Sommario

L’oggetto del lavoro svolto nel tirocinio è la progettazione e l’implementazione di metodi per la simulazione ad eventi discreti di reti di container.

La relazione è divisa in 5 punti fondamentali: l’introduzione alle reti di container e alla simulazione; i concetti necessari per comprendere il lavoro svolto; i metodi implementati spiegati con annesso pseudocodice per ogni algoritmo; i risultati sperimentali dove i metodi vengono implementati e messi alla prova con un sistema reale; e infine viene ipotizzato un possibile miglioramento della gestione della simulazione del tempo dei container che potrebbe rendere maggiormente raffinata la simulazione.

Indice

Capitolo 1

Introduzione

1.1. Sistemi di reti di container e simulazione

Immaginiamo di essere responsabili per lo sviluppo di un sistema militare che ha come scopo quello di controllare un insieme di droni con lo scopo di sorvegliare punti di interesse, da ora in poi il sistema prendere il seguente nome: Centro Di Controllo Droni. L'obbiettivo principale è quello di creare un sistema in grado di essere sicuro, scalabile e che sia in grado di gestire più reti di droni per diversi territori.

Poniamo il caso in cui sia necessario controllare diversi punti d'interesse in territori completamente diversi e distanti, come gestire il sistema in modo da soddisfare le esigenze di latenza per tutte le reti di droni? Come gestire diverse famiglie di insiemi di droni in modo da garantire un tempo di risposta accettabile per ogni rete? Come gestire il caso in cui un componente dell'architettura del sistema diventi indisponibile? Come gestire il sovraccarico di un singolo servizio?

Lo scenario proposto descrive una delle possibili situazioni del mondo reale, dove l'utilizzo di sistemi basati su reti di container sta iniziando a diventare lo standard per lo sviluppo di sistemi in grado di soddisfare tutte le esigenze richieste.

Ma cosa sono le reti di container? Naturalmente prima di parlare delle reti di container bisogna spiegare che cosa è un container, i container sono un tipo di tecnologia che permettono l'esecuzione del software in un ambiente isolato consentendo di pacchettizzate tutti gli elementi necessari per l'esecuzione degli applicativi in un qualsiasi ambiente. Da questa descrizione i container sembrano molto simili alle macchine virtuali, ma al di esse dove la virtualizzazione avviene a livello hardware, nei container la virtualizzazione avviene al livello del sistema operativo, questo porta innumerevoli vantaggi come una virtualizzazione più leggera, un utilizzo delle risorse del sistema ridotto e una più efficiente gestione del software.

Le reti di container sono un modo per connettere tra di loro i container permettendone la comunicazione, creando un'infrastruttura separata dalla rete fisica, che consente una comunicazione efficiente tra i container e le applicazioni.

Ma in sostanza quali sono i vantaggi si usare sistemi basati su reti di container?

- *Maggiore scalabilità*. L'uso di container rende possibile aggiungere o rimuovere risorse di elaborazione, creando o eliminando istanze in modo flessibile, in modo da adattare la capacità alle esigenze dell'applicazione e del carico di lavoro, senza dover creare una nuova istanza dell'applicazione o della macchina virtuale.
- *Migliore tolleranza ai guasti.* Essendo i container facilmente replicabili, nell'eventualità che un container vada offline, un altro container potrebbe essere già pronto per non perdere e continuare il lavoro. Questo rende lo sviluppo di applicazioni *fault tolerance* più veloce ed efficiente.

- *Sicurezza.* Il container costituisce una sorta di bolla, isolato dagli altri container in esecuzione sulla stessa infrastruttura fisica. Ciò significa che anche se un contenitore viene compromesso da un attacco informatico, l'accesso ai dati e alle risorse di altri contenitori all'interno della stessa rete viene impedito.
- *Portabilità.* Una volta che il servizio è implementato all'interno del container esso può funzionare praticamente ovunque, su sistemi operativi Linux, Windows e Mac, su macchine virtuali o server fisici, sul cloud, tutto questo senza la necessità di modificare una riga di codice.

Ma perché l'utilizzo della simulazione ad eventi discreti di reti di container? La simulazione di sistemi è una tecnica per simulare il comportamento di sistemi fisici utilizzando un sistema modellato per poterne studiare il comportamento. Grazia a ciò la simulazione permette di effettuare la validazione e la verifica del sistema preso in esame, per verificare le risorse necessarie, testare il sistema con diversi scenari operativi e studiarne il comportamento; Inoltre, permette di ridurre i costi e i rischi associati alla sperimentazione su sistemi reali.

L'utilizzo degli eventi per la simulazione permette numerosi vantaggi, tra cui i principali:

- *Velocità di simulazione.* Visto che il durante la simulazione il tempo è simulato con istanti di tempo ben definiti, il tempo simulato è completamente indipendente dal tempo reale, questo porta il tempo di simulazione a dipendere solo dalla velocità della macchina su cui viene svolta la simulazione. Di conseguenza in pochi minuti o secondi è possibili simulare l'esecuzione di diversi giorni del sistema.
- *Indipendenza dal sistema di simulazione*. Essendo la simulazione ad eventi discreti le variabili cambiano di stato istantaneamente in ben definiti istanti di tempo ed è indipendente il sistema su cui gira la simulazione, visto che il tempo di esecuzione di ogni componente risulta essere sempre indipendentemente dalla macchina su cui gira la simulazione. Questo permette che l'output del test con stesso input risulterà essere sempre lo stesso.
- *Indipendenza dallo stato della rete.* Grazie all'utilizzo della simulazione ad eventi discreti è possibile costruire il testing definendo l'istante di tempo necessario per lo scambio di messaggi tra i componenti, indipendentemente dal reale stato della rete. Questo permette anche di creare diversi scenari operativi in base allo stato della rete.
- *Interruzione simulazione.* Grazie al fatto che il tempo avanza in base all'accadimento di eventi, la simulazione può essere temporaneamente interrotta per verificare lo stato del sistema, per poi riprendere la simulazione.
- *Valutazione risorse.* La possibilità di assegnare il valore di tempo discreto di elaborazione per ogni componente permette di valutare le risorse necessarie per implementare il sistema in modo da soddisfare i requisiti richiesti.

1.2. Perché la necessità di creare dei metodi per la simulazione?

Nonostante l'utilizzo di sistemi basati su reti di container ormai sia ampiamente utilizzati, ancora oggi non esisto degli standard per gestire la risoluzione dei problemi che emergono nella simulazione ad eventi discreti di reti di container. Come gestire il tempo durante la simulazione? Come gestire il tempo tra i diversi container in modo che si possano scambiare informazioni? Come gestire la memoria condivisa? Meglio la simulazione ad incrementi di tempo fissati o guidata dagli eventi? Come devono essere modellati i client, e come deve essere gestita la loro simulazione? Questi e molti altri problemi, che approfondiremo in seguito, hanno portato alla ricerca di un insieme di metodi in modo creare degli standard per la simulazione di reti di container.

1.3. Cosa permettono i metodi ideati.

Il lavoro svolto nella relazione propone dei metodi generali per riuscire non solo i problemi intrinsechi nella simulazione di reti di container, ma andando anche a permettere di valutare il funzionamento, il comportamento e le prestazioni del sistema. Il tutto permettendo una simulazione molto veloce dove sarà possibile simulare il comportamento del sistema di una settimana in pochi minuti.

Capitolo 2

Conoscenze necessarie

2.1 Introduzione alla simulazione

Un sistema fisico contiene un insieme di nozioni sull'evoluzione degli stati durante il passare del tempo, come ad esempio il numero di droni che stanno sorvegliando i punti di interesse in un determinato momento, per tornare all'esempi del Centro Di Controllo Droni. Di conseguenza la simulazione deve provvedere (1) la rappresentazione degli stati dei diversi componenti del sistema, (2) la modellazione del cambiamento di questi stati, e una (3) rappresentazione del tempo. Per le necessita (1), è necessario definire una collezione di variabili di stato, utilizzando un linguaggio di programmazione, in modo che suddette variabili rappresentino i diversi stati delle componenti del sistema. Per affrontare (2), i cambiamenti del sistema fisico vengono semplicemente modellati realizzando la simulazione in modo da permettere il cambiamento delle variabili di stato. Infine (3), il tempo del sistema fisico è rappresentato attraverso una astrazione chiamata tempo di simulazione. Si discuterà di questo nel prossimo punto.

2.2. Tempo

Esistono diverse nozioni importanti di tempo quando si parla di simulazione. É imperativo mantenere queste nozioni distinte, visto che potrebbero creare confusione. Di seguito le definizioni che verranno utilizzate in seguito:

Tempo fisico si riferisce al tempo fisico del sistema durante la simulazione.

Tempo simulato è una astrazione usata dal modello di simulazione per modellare il tempo fisico.

Tempo di wallclock si riferisce al tempo reale durante l'esecuzione del programma di simulazione. Ad esempio, il tempo, in secondi, dal momento che la simulazione inizia a quando finisce.

Per illustrare al meglio i concetti di tempo, consideriamo la simulazione del Centro Di Controllo Droni per un'intera settimana partendo dal lunedì. In questo caso il tempo fisico della simulazione parte dal giorno uno (lunedì) al giorno sette (domenica). Il tempo simulato potrebbe essere simulato con un numero a virgola mobile, dove ogni unità corrisponde ad un secondo del tempo fisico. Poniamo il caso che l'intera esecuzione del programma di simulazione inizi alle ore 13:00:00 e finisca alle 13:45:27 dello stesso giorno, il tempo di wallclock sarà di 45 minuti e 27 secondi, ciò il tempo per eseguire l'intera simulazione. Per quanto riguarda il tempo fisico e di wallclock sono essenzialmente nozioni di tempo usate nel modo convenzionale e di facile comprensione. Quando si parla

di tempo simulato si fa riferimento ad un nuovo concetto che esiste solo nel mondo della simulazione.

Definizione: Il *tempo simulato* è definito come un insieme totalmente ordinato di valori, dove ogni valore rappresenta un istante del tempo fisico del sistema modellato. Prendiamo due valori del tempo di simulazione, T1 che rappresenta l'istante di tempo fisico P1, e T2 che rappresenta l'istante di tempo fisico P2, se T1 < T2, allora P1 si verifica prima di P2, e (T2 – T1) è equivalente a (P2 – P1) • K per un qualche valore costante di K. [1]

La relazione diretta tra gli intervalli del tempo di simulazione e gli intervalli di tempo del sistema fisico garantisce la corretta corrispondenza tra la durata del tempo simulato e la durata del tempo fisico.

2.3. Cambiamento di stato e Maccanismo di avanzamento del tempo

I modelli di simulazione si differenziano in due tipologie quando si fa riferimento al modo in cui gli stati del modello cambiano in base al passare del tempo simulato.[2]

- *Modelli continui,* dove gli stati del sistema variano con continuità con il passare del tempo.

- *Modelli discreti,* dove gli stati del sistema variano ad istanti ben definiti di tempo.

Nella simulazione ad eventi discreti si può concettualmente pensare al guardare dei fotogrammi di un video che rappresentano l'evoluzione del sistema, dove ogni foto rappresenta il cambiamento di stato del sistema avvenuto tra un istante di tempo e l'altro. È chiaro che nella simulazione ad eventi discreti sia necessario un meccanismo che definisca come far procedere il valore del tempo simulato da un istante al successivo. I due metodi più comuni utilizzati come maccanismo di avanzamento del tempo sono:

- Simulazione ad avanzamento del tempo ad incrementi prefissati.

- Simulazione guidata dagli eventi.

-

2.3.1. Simulazione ad incrementi prefissati

Quando si parla di avanzamento del tempo ad incrementi prefissati, la simulazione è suddivisa in una sequenza di istanti di tempo tutti uguali, cioè presa la sequenza di istanti di tempo, dell'intera simulazione, che chiamiamo in questo caso T si ha che T è uguale a *t1,t2,…..,tk*; preso un qualsiasi istante di tempo ti, si avrà che *(ti+1) - ti = k* con *k* costante fissata uguale a (*tempo di simulazione* / *numero di istanti*). La simulazione ad incrementi prefissati partirà dall'istante *t1* per poi passare al *t2*, fino a completare la sequenza, ad ogni

[1] Richard M. Fujimoto, *Parallel and Distributed Simulation Systems*, John Wiley & Sons, 2000, pp. 28
[2] M. Roma, *Sistemi di Servizio e Simulazione*, http://www.diag.uniroma1.it/~roma/didattica/SSS19-20/parte2.pdf

istante t si considerano tutti i cambiamenti del sistema avvenuti in quell'istante come se fossero avvenuti contemporaneamente; quindi, per una corretta simulazione, è necessario modellare la simulazione in modo che se due eventi si realizzano nello stesso momento nel sistema reale, gli stessi devono realizzarsi nello stesso istante t nel sistema modellato.

