF stimato è 38178.438333 h

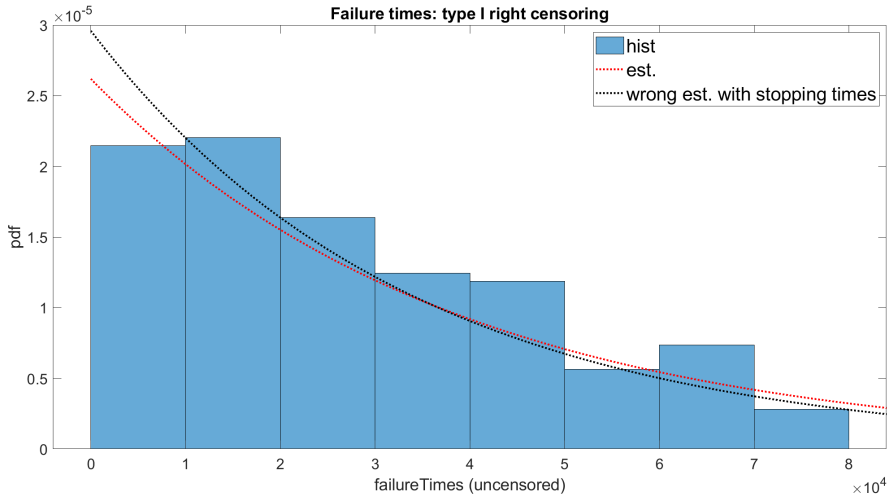

Figura 13: Pdf stimata vs istogramma dei tempi di guasto e pdf stimata in modo errato considerando anche i tempi di stop

Osservando la Figura 13 si evince che la pdf stimata a partire dai dati non censurati forniti approssima bene l'andamento dell'istogramma dei tempi di guasto, così come ci si aspettava. La pdf stimata in modo errato, vale a dire considerando anche il tempo a cui si è interrotto il test come tempi di guasto per tutte le unità censurate a destra, tende a sovrastare la pdf stimata correttamente fino a valori di tempo al guasto pari a circa 30 000 ore e a sottostimare da lì in avanti, essa, infatti, tende a zero più rapidamente, in quanto il λ stimato in tal caso è più grande, rispetto alla pdf stimata correttamente.

6 Analisi di disponibilità

Noti il modello SRN del singolo nodo 5-layer e tutti i rate di fallimento e riparazione, è possibile calcolare la disponibilità stazionaria della SFC formata da 4 nodi connessi in serie ($N_1 - N_2 - N_3 - N_4$) rappresentata in Fig. 14 con il formalismo RBD, e condurre successivamente una *availability analysis* atta all'individuazione di quella configurazione della SFC che presenti la minima ridondanza tale da garantire una disponibilità stazionaria di almeno 0.999999, detta *six nines steady-state availability*. Si evidenzia che per ridondanza di un nodo si intende la replicazione dell'intera struttura 5-layer rappresentata in Fig. 10.

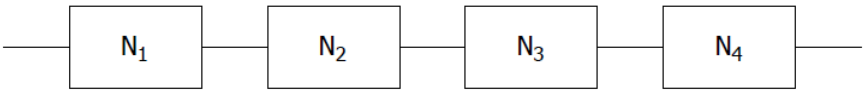

Figura 14: RBD della SFC formata dai 4 nodi connessi in serie ($N_1 - N_2 - N_3 - N_4$)

6.1 Disponibilità della serie ($N_1 - N_2 - N_3 - N_4$)

I rate di fallimento e riparazione considerati sono riportati nella Tabella 1. I valori sono realistici e motivati sia da letteratura tecnica (si guardino, per esempio, [10], [14]) sia da suggerimenti di esperti del settore delle infrastrutture di telecomunicazione virtualizzate [3]. È importante osservare che tutti i nodi sono caratterizzati da rate uguali per quanto concerne i livelli docker, virtual machine, hypervisor e hardware. Il rate di fallimento λ_{CNT} è lo stesso per tutti i container, indipendentemente dal nodo di appartenenza; il rate di riparazione μ_{CNTi} è lo stesso per tutti i container appartenenti al nodo i, e varia di nodo in nodo.

Tabella 1: Reciproco dei valori dei parametri di fallimento e riparazione

1/Parametro	**Valore**
$1/\lambda_{CNT}[h]$	800
$1/\lambda_{DCK}[h]$	1500
$1/\lambda_{VM}[h]$	2000
$1/\lambda_{HYP}[h]$	3000
$1/\lambda_{PHY}[h]$	38178.438333
$1/\mu_{CNTi-N1}[s]$	10
$1/\mu_{CNTi-N2}[s]$	10
$1/\mu_{CNTi-N3}[s]$	100
$1/\mu_{CNTi-N4}[h]$	2
$1/\mu_{DCK}[s]$	10
$1/\mu_{VM}[h]$	3
$1/\mu_{HYP}[h]$	3
$1/\mu_{PHY}[h]$	5.119315

Il calcolo della disponibilità di ciascun nodo, modellato con la SRN di Fig. 11, è stato condotto utilizzando il tool Sharpe [13]. Una valida alternativa è rappresentata dal software TimeNET [18] I risultati ottenuti sono riportati nella Tabella 2.

Tabella 2: Disponibilità stazionaria dei nodi coinvolti nella SFC

Nodo	**Disponibilità stazionaria**
N_1	0.99621419
N_2	0.99621419
N_3	0.99620152
N_4	0.99520173

Si può notare che i nodi N_1 ed N_2 godono della stessa disponibilità stazionaria, che tale disponibilità è maggiore di quella garantita dal nodo N_3, e che quest'ultima è maggiore di quella garantita dal nodo N_4.

Ciò si verifica perché i 4 nodi sono caratterizzati esattamente dagli stessi rate di fallimento e riparazione, ad eccezione dei μ_{CNTi}, tra i quali sussiste la seguente relazione: $\mu_{CNT1} = \mu_{CNT2} < \mu_{CNT3} < \mu_{CNT4}$

La disponibilità stazionaria della serie si può calcolare come segue:

$$A = \prod_{j=1}^{N} A^{(j)} = 0.98392901 \tag{15}$$

dove $A^{(j)}$ è la disponibilità stazionaria del nodo j-esimo, ed N, che è il numero di componenti connessi in serie, è naturalmente uguale a 4.

È banale osservare che la disponibilità stazionaria della serie è inferiore rispetto a quella dei singoli nodi. Infatti, l'indisponibilità di anche uno solo dei 4 nodi determina l'indisponibilità dell'intera struttura serie, quindi l'impossibilità di erogare il servizio di rete. Inoltre, si può notare come il valore ottenuto sia molto lontano da quello desiderato, ovvero dai *six nines*.

6.2 Individuazione della configurazione *six nines*

L'analisi di disponibilità prosegue con l'individuazione di quella configurazione della SFC che presenti la minima ridondanza tale da garantire il soddisfacimento della specifica sulla disponibilità stazionaria di almeno 0.999999, che corrisponde ad un'indisponibilità di circa 0.5256 minuti all'anno [9] e, pertanto, costituisce un requisito molto stringente.

Prima di entrare nel dettaglio dell'analisi, è rilevante evidenziare che si è assunto che tutti i nodi della SFC avessero esattamente lo stesso "costo", che si intende come quantità adimensionale associata all'impiego di una risorsa. Ciò significa che ciascun nodo ha un costo uguale a 1. Tale assunzione deriva dalla mancanza di specifiche di progetto che differenziassero in termini di costi i nodi coinvolti nella SFC da implementare. Naturalmente, nodi diversi possono avere in generale costi diversi, principalmente in base al loro valore economico e alla criticità che assumono nell'ambito della SFC. Inoltre, nonostante non abbia un impatto significativo sull'analisi di disponibilità condotta, poiché le repliche introdotte sono esclusivamente replicazioni di intere strutture 5-layer, è rilevante menzionare che il costo che solitamente si assegna ad un container (es: applicazione software con licenza associata) ammonta a 0.5, mentre quello che viene attribuito all'infrastruttura sottostante (es: virtual machine, sistema operativo, ecc.) è uguale a 1 [3].

6.2.1 Algoritmo availability_calculator

Al fine di velocizzare l'individuazione delle configurazioni ridondate capaci di garantire la specifica desiderata sulla disponibilità stazionaria, si è deciso di implementare in Python un algoritmo, chiamato *availability_calculator*, il cui pseudocidice è riportato sotto (Alg. 2). Il file Python di riferimento è `ava_ass.py`.

Algorithm 1 availability_calculator

Input: $ava1$, $ava2$, $ava3$, $ava4$, $desiredAva$, $maxRedundancy$
Output: Configurations with availability $\geq desiredAva$

```
1:  begin
2:      for i = 1 to maxRedundancy do
3:          for j = 1 to maxRedundancy do
4:              for k = 1 to maxRedundancy do
5:                  for v = 1 to maxRedundancy do
6:                      avaRedundant_i ← 1 − (1 − ava1)^i
7:                      avaRedundant_j ← 1 − (1 − ava2)^j
8:                      avaRedundant_k ← 1 − (1 − ava3)^k
9:                      avaRedundant_v ← 1 − (1 − ava4)^v
10:                 end for
11:             end for
12:         end for
13:     end for
14:     avaSeries ← avaRedundant_i * avaRedundant_j * avaRedundant_k * avaRedundant_v
15:     if avaSeries ≥ desiredAva then
16:         print(i, j, k, v)
17:     end if
18: end
```

La logica dell'algoritmo è estremamente semplice. L'input è costituito dalle disponibilità stazionarie dei nodi N_1, N_2, N_3, N_4 riportate nella Tabella 2; dalla disponibilità stazionaria complessiva desiderata, che nel caso considerato è 0.999999; e dal massimo grado di ridondanza accettabile per ciascun nodo. Quest'ultimo parametro ha l'obiettivo di limitare lo spazio di ricerca delle configurazioni ottimali ed è stato impostato a 6. Per ciascuna delle possibili configurazioni da testare, l'algoritmo calcola la disponibilità stazionaria della serie costituita dai paralleli delle repliche di ciascun nodo. Infatti, se la disponibilità stazionaria di una serie di componenti si calcola adoperando la 15, la disponibilità stazionaria di un parallelo si calcola nel modo seguente:

$$A = 1 - \prod_{i=1}^{N} A^{(j)} \tag{16}$$

dove $A^{(j)}$ è la disponibilità stazionaria del nodo j-esimo, mentre N è il numero di componenti connessi in parallelo. Nel caso considerato, i componenti connessi in parallelo sono sempre repliche di uno stesso nodo.

Dopo aver calcolato la disponibilità complessiva della configurazione, l'algoritmo valuta semplicemente se questa soddisfi o meno il requisito dei *six nines*.

6.2.2 Risultati

La configurazione a ridondanza minima che garantisce il soddisfacimento della specifica dei *six nines* è quella costituita da 3 repliche per ciascun nodo. Nella Tabella 3 si riportano il valore della disponibilità stazionaria di tale configurazione (prima riga) e quelli delle configurazioni che presentano rispetto ad essa una sola replica in più. Si osservi che i valori delle disponibilità sono riportati in ordine crescente dall'alto verso il basso.