2.3.2. Simulazione guidata dagli eventi

Nella simulazione ad incrementi prefissati lo stato del sistema viene elaborato ad ogni incremento, anche se non avviene nessun cambiamento tra un istante all'altro, questo porta ad un inutile spreco di calcolo e tempo. Nella simulazione guidata dagli eventi il tempo simulato avanza ogni qualvolta accade un evento, passando da un evento all'altro fino alla fine della simulazione. Ma cos'è un evento? Un evento è un'astrazione che rappresenta un cambiamento di stato del sistema fisico, quindi il cambiamento di un insieme di variabili nel sistema modellato. Ad esempio, un drone del sistema Centro Di Controllo droni si mette nella coda d'attesa di ricevere il punto d'interesse da sorvegliare, ecco questo è un evento. Ad ogni evento viene associato un valore, che chiameremo *tempo di realizzazione*, che rappresenta il momento, nel tempo di simulazione, in cui un evento avviene. Di conseguenza il tempo di simulazione viene di volta in volta aggiornato partendo dall'evento con il minor tempo do realizzazione per passare al successivo evento con minor tempo di realizzazione ancora non realizzatosi. Si può pensare ad una lista di eventi ordinata in base al tempo di realizzazione, che val valore minore al maggiore. Quel che molto semplicemente avviene durante la simulazione è partire dal primo evento della lista, aggiornare il tempo di simulazione con il valore del tempo di realizzazione dell'evento e aggiornare lo stato del sistema in base all'evento per poi passare al secondo evento della lista e ripetendo il ciclo finché la simulazione non termina.

2.3. Comunicazione nelle reti di container

Quando si parla di reti di container solitamente si parla di sistemi di tipo client-server, dove l'insieme di container rappresenta se il server e client che rappresentano l'applicazione o il dispositivo che invia le richieste. Esisto due principali metodi per gestire la comunicazione tra sistema e client, la comunicazione diretta e quella indiretta.[3]

2.3.1. Comunicazione diretta

Un possibile approccio all'architettura di comunicazione diretta, che prevede che i client inviino le richieste direttamente ai container. In questo approccio per ogni servizio è previsto un endpoint pubblico, che esegue il mapping all'API che gestisce i container relativi al servizio richiesto, bilanciandone il carico e gestendo il numero di istanze.

[3] https://learn.microsoft.com/it-it/virtualization/windowscontainers/about/

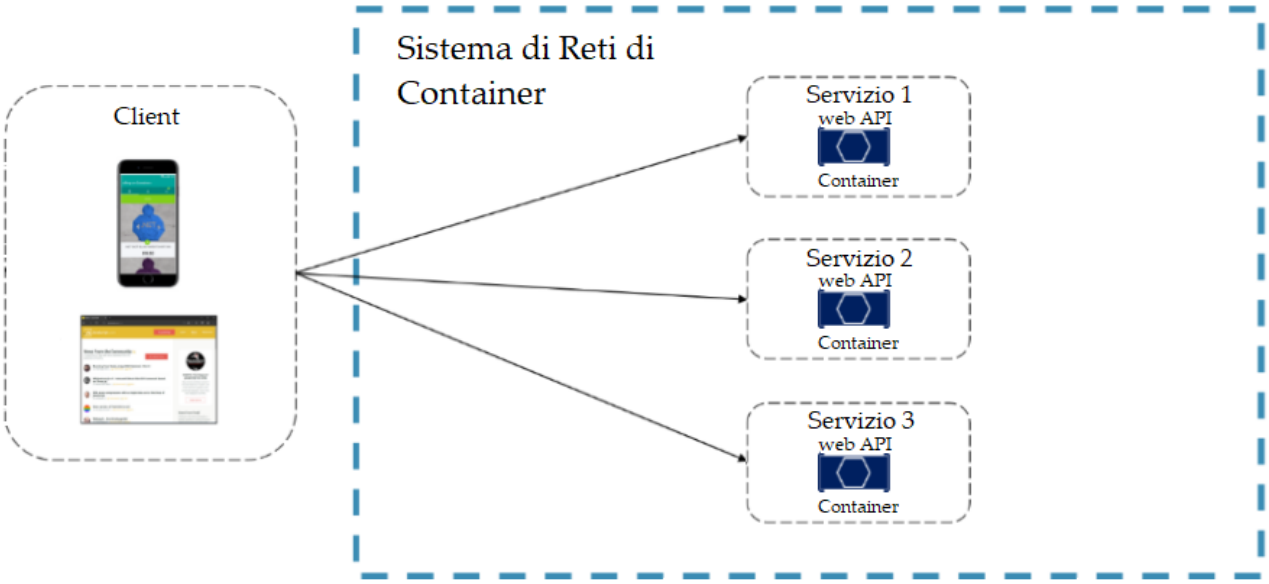

Figura 2.1: Architettura reti di container comunicazione diretta

L'utilizzo della comunicazione diretta tra client e container può portare a diversi problemi, ad esempio:

- *Accoppiamento:* tramite la comunicazione diretta ogni client deve conoscere l'architettura del sistema, e quindi ogni volta che invia una richiesta esso verrà associato ad un servizio offerto dal container. Questo rende complessa l'evoluzione del software, visto che nel caso di *refactoring* interni sarà necessario aggiornare ogni volta l'applicativo lato client.
- *Sicurezza:* mettendo in comunicazione diretta i client con ogni servizio si rende la superficie d'attacco molto ampia, questo rende più complessa la gestione per la prevenzione di eventuali attacchi.
- *Mancata aggregazione richieste:* una singola richiesta da parte del client potrebbe richiede di dover interrogare diversi servizi per poter essere soddisfatta. Questo potrebbe portare ad una latenza significativa, visto il numero di round trip, nel caso di interrogazione di diversi servizi, diventerebbe elevato.
- *Monitoraggio:* La comunicazione diretta tra client e servizi può rendere più difficile il monitoraggio del sistema. Rendendo più complessa la gestione degli errori e la verifica delle prestazioni.

2.3.1. Comunicazione indiretta

Quando si sviluppano applicazioni basate su servizi, che possono essere grandi e complesse, può essere utile utilizzare un'architettura basata su un API Gateway. Questo tipo di architettura prevede l'utilizzo di un servizio centralizzato che funge da punto di ingresso unico per un gruppo di servizi specifici.
Il Gateway API si posiziona tra le app client e i servizi, e agisce come un proxy inverso che inoltra le richieste dei client ai servizi appropriati. Il Gateway API semplifica la gestione delle richieste dei client e fornisce una maggiore sicurezza e scalabilità al sistema complessivo.

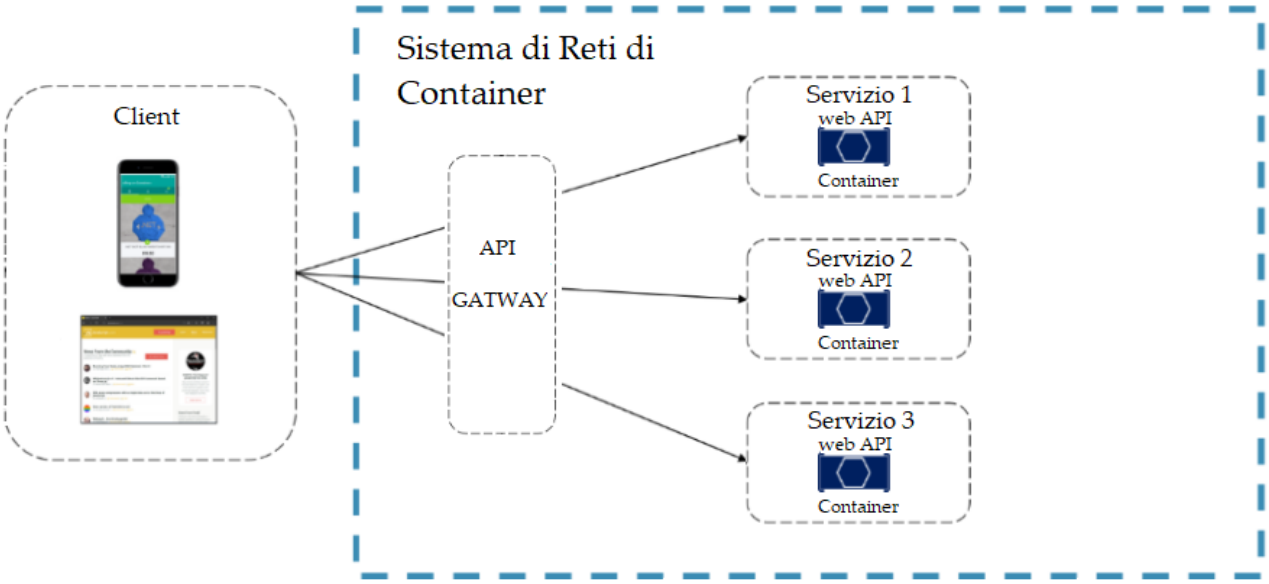

Figura 2.2: Architettura reti di container comunicazione indiretta

Capitole 3

Metodi per la simulazione

3.1. Metodi e algoritmi per la simulazione di reti di container

Lo scopo di questa sezione è quella di descrivere tutti i metodi e algoritmi ideati per riuscire a creare un sistema per la simulazione che si adatti a qualsiasi tipo di sistema basato su di reti di container, partendo dall'architetture, come gestire il tempo, come gestire gli eventi.

3.2. Architettura sistema di simulazione

Quando si pensa alla simulazione come prima cosa deve essere presa in considerazione il sistema che si vuole simulare ed analizzare l'architettura per comprendere come implementare tutti i metodi necessari per la corretta simulazione. Tenendo conto che la tesi si basa sulle reti di container, il tipo di architettura su cui ci concentreremo maggiormente sarà di tipo client-server.

3.2.1. Sistema che si vuole simulare

Per quanto riguarda il sistema che si vuole simulare non sono richieste particolari cambiamenti all'architettura, non è necessario costruire alcun modello, ma basterà implementare i metodi che verranno mostrati in seguito. L'unico importante cambiamento è quello dell'utilizzo del tempo simulato, questo significa che nel se nell'architettura del sistema è previsto un database, e questo database usa delle variabili di tempo, esse dovranno essere convertite con il valore del tempo simulato. Questo vale anche per tutte le variabili di tempo all'interno del codice del sistema.

3.2.2. Creazione del Client

Per quanto riguarda i client è necessario costruire il modello che possa simularne il comportamento. Questo significa rispettare il comportamento reale del client, creando un modello in grado di rispondere agli stimoli dell'ambiente e del sistema in modo equivalente al client reale. Inoltre, è molto importante assegnare ad ogni compito il tempo simulato equivalente a quello reale necessario per svolgere quel determinato compito.

La costruzione di un modello di client equivalente a quello reale è un'operazione molto complessa che richiede diverso tempo nel testare il modello e nello studio del comportamento del client. Ritornando all'esempio del Centro di Controllo Droni, la creazione di un modello di un drone richiede diverse conoscenze che vanno dalla velocità del drone al tempo di scarica e carica della batteria in varie situazioni, in questo caso oltre allo studio fisico del client è necessario studiare l'ambiente e come si comporta il drone in base al cambiamento di stati che può subire. Ad esempio, come si comporta la batteria se il territorio da sorvegliare è particolarmente caldo? Come similare il comportamento del drone quando ha il vento contro? E quando ha il vento a favore? E quando piove? E

quando nevica? Come si può immaginare, per certi tipi di applicazione, una delle sfide principali è quella di riuscire a costruire un modello che riesca a simulare correttamente il client fisico, per fortuna non è un compito che riguarda la tesi in questione.

3.2.3. Architettura simulazione e Gestore tempo ed eventi

Il componente centrale del sistema di simulazione, centrale nel vero e proprio senso della parola, è il gestore del tempo e degli eventi. Centrale perché deve essere implementato come *layer* tra il sistema e i client.

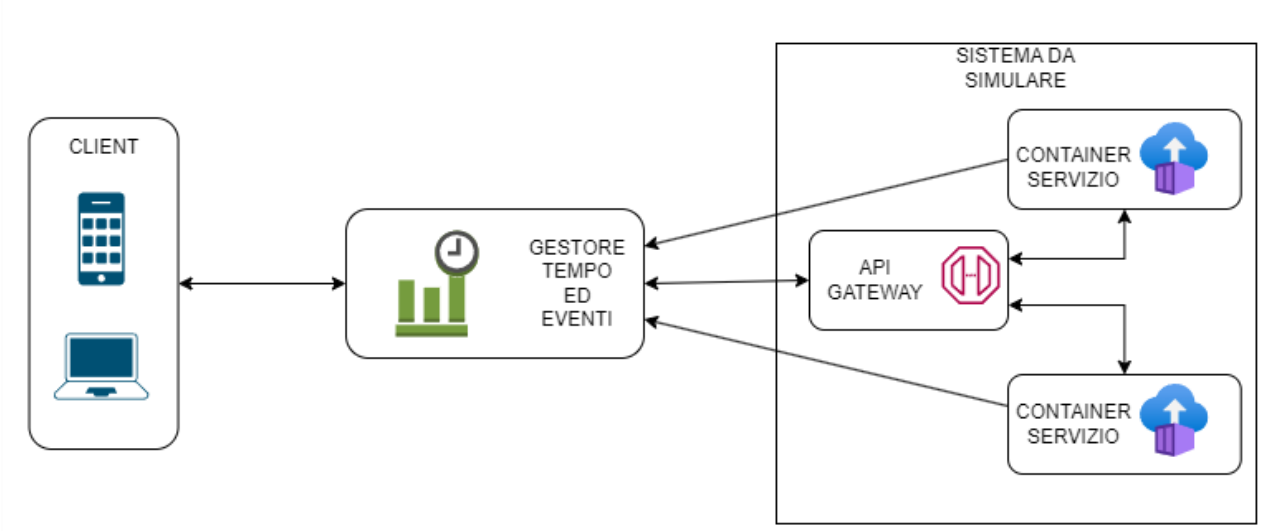

Figura 3.1: architettura per simulazione di sistema di reti di container con comunicazione indiretta

Come si può vedere dalla figura 3.1, il gestore del tempo e degli eventi gestisce l'intera simulazione, ogni pacchetto che viene scambiato tra sistema e client passa dal gestore, questo perché ogni pacchetto viene considerato come un evento e di conseguenza, nel caso della simulazione guidata dagli eventi, deve essere gestito in base al *tempo di realizzazione*. Come si può vedere si ha una comunicazione diretta anche tra i container e il gestore del tempo, questa serve per garantire la sincronizzazione di tutti i container. Nello schema della figura 3.1 la rete di container è gestita da un API Gataway; quindi, la comunicazione tra client e sistema è indiretta, in questo caso se si vuole è possibile integrare i metodi del gestore del tempo direttamente all'interno del codice dell'API. Nel caso di un sistema con comunicazione indiretta la situazione non cambia, il gestore si troverà sempre tra i client e il sistema, ma verrà messo in comunicazione con tutti gli end-point di ogni servizio.

3.3. Simulazione e gestione del tempo e degli eventi

Come già spiegato, la simulazione del tempo ad eventi discreti può essere di due tipologie, ad incremento di tempo prefissati o guidata dagli eventi. Considerando che uno degli obbiettivi del lavoro svolto è quello di permettere una simulazione dove il tempo di wallclock è minimo, sembra ovvia la scelta di utilizzare la simulazione del tempo guidata dagli eventi, ovvio se non si considera il problema della sincronizzazione.

3.3.1. Il problema della sincronizzazione

Si consideri il sistema di simulazione con i diversi container che si scambiano messaggi. Come già spiegato al punto 2.2.2, lo scambio di messaggi rappresenta un evento, ognuno con il proprio tempo di realizzazione, inoltre il messaggio ricevuto può portare alla creazione di un ulteriore evento.

Prendiamo come esempio due container C1 e C2, con C1 che invia un messaggio a C2.

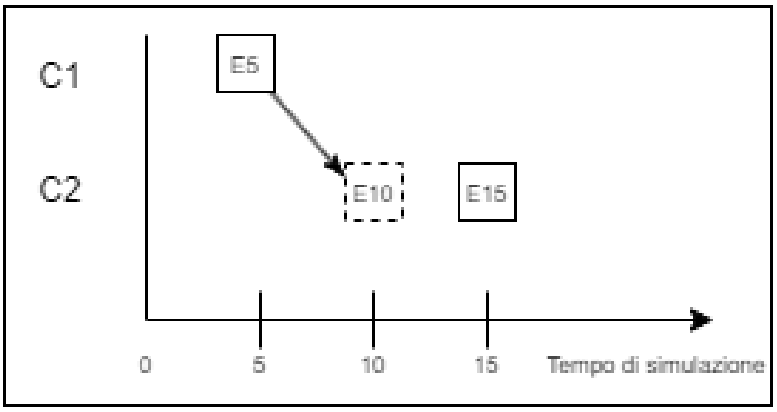

Figura 3.2: scambio messaggi tra container durante simulazione guisata dagli eventi

Come si può vedere dalla figura 3.2, un evento del container C1 al tempo di simulazione 5 porta la creazione all'evento con tempo di simulazione 10 nel container C2. Si può notare come dal grafico che il container C2 prima della generazione del nuovo evento E10 non ha alcun evento dal tempo di simulazione 0 a 15, quindi quando la simulazione del container C1 porta alla generazione dell'evento E10 il container C2 molto probabilmente si trova già al tempo di simulazione 15 e sta simulando l'evento E15. Nel caso l'evento E10 porti alla modifica di variabili di stato che vengono utilizzate da E15 avremo l'insorgenza di un errore di simulazione.4

Lo stesso discorso vale nel caso dell'utilizzo di memoria condivisa o di un database, in quel caso visto che i container sono non sincronizzati il risultato della simulazione risulta essere errato.

Ma quindi come fare per risolvere il problema della sincronizzazione ed a riuscire ad avere un tempo di wallclock ottimale?

3.3.2. Gestione del client

La soluzione pensata per risolvere il problema della sincronizzazione e allo stesso tempo puntare ad un tempo di wallclock basso è quella di gestire la simulazione del tempo in modo differenziato per i client e i container.

I client non condividono la memoria, e l'eventuale comunicazione fra di essi passa per il sistema, di conseguenza essi non devono essere sincronizzati se vengono gestiti in modo che i client si blocchino ogni volta che inviino una richiesta aspettando la risposta del sistema. Si consideri una richiesta come un evento in cui viene messo in coda il messaggio

[4] Richard M. Fujimoto, Parallel and Distributed Simulation Systems, John Wiley & Sons, 2000, pp. 53

per poi esser consegnato al container del servizio richiesto, come spiegato in precedenza al punto 2.2.2., nella simulazione guidata dagli eventi ad ogni evento viene assegnato il tempo di realizzazione, cioè il tempo in cui l'evento si realizza durante la simulazione. Se viene gestito lo scambio di richieste tra client e sistema in modo che la simulazione non vada avanti finché tutti i client non abbiano inviato almeno una richiesta, si riesce a gestire una corretta simulazione dei client scegliendo di volta in volta dalla lista delle richieste l'evento con il più basso tempo di realizzazione. In questo modo viene gestito in modo corretto lo scambio di richieste visto che anche se una richiesta cambia lo stato del sistema. in modo da cambiare delle variabili utilizzate da altre richieste, queste hanno un tempo di realizzazione maggiore rispetta alla richiesta scelta e di conseguenza la corretta modellazione del tempo e dei cambi di stato del sistema viene mantenuta.

Per gestire i client, di conseguenza, avremo bisogno che ogni client abbia la propria coda a cui inviare le richieste del sistema.

```
1: procedure GET_NEXT_REQUEST(list of client queues)
2:     waint until each queue contains at leat one request
3:     setRequests ← remove smallest time stamped requests M from
   queues
4:     clock ← time stamp of M
5:     return setRequests
```

Figura 3.3: funzione che restituisce le prossime richiesta da elaborare

Nella figura 3.3 si può vedere, nella riga 3, che quando si va a scegliere la richiesta con il tempo di realizzazione minimo in realtà si vanno a prendere un insieme di richieste e non una sola richiesta, questo perché potrebbero esserci più di una richiesta con lo stesso valore di tempo minimo, queste richieste devono essere gestite insieme perché le richieste potrebbero poter essere elaborate contemporaneamente nel caso si riferiscano a diversi servizi o a cui si ha a disposizione più istanze. Una volta avuto l'insieme delle richieste, che non può essere vuoto, bisogna gestire le richieste in modo da assegnarle ad ogni container in base al servizio richiesto, nel caso il numero di richieste per un determinato servizio supera il numero di container relativi al servizio, le richieste in eccedenza vengono rimesse nella coda dei clienti con il tempo di realizzazione aumentato in base al tempo di elaborazione dei container delle richieste. In questo modo si riesce a simulare il tempo di attesa delle richieste nel caso di container insufficienti per gestire i messaggi in un determinato momento.

```
1:  procedure ASSIGN_REQUET_TO_CONTAINER(list of requests, list of instances container)
2:      for request in list of request do
3:          S ← get_requested_service(request)
4:          if is available a container for service S then
5:              assign request to container
6:          else
7:              request.update_time(workTime)
8:              push request to client queue
9:          end if
10:         Return
```

Figura 3.4: funzione che assegna le richieste in base al numero di container disponibili per ogni servizio

Infine, per completare il discorso sulla gestione del client manca solamente la descrizione su come viene gestita la simulazione guidata agli eventi per i client.

```
clockTime ← 0.0
S ← connect_to_simulation_tool()
compute_initial_state(x)
T ← compute_sleep_time(x)
clockTime ← clockTime + T
while (simulation is not over) do
    responce ← send_message(S, request, clockTime)
    message ← get_message(responce)
    newTime ← get_clock_time(responce)
    clockTime ← newTime
    compute_state(x, clockTime, message)
    T ← compute_sleep_time(x)
    clockTime ← clockTime + T
```

Figura 3.4: modello simulazione guidata dagli eventi del client

Una volta connesso al gestore del tempo e degli eventi, necessario per la creazione della coda per il client e per la comunicazione con il sistema. Il client elabora lo stato iniziale per poi iniziare il loop per la simulazione dove ad ogni ciclo invia le richieste al sistema, attende la risposta ed aggiorna il clock in base al tempo di riportato nella risposta, che altro non è che il tempo che il server ci ha messo per gestire il messaggio a cui va sommato il tempo di trasmissione. La parte finale del ciclo prevede di aggiornare lo stato del sistema in base alle informazioni ricevute dal server.

3.3.3. Gestione dei container

Per quanto riguarda la simulazione dei container, a causa del problema della sincronizzazione e della memoria condivisa, è necessario utilizzare diversi metodi per la simulazione rispetto ai client. Di norma quando si parla di container si parla di un componente in attesa che arrivi una richiesta per poi gestirla, ecco la simulazione utilizza questa attesa per ottenere la sincronizzazione. Praticamente possiamo considera la simulazione guidata dagli eventi dove gli unici due tipi di eventi presi in considerazione per la simulazione sono l'arrivo di una richiesta al container e il completamento della gestione della richiesta. Considerando che ogni volta che vengono scelte le richieste con il valore minimo di time stamp (naturalmente ogni volta che viene formato l'insieme delle richieste da assegnare ai container esse hanno tutte lo stesso valore minimo di time stamp), si può costruire la simulazione dei container assegnando il time stamp della richiesta che lo ha fatto sbloccare al clock del container, in questo modo tutti i container che riceveranno una richiesta che andrebbe eseguita contemporaneamente avranno il valore del clock uguale. Qui sorge un problema, i container a cui non arrivano le richieste come fanno a essere sincronizzati. Si potrebbe pensare che sia inutile analizzare questo problema visto che il container che non riceve alcuna richiesta non dovrebbe avvenire alcuna richiesta, ma mettiamo caso si debba simulare un servizio che in base ai cambi di sistema si attivi e svolga dell'operazione. In questo caso il servizio anche se originalmente non prevedeva l'attesa di una richiesta, il servizio dovrà comunque prevedere un'attesa per essere sbloccato, questo perché è necessaria la sincronizzazione. Ma come sbloccare un servizio che non riceve richieste dal client, molto semplicemente ogni volta che vengono assegnate le richieste ai container ad ogni container che non è stata inviata alcuna richiesta verrà inviata una richiesta speciale "nulla" con solo il time stamp, in questo modo tutti i container, compresi quelli a cui non sarebbe arrivata una richiesta, riceveranno ogni volta contemporaneamente una richiesta con lo stesso time stamp, in questo modo tutti i container saranno sincronizzati.

```
1:  clockTime ← 0.0
2:  while (simulation is not over) do
3:      request ← wait_request()
4:      message ← get_message(requeste)
5:      clockTime ← get_clock_time(request)
6:      if message ≠ NULL then
7:          do work
8:      else
9:          pass
10:     end if
11:     clockTime ← clockTime + workTime
12:     send signal that work is completed to simulation handler
```