Tabella 3: Disponibilità stazionaria di alcune configurazioni ridondate che soddisfano i *six nines*

Configurazione	**Livello di ridondanza**	**Disponibilità stazionaria**
C_1	$N_1 = 3,\ N_2 = 3,\ N_3 = 3,\ N_4 = 3$	0.9999997262
C_2	$N_1 = 3,\ N_2 = 4,\ N_3 = 3,\ N_4 = 3$	0.9999997803
C_3	$N_1 = 4,\ N_2 = 3,\ N_3 = 3,\ N_4 = 3$	0.9999997803
C_4	$N_1 = 3,\ N_2 = 3,\ N_3 = 4,\ N_4 = 3$	0.9999997808
C_5	$N_1 = 3,\ N_2 = 3,\ N_3 = 3,\ N_4 = 4$	0.9999998361

Si può osservare che le configurazioni C_2 e C_3 offrono esattamente la stessa disponibilità , il che dipende dal fatto che i nodi N_1 ed N_2 sono caratterizzati dalla stessa disponibilità. Inoltre, la configurazione che tra quelle riportate garantisce la disponibilità più elevata è C_5, il che è dovuto al fatto che il nodo che in essa è replicato più volte, ovvero N_4, è proprio quello con disponibilità più bassa, ovvero il *bottleneck* della SFC.

La configurazione a minima ridondanza che soddisfa i *six nines*, ovvero C_1, è anche illustrata, secondo il formalismo di alto livello RBD, in Fig. 15.

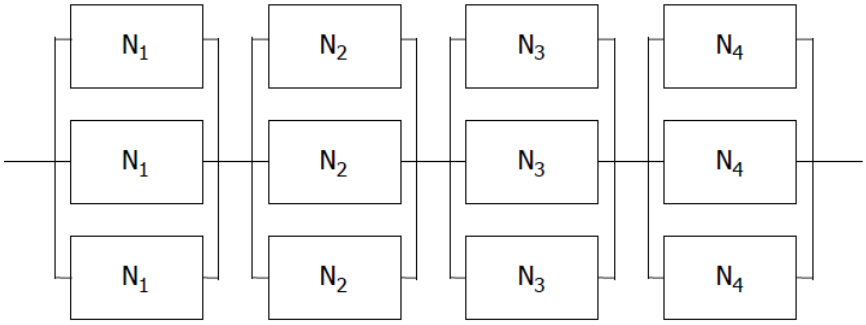

Figura 15: RBD della SFC della configurazione C_1, che è quella a minima ridondanza tale da garantire una disponibilità stazionaria complessiva di almeno 0.999999

Naturalmente, tra le configurazioni riportate in Tabella 3, C_1 è quella che presenta disponibilità stazionaria minore, in quanto, rispetto alle altre, dispone di una replica in meno di almeno uno dei 4 nodi della SFC. Nonostante ciò, C_1 garantisce con ampio margine il requisito dei *six nines*, infatti presenta un'indisponibilità annuale di circa 8.64 secondi.

Inoltre, in Fig. 16 si propone un confronto tra le indisponibilità $(1 - A)$ delle configurazioni considerate. Si rileva che la configurazione caratterizzata da maggiore indisponibilità stazionaria è proprio C_1; le configurazioni $C2$, $C3$, $C4$ si assestano su valori analoghi ($C2$, $C3$ addirittura uguali per le considerazioni fatte sopra); C_5, naturalmente, garantisce la minore indisponibilità, che corrisponde a poco più di 5 secondi all'anno.

A conclusione dell'analisi di disponibilità, è interessante osservare che, qualora il requisito fosse stato il raggiungimento dei *five nines* invece dei *six nines*, il risultato finale cui si sarebbe giunti sarebbe stato il medesimo, in quanto non sono emerse configurazione ridondate con disponibilità stazionaria esattamente uguale ai *five nines*, ma solo inferiori o superiori.

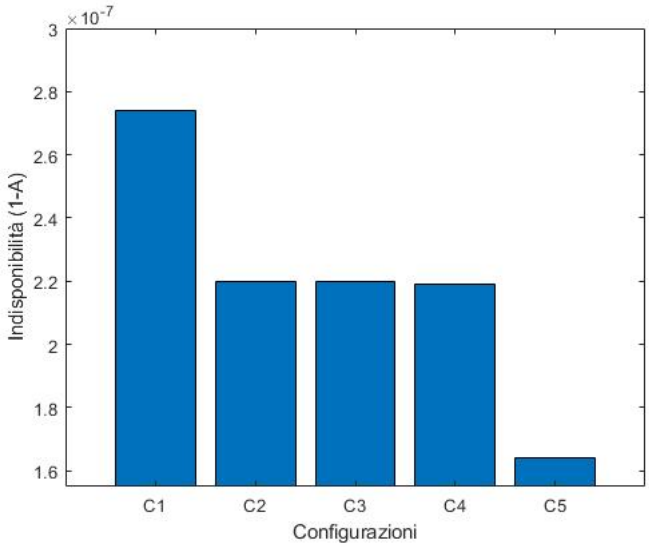

Figura 16: Indisponibilità della SFC per 5 diverse configurazioni ridondate

6.2.3 Validazione dei risultati in TimeNET

Al fine di validare i risultati in Sharpe, viene implementato il modello SRN presentato in Figura 11 in TimeNET. I risultati ottenuti confermano quanto già ottenuto in precedenza con Sharpe.
Per l'ottenimento dei valori di disponibilità ottenuti da ogni nodo (si veda Tabella 2), è stata creata la funzione MATLAB `extract_from_TimeNET`. La procedura realizzata è presente in Figura 17.

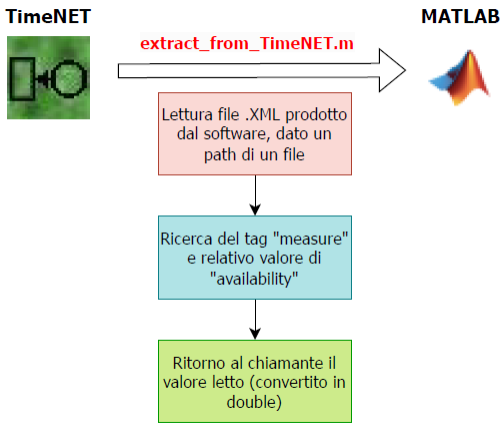

Figura 17: Estrazione da TimeNET della disponibilità

In particolare, sono stati creati 4 modelli di SRN (identificati come `N1.xml`, `N2.xml`, `N3.xml` e `N4.xml`) opportunamente configurati con i valori in Tabella 1.
L'operazione di estrazione consiste nella lettura da ognuno dei file TimeNET del valore di disponibilità calcolato importandolo nel Workspace di MATLAB. Una volta ottenute le disponibilità dei singoli nodi, viene valutata la disponibilità usando `availability_SoP`, che ha lo scopo di calcolare la disponibilità complessiva dell'intero sistema partendo dalla disponibilità dei singoli nodi, data la configurazione del sistema. Ad esempio, se ogni nodo N_i con $i \in \{1, 2, 3, 4\}$ viene replicato 3 volte, allora la configurazione viene espressa come una sequenza *3 3 3 3*. Una descrizione del comportamento di `availability_SoP` è fornita in Figura 18.

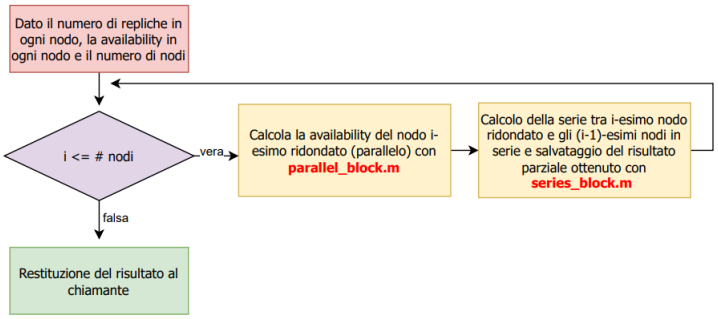

Figura 18: Calcolo della disponibilità dell'intero sistema per un certo numero di nodi in serie

Si osserva che è necessario, al fine dell'ottenimento della disponibilità dell'intero sistema, calcolare serie e paralleli in accordo alla configurazione esaminata. A tal proposito sono state realizzate le funzioni `series_block` e `parallel_block`. Poiché, sia nel caso di paralleli sia nel caso di serie, il task è quello di calcolare delle produttorie delle singole disponibilità in maniera opportuna, le sopracitate funzioni si avvalgono della funzione `product_block` (si veda Figura 19). In conclusione, possiamo riepilogare il processo di estrazione da TimeNET come segue:

1. estrazione delle singole disponibilità da ogni nodo e caricamento nel Workspace di MATLAB;
2. calcolo della disponibilità complessiva utilizzando `availability_SoP`.

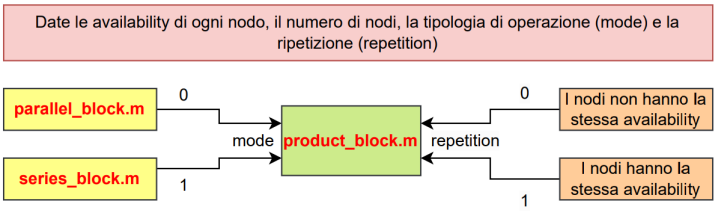

Figura 19: Funzione di servizio per il calcolo della disponibilità nel caso di blocchi in serie o in parallelo. Si osserva che questa funzione prima di poter essere invocata ha bisogno di 2 dati rilevanti: la *modalità* (serie o parallelo) e la *ripetizione*, che evidenzia se i nodi coinvolti sono tutti caratterizzati da identica disponibilità o meno

In definitiva, ogni qual volta sia necessario variare un parametro di ognuno dei nodi, è necessario che vengano dapprima calcolati i valori nuovi di disponibilità all'interno di TimeNET e in un secondo momento procedere alla valutazione della disponibilità del sistema. Quanto detto ora, viene riassunto brevemente in Figura 20.

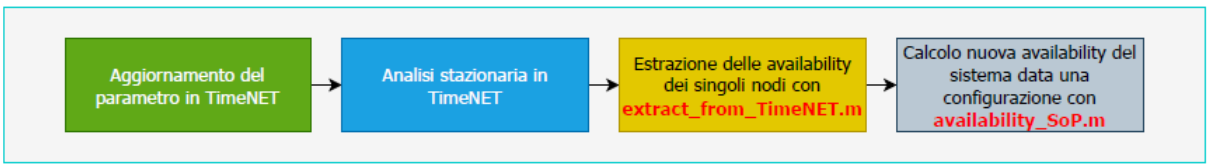

Figura 20: Processo di estrazione di un nuovo valore di disponibilità da TimeNET

Si osserva come il processo di acquisizione dei dati sia abbastanza complesso, tuttavia, il ristretto numero di test da effettuarsi è esiguo e con l'unico scopo di testare l'algoritmo proposto in 2.

Inoltre, sono stati validati anche i risultati ottenuti sulle configurazioni ridondate. Anche in questo caso, si osserva che i risultati sono coerenti con quelli mostrati in Tabella 3. A tal proposito, è stata creata la funzione MATLAB `find_configurations` che permette di ottenere tutte le configurazioni ridondate a partire dalle disponibilità stazionarie dei singoli nodi superiori ad una certa soglia. Il comportamento ottenuto è sintetizzabile in Figura 21 ed è basato sull'algoritmo in 2.