Figura 3.5: simulazione per i container

Come si può vedere dalla riga 11 della figura 3.5 il clock del container viene aumentato dalla variabile *workTime*, questa variabile rappresenta il tempo di servizio (tempo per l'esecuzione da parte del container di tutte le istruzione) del container. Per permettere una corretta simulazione e sincronizzazione il workTime deve essere uguale per tutti i container, è consigliato impostare il valore del workTime pari al tempo impiegato dal servizio più lento per essere svolto. Alla riga 12 invece vediamo l'invio del segnale al gestore del tempo e degli eventi che il lavore e finito. Questo avviene perché una volta che il gestore invia tutte le richieste esso devo bloccarsi finchè tutti i container non abbiano finito visto che potrebbe accadere che la simulazione di un container sia più veloce di altri, senza questo ulteriore meccanismo di sincronizzazione, il container più veloce si ritroverebbe con un valore del clock maggiore rispetto ai container più lenti.

```
1: procedure SYNC_CONTAINERS(number of instances of container)
2:     B ← 0
3:     while B <  number of istances  do
4:         wait_signal_from_container()
5:         B ← B + 1
        return
```

Figura 3.6: algoritmo sincronizazzione container del gestore del tempo e degli eventi

3.3.4. Gestione del tempo e degli eventi

Una volta spiegate tutte le procedure necessarie per la gestione dei container viene mostrato l'algoritmo principale per gestire il tempo e gli eventi della simulazione.

```
1:  N ← number of container istances for services
2:  C ← clock time
3:  T ← max simulation time
4:  Q ← queue client
5:  wait_clients_registration(Q)
6:  while  C ≤ T  do
7:      setRequests ← get_next_request(Q)
8:      assign_request_to_container(setRequests, N)
9:      send null message to free container
10:     sync_containers(N)
11:     C ← C + work time
12:     if event is occured then
13:         pause simulation and save state
14:         wait until unlock signal
15:     end if
```

Figura 3.7: algoritmo principale per la gestione del tempo

L’algoritmo per la gestione del tempo una volta che tutti i client si sono registrati alla simulazione, può partire per il ciclo principale che utilizza le procedure spiegate precedentemente. I client vengono registrati inizialmente, perché è possibile scegliere che un client inizi la simulazione ad un valore maggiore del tempo di simulazione iniziale, per fare ciò basta che la prima richiesta del client abbia un tempo di simulazione pari a quello a cui si vuole far iniziare la simulazione per il client.

Nella riga 12 viene mostrato che è possibile bloccare la simulazione se accade un evento precedentemente definito, questa opzione è molto utile nel caso si voglia analizzare lo stato del sistema quando avviene un particolare errore.

Capitolo 4

Risultati sperimentali

4.1. Obbiettivi

In questa sezione si pone come obbiettivo principale quello di mostrare le potenzialità dell'utilizzo dei metodi ideati per la simulazione di reti di container. Viene preso come sistema da simulare il Centro Di Controllo Droni, già introdotto durante la relazione, e viene simulato mostrando che la simulazione può venir usata per:

- Simulazione del funzionamento del sistema, mostrando i diversi scenari operativi del sistema.
- Simulare il comportamento del sistema per una settima nel tempo fisico in pochi minuti.
- Analisi delle prestazioni, il tool permette di valutare le prestazioni del sistema, come ad esempio il tempo di risposta, la capacità di scalare Ecc.
- Modello di ottimizzazione che consente di trovare la soluzione migliore, ovvero quella che minimizza o massimizza una funzione obiettivo, soggetta a un insieme di vincoli.

4.2. Hardware e software utilizzato

Per una migliore valutazione dei risultati della simulazione viene riportato l'hardware della macchina e il software utilizzati per implementare la simulazione:

Hardware:

Processore: AMD Ryzen 5 5600G con 6 cores e 12 threads e una velocità di 3.9GHz.
Memoria: 16 GB DDR4 con una velocità di 3000MHz

Software:

Sistema operativo: Ubuntu 20.04 LTS 64 bit
Container: per ogni istanza dei servizi del sistema è stata utilizzata un Docker container

Si ricorda che anche cambiano il software e l'hardware i risultati della simulazione non cambierebbero, l'unica differenza sarebbe il tempo di wallclock che dipende dalla velocità del sistema.

4.3. Caso Studio

Come caso studio utilizzeremo il Centro Di Controllo droni, usato già come esempio durante la tesi per spiegare diversi concetti.

Un drone autonomo è un sistema aereo a pilotaggio remoto, che ha la capacità id opera senza l'intervento del pilota nella gestione del volo. I droni vengono utilizzati per diversi scopi, tra cui la mappatura e la sorveglianza del territorio, la prevenzione degli incendi e le

ricerche di persone disperse, usi militari Ecc.
Lo scopo principale del sistema sviluppato è di fornire la possibilità del controllo del territorio diviso in punti d'interesse attraverso l'utilizzo dei droni. Il compito principale del sistema è quello di assegnare ai droni l'area da sorvegliare in modo da rispettare i requisiti del sistema.

4.3.1. Sistema Controllo Droni – Specifica dei requisiti

In questa sezione vengono specificati i requisiti funzionali e non funzionali, cioè le specifiche delle funzionalità e dei servizi che il sistema deve fornire per soddisfare le esigenze richieste in base alla commessa. Non si andrà parecchio nello specifico visto che il sistema viene progettato unicamente per far vedere le potenzialità dei tools sviluppati per la simulazione.

Requisiti Funzionali

1.1 Il sistema deve permettere di salvare la mappa di un territorio diviso in punti di interesse e punti di ricarica del drone.
1.2 Il sistema deve gestire i droni assegnando il punto di interesse o il punto di ricarica in base allo stato di sorveglianza delle aree e dalla percentuale di batteria residua del drone.
1.3 Il sistema deve gestire un insieme di droni assegnato ad ogni territorio in modo che ogni punto di interesse venga sorvegliato per 5 minuti continui, ogni 30 minuti.
1.4 Il sistema deve garantire che ogni drone non si scarichi.
1.5 Il sistema ogni volta che assegna un'area ad un drone deve fornirgli il miglior percorso per raggiungere l'area designata.
1.6 Il sistema deve tener salvato per ogni punto di interesse lo stato in cui si trova (libera, sorvegliata, assegnata).
1.7 Il sistema deve tener salvato la posizione e lo stato del drone.

Requisiti non Funzionali

2.1 Il sistema deve essere ottimizzato in modo da mantenere i tempi di risposta, in media, inferiori a 5 secondi.
2.2 Il sistema deve essere in grado di gestire un numero indefinito di droni in modo che non vengano sprecate inutili risorse. Di conseguenza deve essere implementato in modo da garantire una scalabilità flessibile.

4.3.2 Sistema Controllo Droni – Descrizione Drone

Esistono diverse tipologie di drone, con capacità e proprietà diverse in base all'esigenze richieste. Per il caso d'uso in esame si è deciso di usare un drone autonomo, cioè in grado di svolgere azioni in modo autonomo una volta assegnate dal Sistema Controllo Droni.
Il comportamento del drone è quello come prima cosa di registrarsi alla rete di droni per poi rimanere in attesa dei comandi da parte del Centro di Controllo Droni, per poi successivamente eseguire i comandi e una volta completati rimettersi in attesa, creando così un loop infinito. I comandi che può ricevere il drone sono di due tipi di modalità: sorveglianza o ricarica, questo varia in base alla percentuale di batteria che ha il drone,

insieme alla modalità di comando il drone riceve il cammino dell'aree da percorrere per arrivare all'area designata per poi eseguire il comando assegnato.

Per una migliore comprensione si descrive il comportamento del drone e dei suoi stati attraverso un diagramma di macchina a stati.

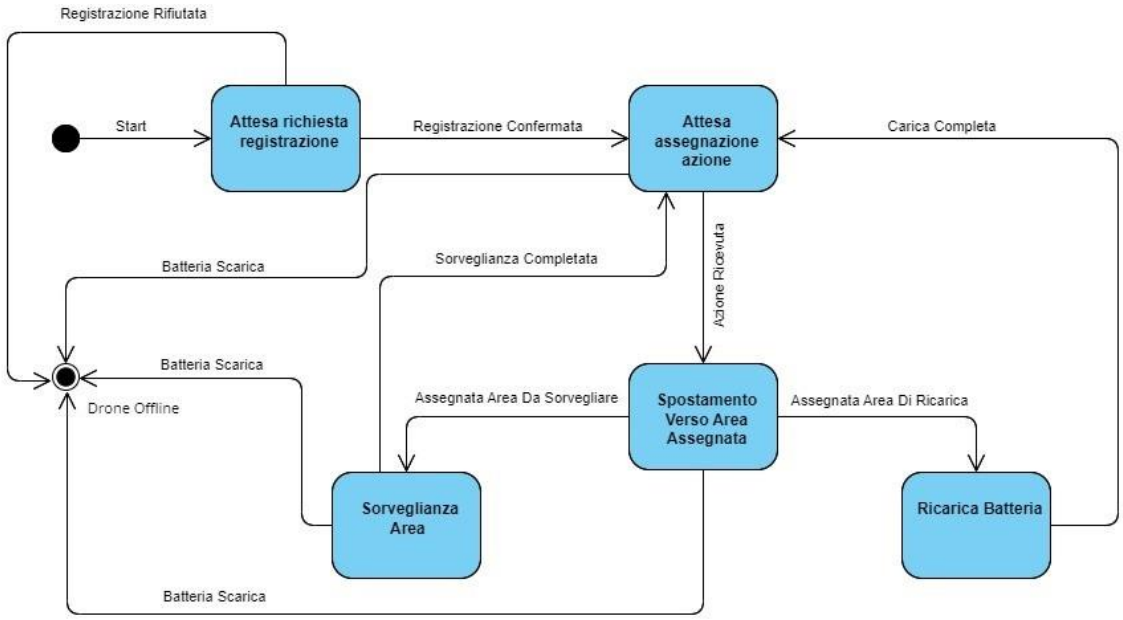

Figura 5.1: diagramma macchina a stati drone

La velocità del drone è stata fissata a 20 km/h, nella tabella sottostante sono riportati i valori di scarica e carica della batteria al minuto in base allo stato in cui si trova il drone.

STATO	PERCENTUALE BATTERIA
In attesa di comandi	- 1.0% al minuto
Sorveglianza area	- 4.0% al minuto
Movimento verso area designata	- 5.5% al minuto
In carica	+ 10.0% al minuto

Inoltre, si considerano le seguenti funzionalità:

- Salvare nella propria memoria la mappa del territorio diviso in area come grafo pesato diretto.
- Volo automatico una volta ricevuto il percorso verso l'area designata.
- Sorveglianza autonoma dell'area designata.
- Ricarica automatica una volta raggiuta l'area predisposta alla ricarica del drone.

4.4. Simulazione funzionamento del sistema

Per testare la correttezza del sistema vengono mostrati i diversi scenari operativi. I possibili scenari operativi del sistema modellato sono derivati dalla mappa data in input, dal numero di droni e dal numero di punti di ricarica. È stato anche implementato un monitor che verifica gli eventuali requisiti che non vengono rispettati.

Tutti i possibili scenari operativi sono stati testati con un'unica istanza dei container per servizio. Per tutti i test il sistema viene simulato per 86400 tic, considerando che è stato deciso che ogni tic equivale ad un secondo, si ha che per ogni test il sistema viene simulato per un tempo equivalente a 86400 secondi, pari a 24 ore.

Tutti i grafici che verranno mostrati, tranne il monitor, saranno limitati a 3000 tic in modo da rendere la lettura maggiormente intellegibile.

Scenario operativo 1: nessun requisito funzionale viene violato

Nel primo scenario operativo viene mostrato come il sistema si comporta nel caso in cui il numero dei drone e dei punti di ricarica sia adeguato a rispettare i requisiti funzionali. L'area di partenza di tutti i droni e differente, tranne per i droni con identificativo 0 e 1 che partiranno dalla stessa area.