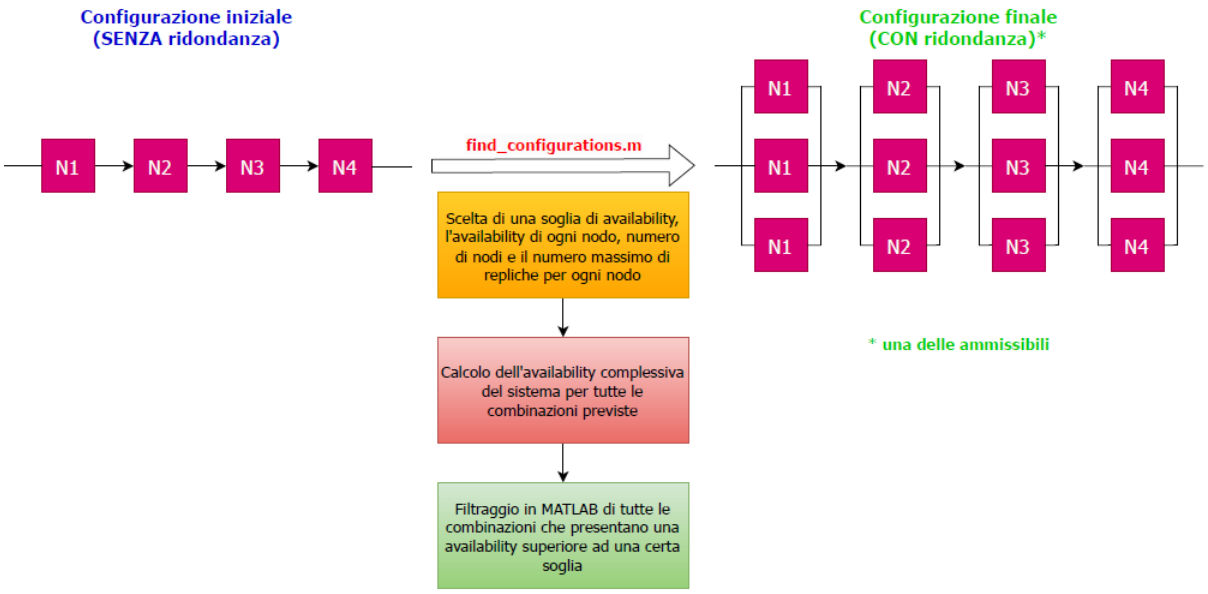

Figura 21: Ricerca delle configurazioni ridondate per la validazione del risultato

In conclusione, in questo paragrafo abbiamo verificato due risultati prodotti in precedenza con Sharpe:

1. validazione della disponibilità complessiva del sistema serie ($N_1 - N_2 - N_3 - N_4$). Utilizzando opportunamente le chiamate a `extract_from_TimeNET` e `availability_SoP` si ottiene dalla prima la disponibilità stazionaria di ogni nodo e dalla seconda la disponibilità stazionaria complessiva dell'intero sistema. Quindi, sono state unite le rispettive chiamate a queste procedure con lo script MATLAB chiamato `check_sys`.

2. validazione delle configurazioni ridondate del sistema serie ($N_1 - N_2 - N_3 - N_4$). In particolare, la definizione dei parametri necessari per l'utilizzo `find_configurations` e la sua invocazione è stata inserita in `evaluate_conf` producendo le configurazioni indiduate in Tabella 3.

7 Analisi di sensitività

Generalmente, un'analisi di sensitività ha l'obiettivo di valutare la robustezza di un sistema rispetto a deviazioni di uno o più parametri dai loro valori nominali [6]. Di fatto, i valori dei parametri di fallimento e riparazione messi a disposizione dai fornitori talvolta possono rivelarsi fin troppo ottimistici, ragion per cui è estremamente utile determinare come vari la disponibilità stazionaria dell'intero sistema in funzione del rilassamento di quei parametri ritenuti di maggiore importanza.

Nell'ambito del progetto, secondo quanto richiesto, si è condotta dapprima un'analisi di sensitività al variare del parametro λ_{DCK}, assumendo costanti e uguali ai valori nominali tutti gli altri (si faccia riferimento alla Tabella 1); successivamente è stata condotta un'analisi di sensitività al variare del parametro μ_{DCK}, sempre assumendo costanti e uguali ai valori nominali tutti gli altri, compreso λ_{DCK}. In entrambi i casi l'obiettivo è stato l'individuazione del valore critico del parametro sotto esame, ovvero il valore per cui la disponibilità stazionaria del sistema scende al di sotto di quella fissata dalle specifiche, che in questo caso è 0.999999. Si riportano di seguito i risultati ottenuti e la descrizione della procedura automatizzata adottata per ottenerli.

7.1 Risultati

I risultati sono illustrati nelle Fig. 22 e 23, in cui si può osservare che il reciproco del failure rate del livello docker, ovvero λ_{DCK}, può essere rilassato fino a 131.39 h circa (rispetto alle 1500 h nominali) senza disattendere il requisito dei *six nines*, che è indicato dalla linea orizzontale tratteggiata. Similmente, si osserva che il reciproco del repair rate del livello docker, ovvero μ_{DCK}, può essere rilassato fino a 1.47 h circa (rispetto ai 10 s nominali) senza disattendere il requisito dei *six nines*.

L'ampio margine, in entrambi i casi, è giustificato dal fatto che, nonostante la configurazione individuata sia quella a ridondanza minima, la sua steady-state availability supera in modo sensibile la soglia dei six nines, essendo circa uguale a 0.9999997.

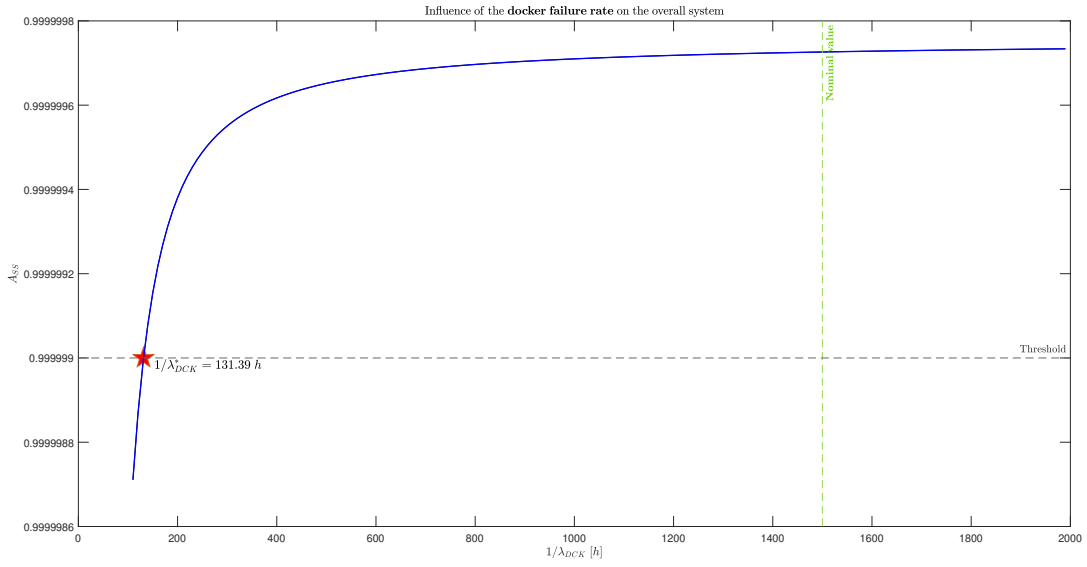

Figura 22: Analisi di sensitività sul parametro λ_{DCK} con individuazione del valore critico λ^*_{DCK}