Dati in input:

Numero Aree Mappa	30
Numero Droni	15
Numero Punti Di Ricarica	30
Tempo risposta servizi in secondi	0.5
Tempo di trasmissione in secondi	1.0
Tempo Di Simulazione	86400

Risultati simulazione

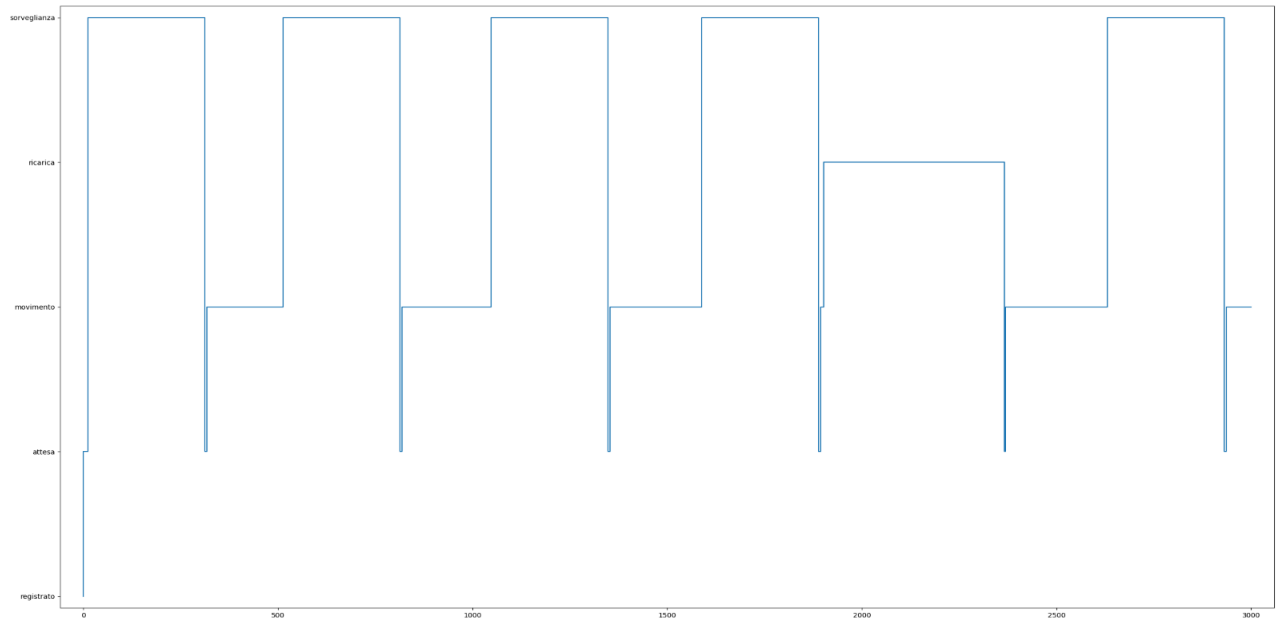

Figura 5.2: grafico stati del drone con identificativo 0

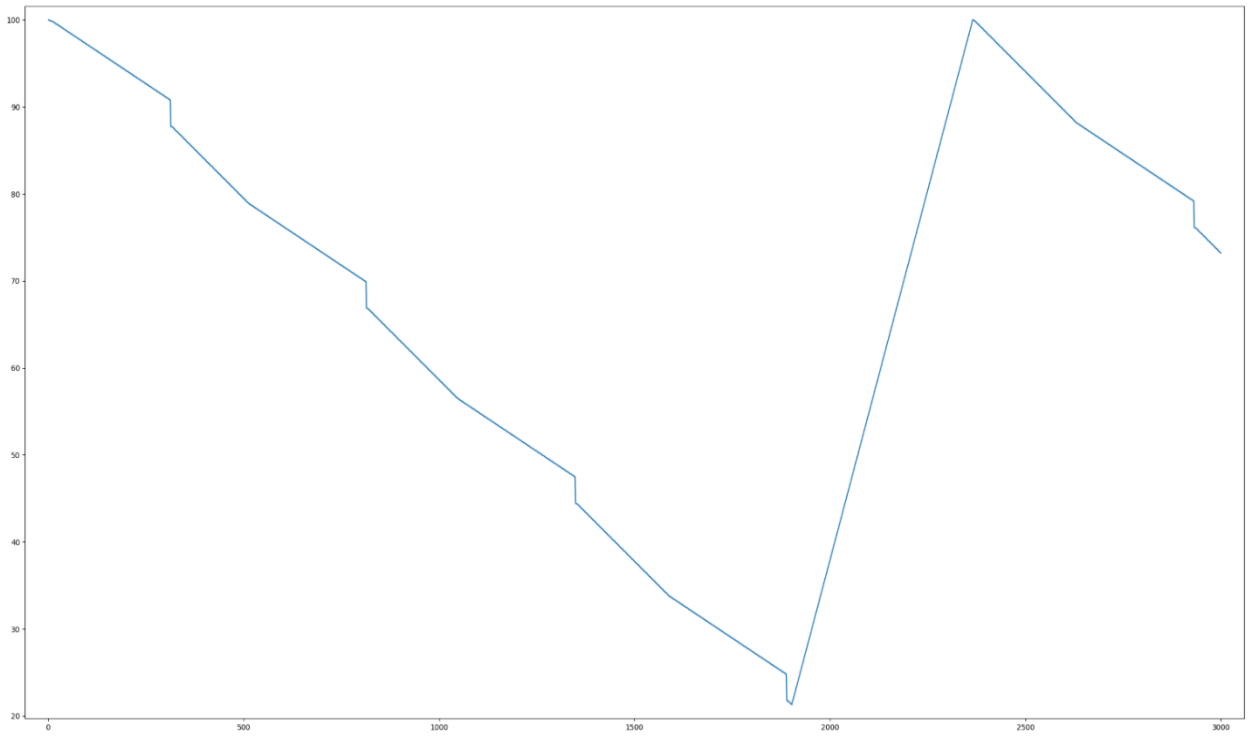

Figura 5.3: grafico stati del drone con identificativo 0

Come si può vedere dalla figura 5.2 il drone si comporta come descritto nel diagramma a stati al punto 5.3.3.1. Partendo dallo stato “registrato” passa successivamente allo stato “attesa”, e rimane in attesa finché il Centro Di Controllo Droni non gli invii l’area assegnata. Una volta ricevuta l’area il drone passa nello stato “in movimento” fino al raggiungimento dell’area designata dove inizia la sorveglianza passando nello stato “in sorveglianza”. Questo ciclo si ripete finché il drone non si mette in attesa con un livello di carica inferiore al 35%, in questo caso il Centro Di Controllo Droni assegna un punto di ricarica, una volta raggiunto il punto il drone rimane nell’area fino a carica completata, per poi riprendere il ciclo di sorveglianza.

Come si può vedere l’unico momento in cui il drone passa direttamente dallo stato “in attesa” a quello “in sorveglianza” è alla prima richiesta di assegnamento area, questo perché all’inizio della simulazione tutte le area hanno lo stesso valore di partenza; quindi, l’algoritmo si comporta correttamente assegnando al drone l’area più vicina, che altro non è che l’area in cui già si trova.

La figura 5.3 mostra come la batteria si comporta come modellato nella tabella al punto 5.3.3.3, dove la velocità di scarica dipende dallo stato del drone

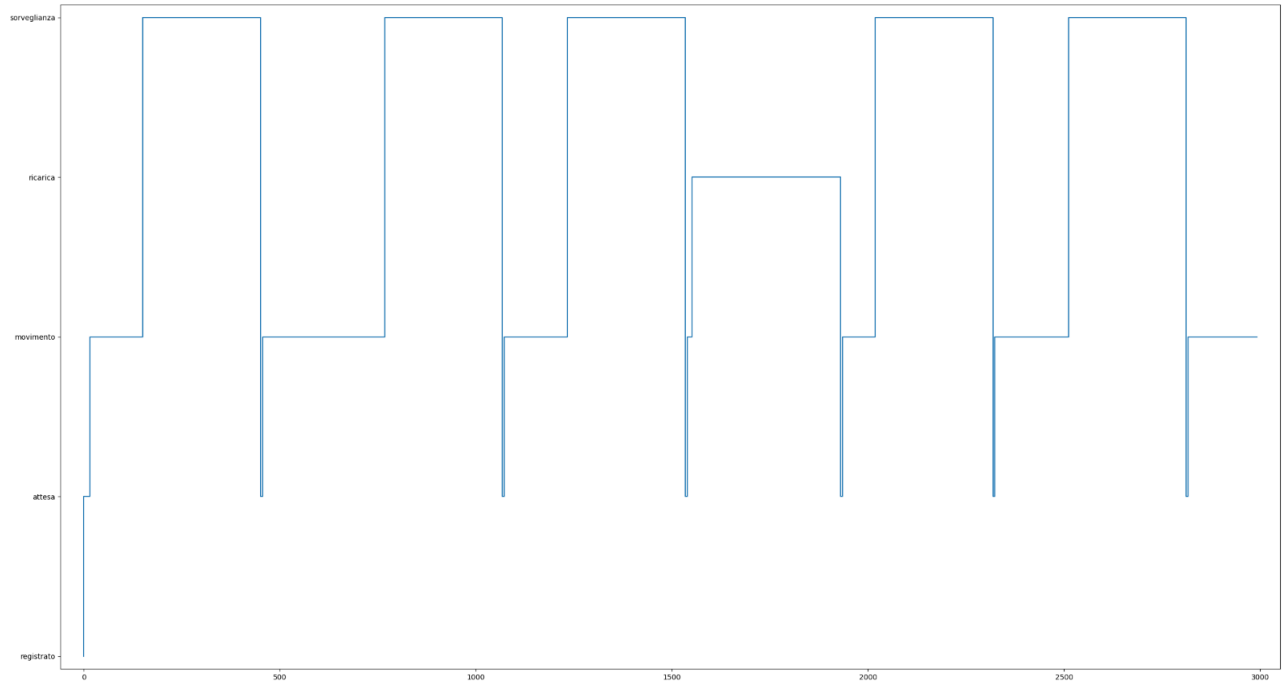

Figura 5.4: grafico stati del drone con identificativo 1

Come specificato in precedenza, gli unici droni che partono dalla stessa area (l'area con identificativo 0) sono il drone con identificativo 0 e 1, questo significa che uno dei due droni è l'unico che alla prima richiesta di assegnamento area non riceve l'area in cui già si trova. Come visto in precedenza il drone con identificativo 0 riceve come prima area da sorvegliare l'area in cui già si trova, di conseguenza il drone con identificativo 1 ne riceve un'altra. Questo si può vedere dal grafico del drone con identificativo 1, dopo l'attesa inziale, invece che passare direttamente allo stato "in sorveglianza" passa allo stato "in movimento" finché non raggiunge l'area designata.

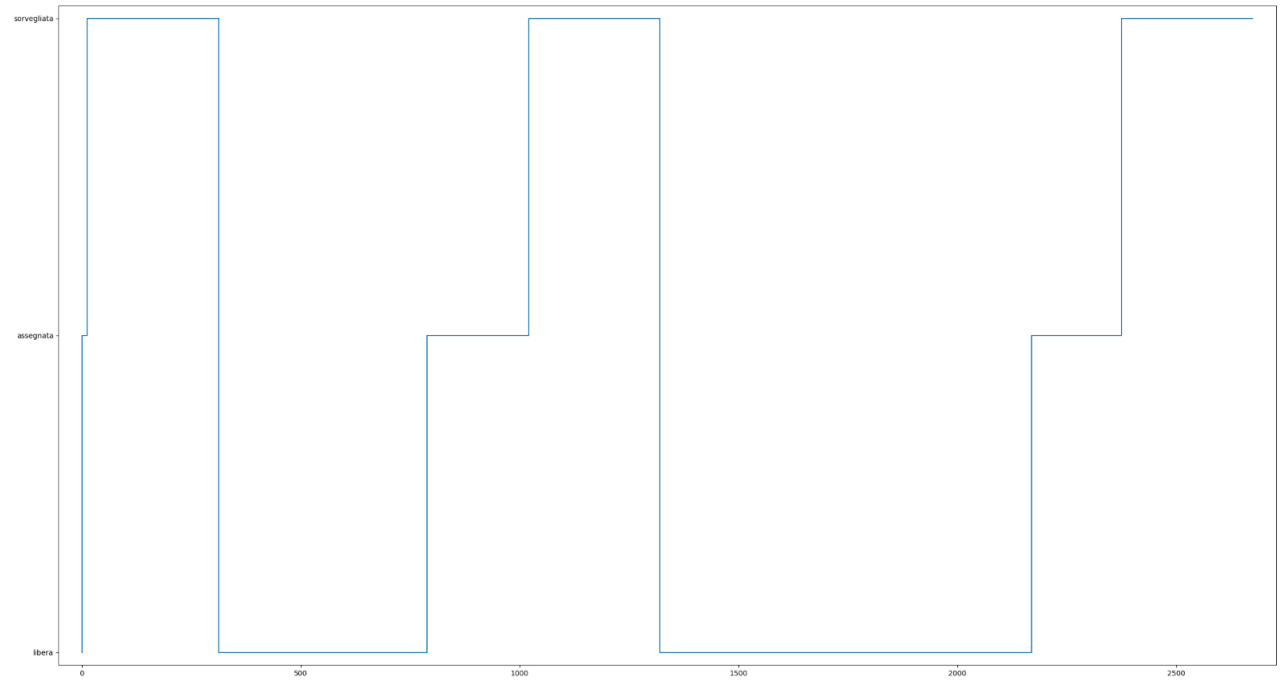

Figura 5.5: grafico stati area con identificativo 0

Come si può vedere dalla figura 5.5, l'area passa ciclicamente dallo stato libero, a quello "assegnato" quando l'area viene assegnata al drone, passo allo stato "sorvegliato" dopo il tempo necessario per il drone di arrivare all'area ed iniziare la sorveglianza. Si può vedere come sia minimo il tempo che passa dallo stato "assegnato" a quello "in sorveglianza" all'inizio del grafico, questo perché come detto in precedenza il sistema parte con il drone con identificativo 0 già nella posizione dell'area 0, di conseguenza il tempo che intercorre tra l'assegnazione e la sorveglianza è solo quello della trasmissione del comando da parte del Centro di Controllo Droni verso il drone.

Non essendo stati segnalato alcuna violazione dei requisiti, risulta inutile mostrare il grafico del monitor.

Scenario operativo 2: Numero punti di ricarica insufficienti

In questo scenario operativo si vuole mostrare il caso in cui il numero di punti di ricarica sia insufficiente, questo comporta alla violazione del vicolo che nessun drone deve scaricarsi completamente.

Dati in input:

Numero Aree Mappa	30
Numero Droni	15
Numero Punti Di Ricarica	7
Tempo risposta servizi in secondi	0.5
Tempo di trasmissione in secondi	1.0
Tempo Di Simulazione	86400

Risultati simulazione

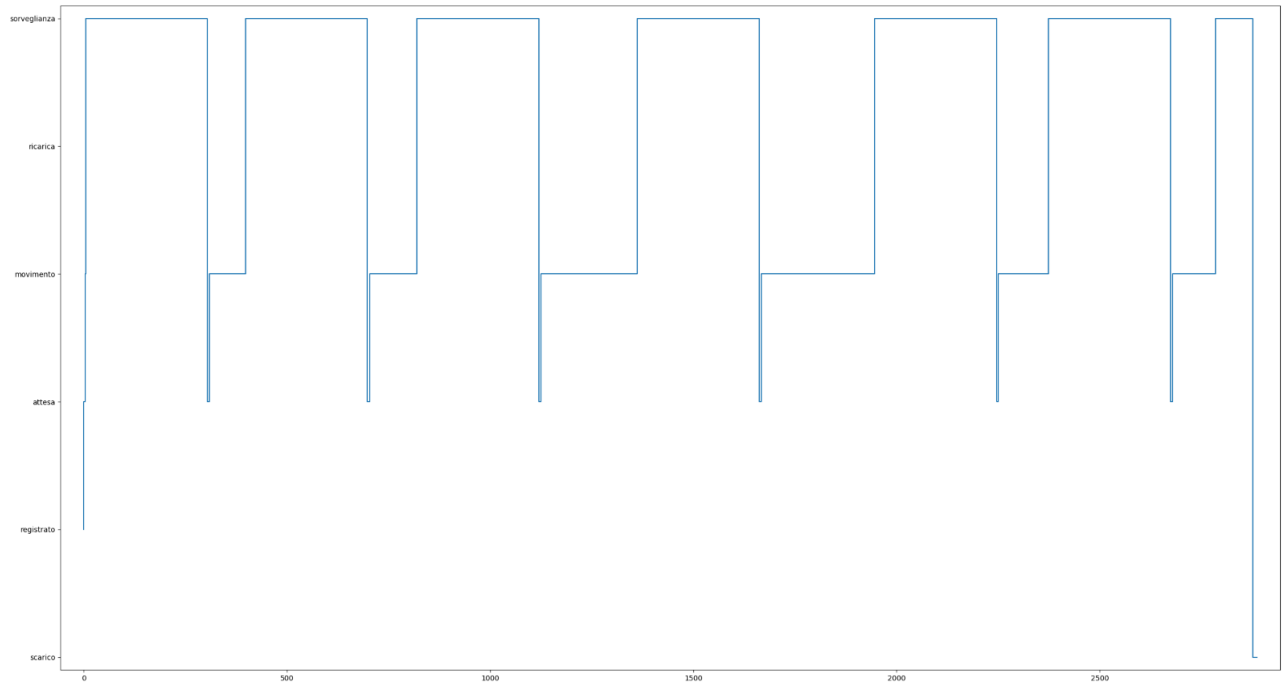

Figura 5.6: grafico stati del drone con identificativo 0

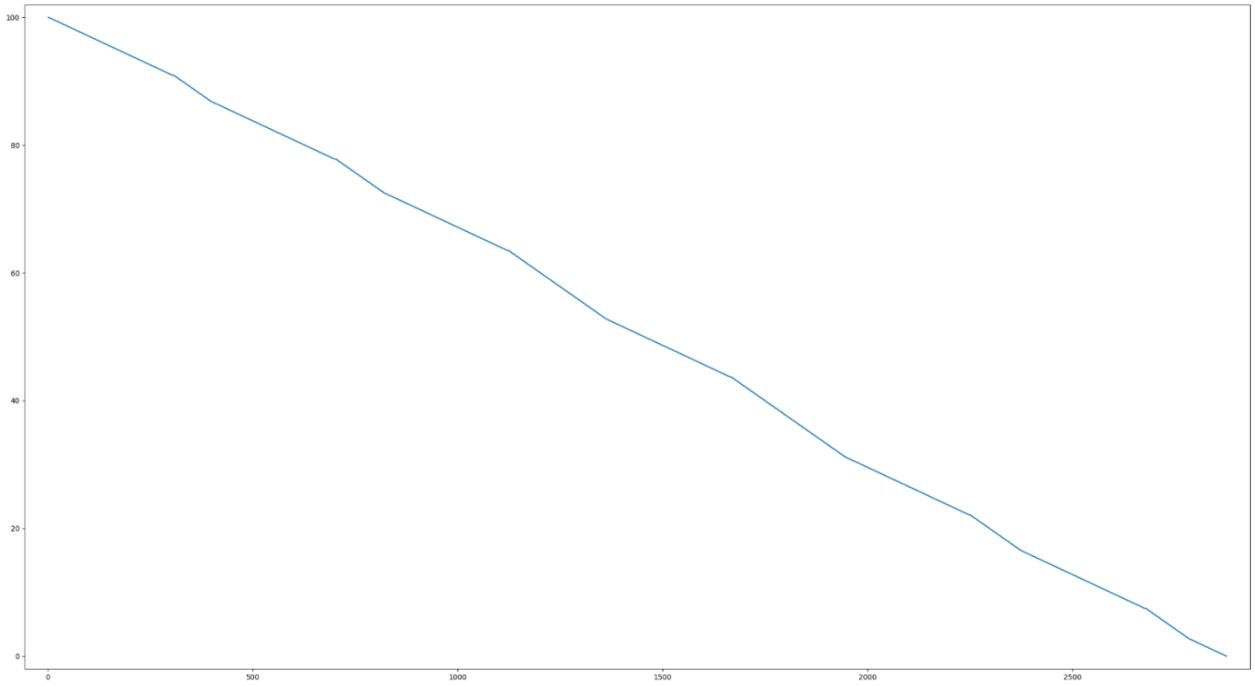

Figura 5.7: grafico andamento batteria drone con identificativo 0

Come si può vedere dalle figure 5.6 e 5.7 nonostante il drone nella sua ultima richiesta di assegnazione area, quando il suo stato è "in attesa", abbia una percentuale di batteria inferiore al 35% gli viene comunque assegnata un'area da sorvegliare, con conseguente completa scarica di batteria da parte del drone.
Questo, come previsto da scenario operativo, è ciò che accade quando il numero di punti di ricarica è insufficiente e tutti i punti di ricarica sono occupati durante la richiesta di assegnazione area da parte di un drone che ne avrebbe bisogno.

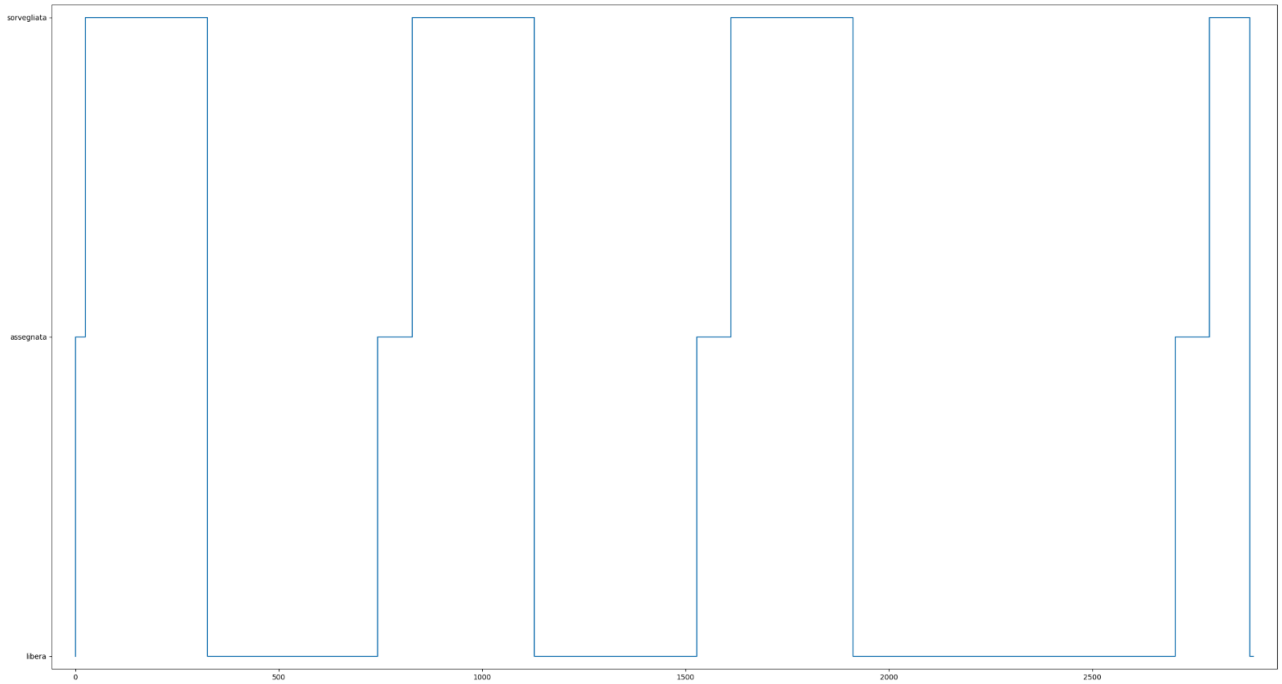

Figura 5.8: grafico stati area con identificativo 8

Nel grafico della figura 5.8 viene mostrato come viene gestita correttamente dal sistema e dai metodi per la simulazione la disconnessione dei client; infatti, l'area sorvegliata dal drone risulta essere correttamente reimpostata nello stato "libera" nel momento in cui il drone che sta sorvegliando si disconnette (drone figura 5.5). Come si può notare dal grafico il tempo trascorso nello ultimo stato "in sorveglianza" è minore rispetto ai precedenti.