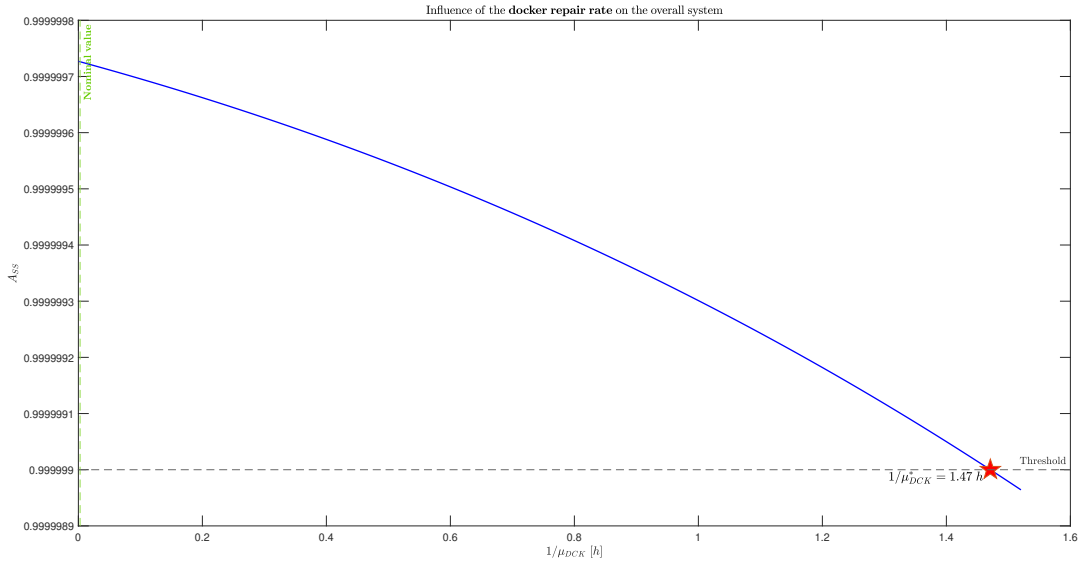

Figura 23: Analisi di sensitività sul parametro μ_{DCK} con individuazione del valore critico μ^*_{DCK}

7.2 Focus sulla procedura adottata

Al fine di garantire elevata precisione nell'individuazione dei valori critici e di fornire uno strumento riutilizzabile anche per eventuali progetti futuri, si è deciso di progettare e implementare una procedura automatizzata per l'analisi di sensitività. Essa sfrutta soprattutto la possibilità di interagire direttamente da linea di comando con il tool Sharpe attraverso delle funzioni appositamente scritte in Python. All'individuazione del valore critico del parametro sotto esame (es: μ_{DCK}), realizzata da un'implementazione in Python dell'algoritmo *sensitivity_calculator* (vedi Alg. 2), segue la costruzione di un file di testo, che riporta una serie di coppie del tipo (reciproco del rate del parametro sotto esame, disponibilità stazionaria associata), compresa quella (valore critico, soglia considerata). Tale file viene quindi utilizzato da una procedura MATLAB per generare un grafico simile a quelli riportati nelle Fig. 22 e 23.

7.2.1 Algoritmo sensitivity_calculator

L'individuazione del valore critico viene effettuata con l'ausilio del seguente algoritmo:

Algorithm 2 sensitivity_calculator

Input: *nameFiles*, *margin*, *configuration*, *lower*, *upper*, *step*, *avaTarget*, *transToChange*, *direction*
Output: The value of *transToChange* for which the availability is equal to *avaTarget* with an arbitrary *margin*

```
1: begin
2:     if direction is equal to "ifUpperWorse" then
3:         for each testedValue in the range from lower to upper with step step do
4:             avaUpdateFile(nameFiles, testedValue, transToChange)
5:             avaSeries ← computeAva(nameFiles, configuration)
6:             if avaSeries < avaTarget then
7:                 if avaTarget − avaSeries ≤ margin then
8:                     print("CRITICAL VALUE OF transToChange FOUND")
9:                     output testedValue
10:                else
11:                    print("The critical value of transToChange is close but not enough")
12:                    newLower = testedValue − step
13:                    newUpper = testedValue
14:                    output sensitivity_calculator(nameFiles, margin, configuration, newLower,
15:                                newUpper, step/10, avaTarget, transToChange, direction)
16:                end if
17:            end if
18:        end for
19:        print("End of cycle: upper must be increased")
20:        output sensitivity_calculator(nameFiles, margin, configuration, lower,
21:                    upper ∗ 2, step/10, avaTarget, transToChange, direction)
22:    else
23:        for each testedValue in the range from upper to lower with step − step do
24:            avaUpdateFile(nameFiles, testedValue, transToChange)
25:            avaSeries ← computeAva(nameFiles, configuration)
26:            if avaSeries < avaTarget then
27:                if avaTarget − avaSeries ≤ margin then
28:                    print("CRITICAL VALUE OF transToChange FOUND")
29:                    output testedValue
30:                else
31:                    print("The critical value of transToChange is close but not enough")
32:                    newLower = testedValue
33:                    newUpper = testedValue + step
34:                    output sensitivity_calculator(nameFiles, margin, configuration, newLower,
35:                                newUpper, step/10, avaTarget, transToChange, direction)
36:                end if
37:            end if
38:        end for
39:        print("End of cycle: lower must be decreased")
40:        output sensitivity_calculator(nameFiles, margin, configuration, lower/2,
41:                    upper, step/10, avaTarget, transToChange, direction)
42:    end if
43: end
```

L'algoritmo sopra riportato prende in input:

- *nameFiles*: lista dei nomi dei file contenenti le descrizioni Sharpe dei nodi considerati.
- *margin*: massima differenza accettabile tra la disponibilità specificata da *avaTarget* e quella calcolata.

- *configuration*: lista di interi rappresentante la configurazione ridondata da valutare (es: [3,3,3,3]).
- *lower*: estremo inferiore dell'intervallo di valori da testare come reciproco del rate della transizione temporizzata *transToChange*.
- *upper*: estremo superiore dell'intervallo di valori da testare come reciproco del rate della transizione temporizzata *transToChange*.
- *step*: intervallo tra i valori da testare come reciproco del rate della transizione temporizzata *transToChange*.
- *avaTarget*: disponibilità stazionaria che deve essere raggiunta nel contesto dell'analisi di sensitività condotta.
- *transToChange*: la transizione temporizzata sotto test.
- *direction*: specifica la direzione secondo cui la variazione di *transToChange* determina la diminuzione della disponibilità stazionaria del sistema. Pertanto, deve essere "ifUpperWorse" nel caso di rate di riparazione e "ifLowerWorse" nel caso di rate di fallimento.

e restituisce il valore critico, ovvero il valore di *TransToChange* per il quale la disponibilità stazionaria calcolata è uguale a quella oggetto dell'analisi di sensitività (nel caso di questo progetto si tratta di 0.999999), con un margine di errore arbitrariamente configurabile.

L'algoritmo opera ricorsivamente, richiamando sé stesso con i parametri di ingresso *lower*, *upper* e *step* appositamente aggiornati con l'obiettivo di avvicinarsi sempre di più ad *avaTarget*. In particolare, *sensitivity_calculator* esegue o il blocco *if* più esterno o il suo blocco *else* corrispondente, a seconda del valore assunto da *direction*; per ogni valore di *transToChange* testato, richiama la procedura *avaUpdateFile()*, la quale modifica tutti i file Sharpe dei nodi considerati, aggiornando al loro interno proprio il valore attribuito a *transToChange*; richiama la procedura *computeAva()* al fine di calcolare la disponibilità stazionaria dell'intero sistema sfruttando Sharpe; verifica che la disponibilità ottenuta sia inferiore a quella target. In tal caso, valuta se la differenza tra di esse sia minore del margine impostato, e se ciò è vero restituisce il valore usato come reciproco di *transToChange*, in quanto questo costituisce proprio il valore critico ricercato. Se, invece, la differenza tra il risultato di *computeAva()* e *avaTarget* è maggiore di *margin*, allora l'algoritmo richiama sé stesso con i parametri *lower* e *upper* modificati, in modo da definire un intorno sempre più stretto intorno al valore critico, e con uno *step* più piccolo.

Si osservi che l'algoritmo è progettato in modo tale da riuscire sempre a convergere, anche nel caso in cui l'intervallo iniziale definito dai parametri *lower* e *upper* non comprenda al suo interno il valore critico. Infatti, se un intero ciclo termina senza mai entrare nel blocco "**if** *avaSeries* < *avaTarget*", l'algoritmo richiama sé stesso con dei parametri *lower* e *upper* opportunamente modificati, nella speranza che il nuovo intervallo contenga il valore critico.

Le implementazione in Python di *sensitivity_calculator* e delle procedure che richiama, ovvero *avaUpdateFile()* e *computeAva()*, sono contenute nel file `ava_sens.py`.

7.2.2 Estrazione dei risultati e plot in MATLAB

Al fine di mostrare l'andamento della disponibilità stazionaria (Fig. 22 e 23) al variare dei parametri λ_{DCK} e μ_{DCK}, è necessario che vengano opportunamente caricati nel *Workspace* di MATLAB i valori dei parametri unitamente alle disponibilità stazionarie corrispondenti.
La funzione `ava_write_csv`, contenuta nel file `ava_sens.py`, ha il compito di scrivere in un file di testo le disponibilità steady-state di una certa configurazione ridondante per diversi valori del reciproco del rate di una specifica transizione temporizzata. In particolare, il file di testo prodotto contiene coppie del tipo (reciprocal_of_rate, availability).
Partendo dai file di testo generati al termine delle esecuzioni della funzione Python `ava_write_csv` per il failure e repair rate, questi vengono letti in MATLAB con la funzione `extract_from_SHARPE`.
Quest'ultima viene invocata da `save_data_from_SHARPE` che, partendo dai nomi dei file di ogni parametro da analizzare, salva come MATLAB data `.mat` sia i valori dei parametri sia i valori di disponibilità stazionaria raggiunti. Il comportamento della funzione `save_data_from_SHARPE` è descritto in Fig. 24. Nel caso in esame, i `.mat` creati sono così chiamati:

- `availability_obtained_lambda_SHARPE` per i valori di disponibilità raggiunti in funzione di λ_{DCK}

- `availability_obtained_mu_SHARPE` per i valori di disponibilità raggiunti in funzione di μ_{DCK}
- `values_lambda_SHARPE` contiene i valori di λ_{DCK}
- `values_mu_SHARPE` contiene i valori di μ_{DCK}

Successivamente, i file `.mat` vengono importati grazie allo script MATLAB `sensitiviy_plot_SHARPE` nel *Workspace* e gli andamenti della disponibilità al variare dei parametri λ_{DCK} e μ_{DCK} vengono mostrati a video. Il comportamento della funzione `save_data_from_SHARPE` è descritto in Fig. 25