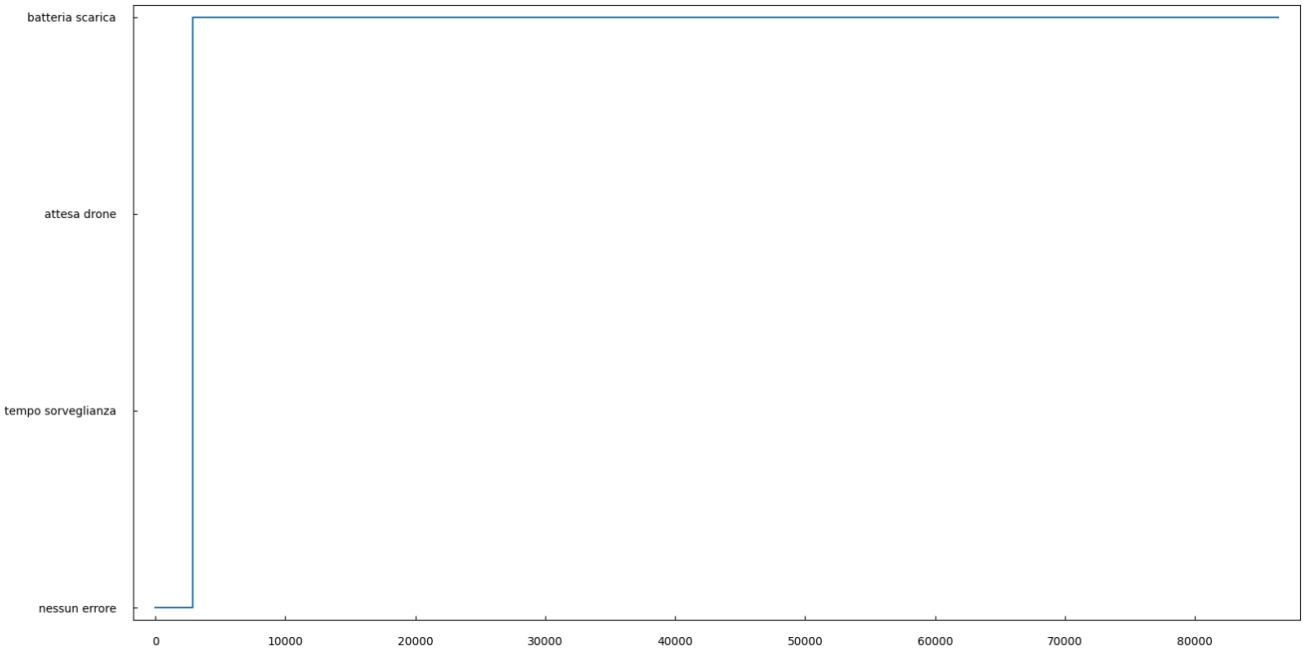

Figura 5.9: grafico monitor violazione batteria scarica

Nel grafico del monitor correttamente segnala la violazione del requisito che verifica che la batteria non si scarichi.

Scenario operativo 3: Numero droni insufficienti

In questo scenario operativo si mostra il caso in cui il numero di droni sia insufficiente per sorvegliare i punti di interesse secondo i requisiti. I punti di interesse vanno sorvegliati per 5 minuti (300 Tic) ogni mezzora (1800 Tic)

Dati in input:

Numero Aree Mappa	30
Numero Droni	5
Numero Punti Di Ricarica	7
Tempo risposta servizi in secondi	0.5
Tempo di trasmissione in secondi	1.0
Tempo Di Simulazione	86400

Risultati sperimentali:

Figura 5.10: grafico stati area

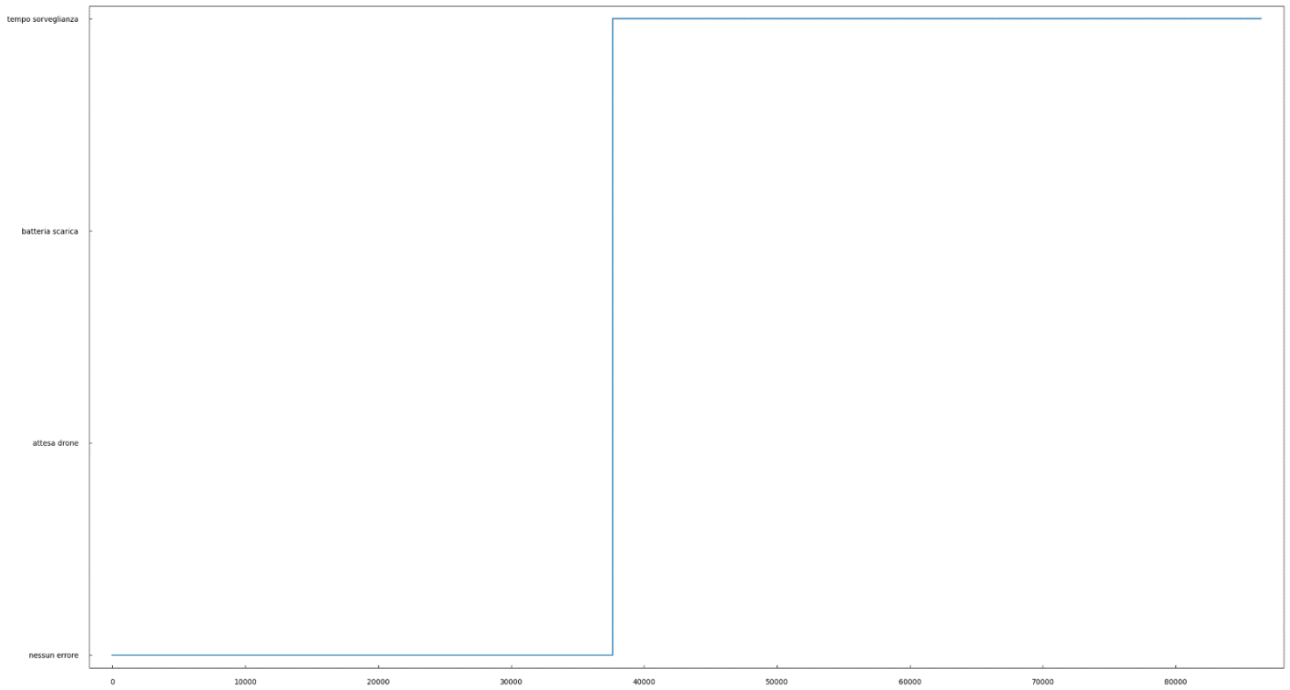

Figura 5.11: grafico monitor violazione sorveglianza punto di interesse

Come mostrato nella figura 5.10 il punto di interesse finisce di essere sorvegliato al Tic 35848 e di nuovo al Tic 37665. Secondo i requisiti un drone avrebbe dovuto iniziare la sorveglianza del punto entro il Tic 37648 di conseguenza il requisito di sorveglianza risulta essere violato, come riportato anche dal monitor.

4.5. Valutazione tempo di wallclock

Come già esplicitato nella relazione, uno degli obbiettivi principali dei metodi creati per la simulazione è stato quello di permettere un tempo di wallclock minimo. Ricordiamo che il tempo di wallclock è il tempo reale che viene impiegato dall'inizio della simulazione alla fine della simulazione. Ma come viene simulato il contato il tempo di wallclock? Nel nostro caso prima di far partire il test, viene salvata in una variabile l'ora attuale di inizio del test, una volta finito il test viene salvata l'ora attuale. Il wallclock time sarà dato dal valore della variabile con l'ora di fine del test a cui viene sottratto il valore della variabile di inizio test.

$startTime \leftarrow getTimeNow()$
startTest()
$finishTime \leftarrow getTimeNow()$
$wallclockTime \leftarrow finishTime - startTime$

Figura 5.12: Algoritmo calcolo tempo di wallclock

Ma cosa da cosa dipende il valore del tempo di wallclock? Sono molti i fattori, come la macchina su cui gira il simulatore, i dati in input visto che in base ad essi cambiano gli stati del sistema e quindi i calcoli necessari, lo stato della rete, il numero di client, il numero di TIC che verranno simulati e quindi il tempo fisico simulato. I due fattori principali si cui ci concentreremo saranno il numero di client e il tempo fisico di simulazione, questo perché tra tutti i fattori sono i due che portano ad una maggiore differenziazione tra i valori del tempo di wallclock, e inoltre sono i due maggiormente quantificabili.

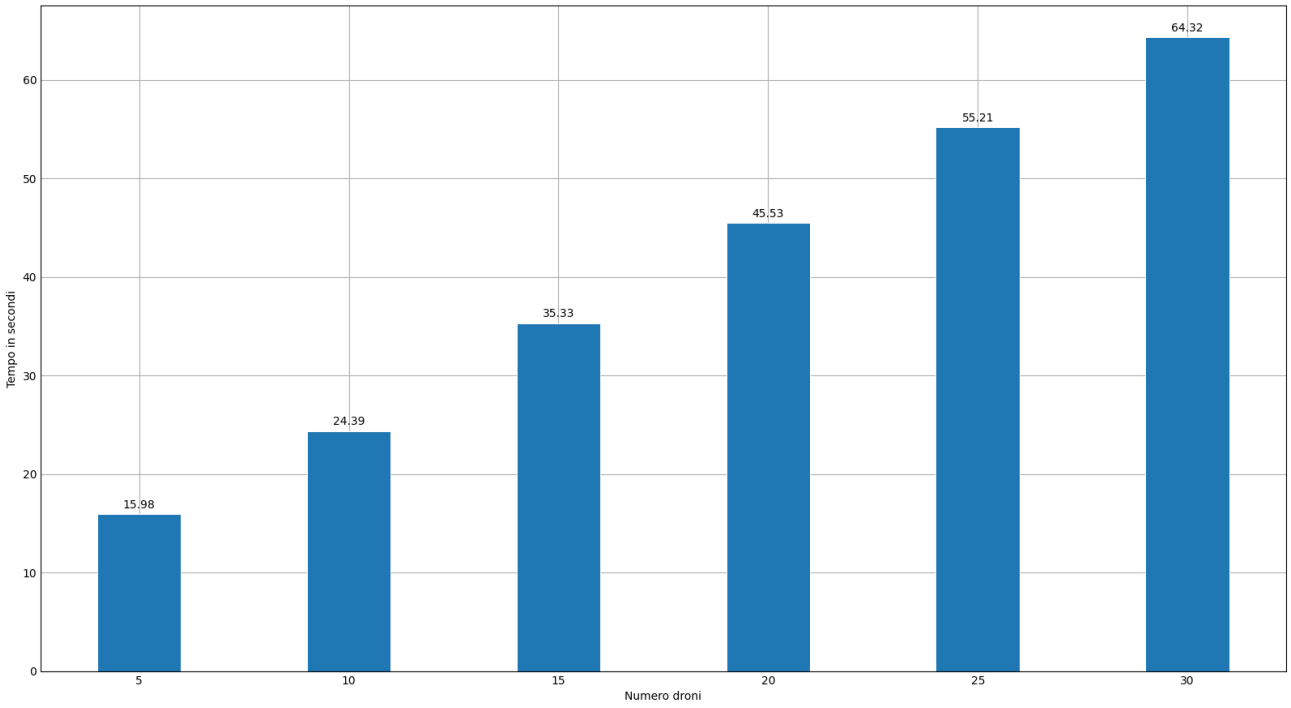

Figura 5.13: Grafico tempo di wallclock in secondi in funzione del numero di droni

Come si può vedere dal grafico 5.11 come ci si aspetta il tempo di wallclock cresce linearmente in funzione al numero di droni, e quindi client. Questo risultato lo si poteva immaginare visto che con l'aumentare il numero di client aumenta anche il numero di eventi e di conseguenza il gestore del tempo e degli eventi dovrà fare un numero maggiore di cicli durante la simulazione per gestire tutti i client.

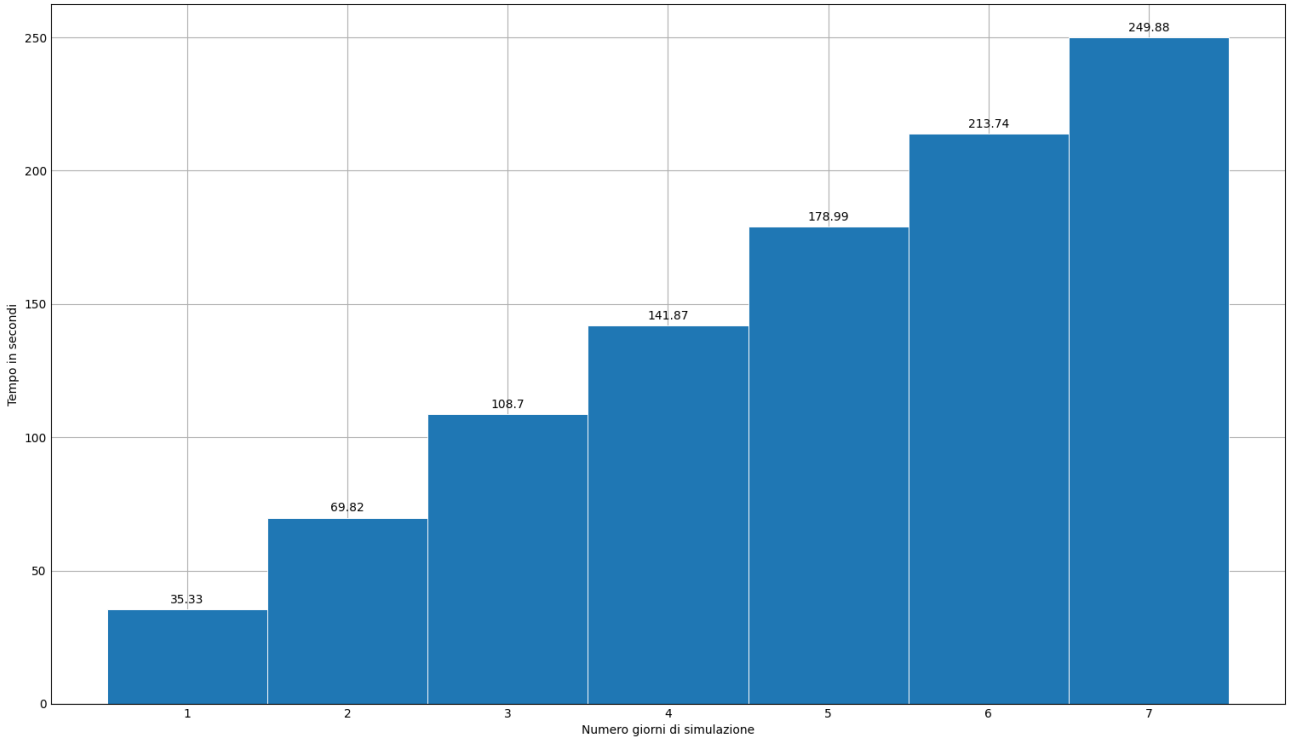

Figura 5.14: Grafico tempo di wallclock in secondi in funzione del numero di giorni da simulare

Così come il grafico 5.13 anche nel grafico 5.14 si ha una crescita lineare, anche qui il risultato è inaspettato, visto che il valore del tempo di wallclock per simulare 15 droni per 1 giorno è di circa 35 secondi, il tempo per simulare n giorni sarà dato da $n \cdot 35$, così come mostrato nel grafico dai risultati sperimentali.
Dai risultati sperimentali per la valutazione del tempo di wallclock, si può costatare che l'obbiettivo di simulare il lavoro del sistema di diversi giorni in pochi minuti è stato raggiunto.