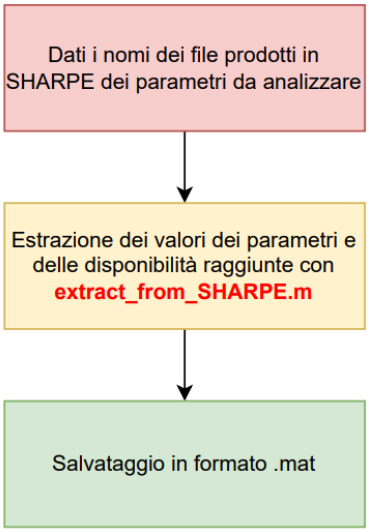

Figura 24: Funzione `save_data_from_SHARPE`

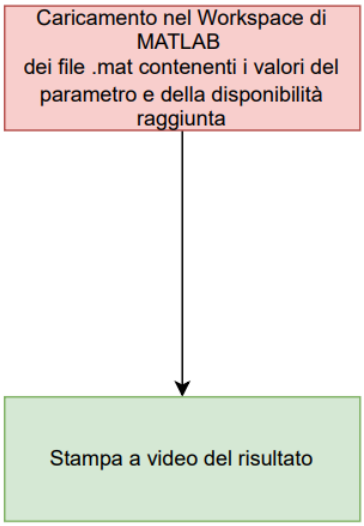

Figura 25: Funzione `sensitiviy_plot_SHARPE`

8 Conclusioni

In conclusione, dopo aver delineato una panoramica del contesto tecnologico, un attento confronto tra i metodi stocastici tipicamente utilizzati per la valutazione della disponibilità stazionaria, una dettagliata descrizione del modello di disponibilità del singolo nodo SFC a 5-layer e, la procedura di stima a massima verosimiglianza dei parametri mancanti, si è svolta una rigorosa analisi di disponibilità, individuando la configurazione a minima ridondanza che rispetta il requisito di disponibilità stazionaria richiesto. In particolare, tale analisi è stata condotta avvalendosi di due tool (Sharpe e TimeNET) così da validare ulteriormente i risultati ottenuti. Infine, si è effettuata un'attenta analisi di sensitività, al fine di esaminare la robustezza del sistema ridondato alle deviazioni rispetto ai valori nominali di due parametri critici. Quest'ultima analisi, come la precedente, ha adoperato entrambi i tool già citati, confermando in tal modo la coerenza dei risultati conseguiti.

Riferimenti bibliografici

[1] Docker. available online: https://www.docker.com, accessed: 2023-06-07.

[2] Network functions virtualization. Page Version ID: 110249471.

[3] M. Di Mauro, G. Galatro, M. Longo, F. Postiglione, and M. Tambasco. IP multimedia subsystem in a containerized environment: availability and sensitivity evaluation. In *2019 IEEE Conference on Network Softwarization (NetSoft)*, pages 42–47. IEEE.

[4] M. Di Mauro, M. Longo, F. Postiglione, G. Carullo, and M. Tambasco. Service function chaining deployed in an NFV environment: An availability modeling. In *2017 IEEE Conference on Standards for Communications and Networking (CSCN)*, pages 42–47.

[5] Mario Di Mauro, G Galatro, M Longo, F Postiglione, and M Tambasco. Availability evaluation of a virtualized IP multimedia subsystem for 5g network architectures. In *Safety and Reliability – Theory and Applications*, pages 321–321. CRC Press.

[6] Mario Di Mauro, Maurizio Longo, Fabio Postiglione, and Marco Tambasco. Availability modeling and evaluation of a network service deployed via NFV. In Alessandro Piva, Ilenia Tinnirello, and Simone Morosi, editors, *Digital Communication. Towards a Smart and Secure Future Internet*, volume 766, pages 31–44. Springer International Publishing. Series Title: Communications in Computer and Information Science.

[7] ETSI. Network functions virtualisation (NFV): Architectural framework.

[8] James F. Kurose, Keith W. Ross, Antonio Capone, and Sabrina Gaito. *Reti di calcolatori e internet. Un approccio top-down. Ediz. mylab. Con eText. Con aggiornamento online*. Pearson, 7° edizione edition.

[9] Jeremy Langston. System availability benchmarking-a survey, 2007.

[10] Rubens de S Matos, Paulo RM Maciel, Fumio Machida, Dong Seong Kim, and Kishor S Trivedi. Sensitivity analysis of server virtualized system availability. *IEEE Transactions on Reliability*, 61(4):994–1006, 2012.

[11] Mario Di Mauro, Giovanni Galatro, Fabio Postiglione, and Marco Tambasco. Performability of network service chains: Stochastic modeling and assessment of softwarized IP multimedia subsystem. 19(5):3071–3086.

[12] Fabio Postiglione and Mario Di Mauro. Appunti dal corso di system safety engineering a.a. 2022/2023.

[13] Robin A Sahner and Kishor S Trivedi. Reliability modeling using sharpe. *IEEE Transactions on Reliability*, 36(2):186–193, 1987.

[14] Stefano Sebastio, Rahul Ghosh, and Tridib Mukherjee. An availability analysis approach for deployment configurations of containers. *IEEE Transactions on Services Computing*, 14(1):16–29, 2018.

[15] Kishor S. Trivedi and Andrea Bobbio. *Reliability and Availability Engineering: Modeling, Analysis, and Applications*. Cambridge University Press, 1 edition.

[16] Kishor S Trivedi, Steve Hunter, Sachin Garg, and Ricardo Fricks. Reliability analysis techniques explored through a communication network example. 1996.

[17] Ying Zhang. *Network Function Virtualization: Concepts and Applicability in 5G Networks*. John Wiley & Sons. Google-Books-ID: pqJFDwAAQBAJ.

[18] Armin Zimmermann, Jörn Freiheit, Reinhard German, and Günter Hommel. Petri net modelling and performability evaluation with timenet 3.0. In *Computer Performance Evaluation. Modelling Techniques and Tools: 11th International Conference, TOOLS 2000 Schaumburg, IL, USA, March 27–31, 2000 Proceedings 11*, pages 188–202. Springer, 2000.