4.6. Analisi delle prestazioni

Come scritto anche negli obbiettivi grazie ai metodi implementati è possibile analizzare la prestazione del sistema in modo da permettere l'analisi del comportamento del sistema in base ai valori assegnati al tempo di servizio dei container, il tempo di trasmissione e il numero di istanze dei container per servizio. Vengono mostrati i grafici della simulazione in modo da mostrare come il tempo di risposta vari in base al numero di istanza del container, questi grafici sono molto importati perché mostrano come i metodi implementati possano essere utilizzati per verificare la scalabilità del sistema simulato.

Dati in input:

Numero Aree Mappa	30
Numero Droni	30
Numero Punti Di Ricarica	15
Tempo risposta servizi in secondi	2
Tempo di trasmissione in secondi	1.0
Tempo Di Simulazione	86400

Nel caso dei risultati dei test che si vedranno, quando si fa riferimento al numero di istanze ci si riferisce al numero di container del servizio "assegna punto di interesse", visto che è servizio che incide maggiormente sul tempo d'attesa di risposta in cui rimane il drone.

Risultati sperimentali:

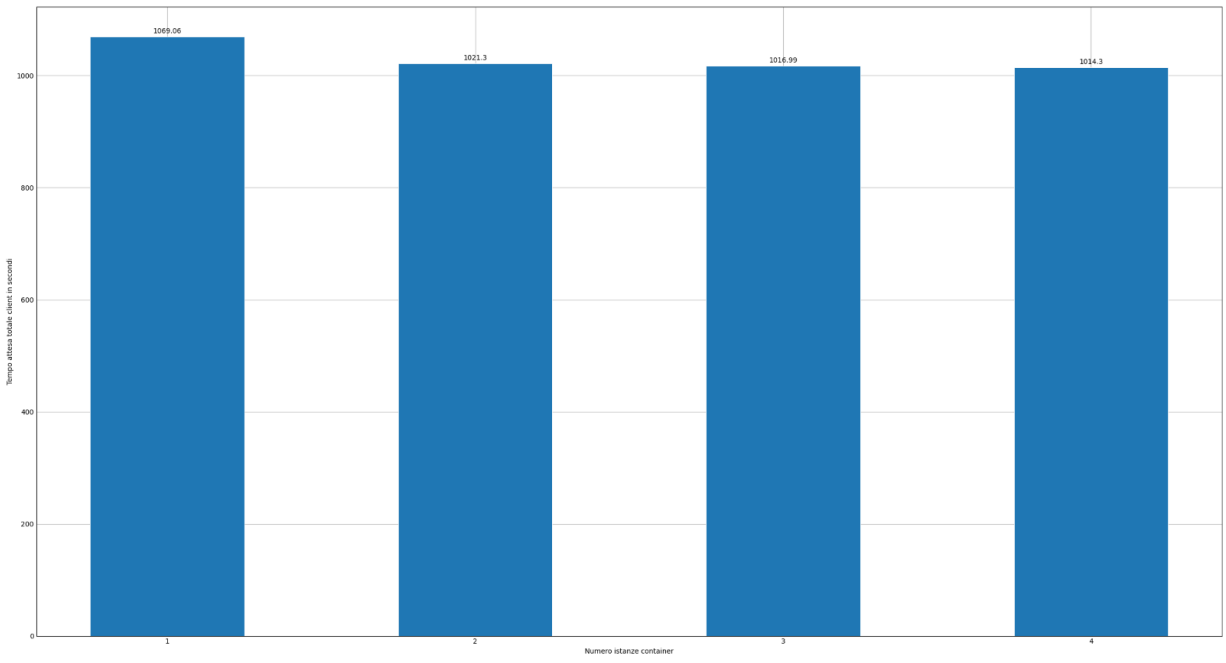

Figura 5.14: Grafico tempo totale di un drone in cui rimane in attesa di risposta durante la simulazione

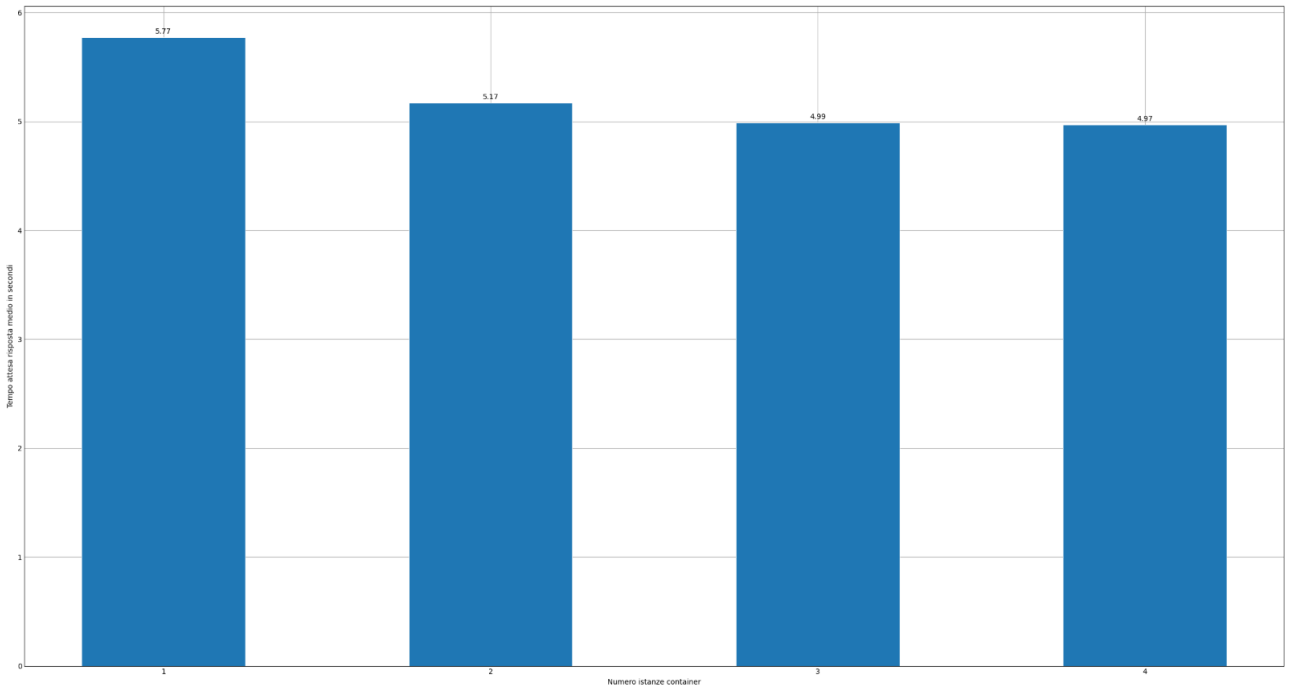

Figura 5.15: Grafico tempo in cui tempo totale del drone che rimane in attesa di risposta in funzione del numero di istanze del servizio che assegna l'area.

Dalle figure 5.14 e 5.15 si può vedere che con l'aumentare del numero di istanze, i droni rimangano in attesa per un periodo di tempo inferiore. Questo risulta essere ovvio, ma la parte interessante è che grazie al test non solo possiamo analizzare il tempo di risposta che diminuisce, ma anche il numero massimo di istanze che devono essere allocate per una diminuzione significativa del tempo di risposta. Nella figura 5.14 la differenza tra 3 e 4 istanze è minima, solo 2 secondi di attesa in meno per drone; quindi, nel caso si abbiano solo 30 droni da gestire è sufficiente un'architettura che riesca a far girare 3 istanze del servizio "assegna punto di interesse". Inoltre, si può notare come con 3 istanze il tempo di risposta medio vada al di sotto dei 5 secondi, così da non violare il requisito non funzionale 2.1.

Capitolo 5

Conclusioni

Nella relazione svolta viene descritto il lavoro svolto nell'ideazione degli algoritmi utili alla creazione di metodi generali da applicare nel caso si voglia simulare sistemi di reti di container tramite la simulazione ad eventi discreti.
Inizialmente si sono presentati i diversi motivi per cui l'utilizzo di sistemi di reti di container sia sempre più in crescita grazie ai numerosi vantaggi, tra cui la scalabilità, la migliore tolleranza ai guasti, una maggiore sicurezza e portabilità del sistema. Inoltre, si è spiegato l'importanza della simulazione dei sistemi per permetterne diverse possibilità come la validazione e la verifica del sistema, analizzare il comportamento in base ai diversi input del sistema modellato e valutarne le prestazioni.

I metodi presentati affrontano i principali problemi della simulazione ad eventi discreti, trovandone una soluzione che permettono non solo di simulare il sistema correttamente, ma anche di utilizzare nella simulazione il sistema fisico senza dover andare a creare un modello, ma applicando i metodi direttamente al sistema reale. In questo modo, non solo si evita la costruzione di un modello del sistema che oltre a richiedere molto tempo portando al rischio di errori di modellazione, ma ci si assicura che il codice scritto sia stato corretto e funzioni come progettato grazie all'utilizzo del sistema fisico finale.

Nella parte finale della relazione si sono mostrati il lavoro svolto all'opera su un sistema reale, mostrando come sia possibile utilizzare i metodi per mostrare e analizzare i diversi scenari operativi, analizzare le prestazioni in base al tempo di servizio dei container e al numero di istanza assegnate ai servizi, il tutto mostrando che è possibile simulare il comportamento del sistema di una settimana in pochi minuti di tempo di wallclock.

5.1 Sviluppi futuri

Un importante miglioramento che potrebbe essere fatto per raffinare la simulazione sarebbe quello di permetter un tempo di servizio diversificato per i diversi servizi, riuscendo a mantenere la sincronizzazione e la simulazione del tempo guidata dagli eventi. Anche se sembra semplice questo è un problema molto complesso da risolvere, perché per mantenere la gestione del tempo in modo che se un container impiega meno tempo gli sia possibile assegnare diverse richieste mentre allo stesso tempo ad un altro container gli si assegnino meno richieste, il tutto riuscendo a mantenere la sincronizzazione, risulta essere abbastanza ostica. Una possibile soluzione potrebbe essere quella di utilizzare diverse barriere all'interno del codice, dove ogni barriera rappresenta il passare un istante di tempo x. Tra una barriera e l'altra all'interno del codice dei container passere l'istante del tempo x, tutte le barriere di tutti i container dovrebbero rappresentare lo stesso istante di tempo e il numero di barriere per container rappresenterebbe il tempo di servizio totale del container.

Bibliografia

[1] Richard M. Fujimoto, Parallel and Distributed Simulation Systems, John Wiley & Sons, 2000

[2] M. Roma, *Sistemi di Servizio e Simulazione*, *http://www.diag.uniroma1.it/~roma/didattica/SSS19-20/parte2.pdf*

[3] Microsoft Learn, h*ttps://learn.microsoft.com/it-it/virtualization/windowscontainers/about/*