From Cloud to Edge: Seamless Software Migration at the Era of the Web of Things

Michele Beccari 856608

2023

1 Riassunto

Lo standard Web of Things (*WoT*) promosso di recente dal W3C costituisce un approccio promettente per progettare sistemi IoT interoperabili in grado di gestire l'etereogeneità delle piattaforme software e dei dispositivi.
L'architettura *WoT* prevede scenari di IoT caratterizzati da una moltitudine di Web Things (*WTs*) che comunicano secondo delle interfacce software ben definite.
Allo stesso tempo presume un'allocazione statica delle *WTs* agli host e non è in grado di gestire la dinamicità intrinseca degli ambienti IoT per quanto riguarda le variazioni del carico di rete e computazionale.
In queto paper vogliamo estendere il paradigma *WoT* al deployment nel continuo cloud-edge. In questo modo potremmo supportare un'orchestrazione dinamica e la mobilità delle *WTs* su tutte le risorse di calcolo disponibili.
A differenza degli approcci Mobile Edge Computing (*MEC*) allo stato dell'arte vogliamo sfruttare lo standard *WoT* e in particolare la sua capacità di standardizzare le interfacce software delle *WT* per propporre un concetto di Migratable WoT (*M-WoT*).
In un Migratable WoT le *WT* sono allocate senza soluzione di continuità agli host a seconda delle loro interazioni dinamiche.
In questo paper verranno proposte tre contribuzioni:

1. Architettura del framework *M-WoT*:
   Ci concentreremo sulla migrazione delle *WT* con il loro stato e sulla gestione della procedura di handoff della WT.

2. Formulazione rigorosa dell'allocazione delle *WTs* come problema di ottimizzazione multi obbiettivo.
   Proporremo anche un'euristica basata su grafi.

3. Descriviamo un'implementazione basata su container di M-Wot e una two fold validation.
   Utilizzeremo la two fold validation per verificare la performance della policy di migrazione proposta in una situazione con un setup di edge computing distribuito e in uno scenario di monitoraggio IoT del mondo reale.

2 Introduzione

L'impressionante crescita dell'Internet of Things (*IoT*) in termini di dispositivi connessi e di dati prodotti può essere spiegata dalla versatilità del suo paradigma, che può essere applicato in una grande varietà di casi d'uso, dal manifatturiero digitalizzato alle città smart e al monitoraggio dell'ambiente.

In questi domini la mobilità del servizio ha iniziato a rendersi molto interessante per vari scopi.

Da una parte molte applicazioni IoT operano in ambienti dinamici: di conseguenza alle soluzioni software è richiesto di adattarsi a cambiamenti rapidi

1. nell'utilizzo di banda
2. nell'utilizzo di risorse computazinali
3. nel numero di dispositivi connessi
4. nei requisiti del servizio

Diverse piattaforme per l'IoT forniscono uno strato di adattamento supportando la mobilità senza soluzione di continuità lungo i nodi di un continuo cloud-edge.

Dall'altra parte i dispositivi IoT mobili che generano stream di dati mutevoli nello spazio e nel tempo spingono la ricerca verso delle architetture di calcolo flessibili in grado di auto configurarsi per soddisfare la qualità del servizio (QoS) per le applicazioni utente.

Questo è il caso dell'architettura mobile edge computing (MEC) (e concetti simili come il Cloudlet, il Fog Computing e il follow me cloud) che cerca di eseguire i processi in prossimità delle sorgenti dati. Una caratteristica fondamentale delle architetture MEC è la possibilità di delegare i servizi software ai server edge/fog il più vicino possibile alla posizione corrente dell'utente, utilizzando spesso:

- Tecniche di mobilità di container e/o virtual machine
- Politiche di migrazione guidate dalla mobilità fisica dei dispositivi IoT

La migrazione di servizi non è l'unica sfida aperta nel panorama IoT, che comprende un gran numero di procolli, stack tecnologici ed ecosistemi cloud. La maggior parte degli ambienti IoT sono caratterizzati dall'eterogeneità dei componenti hardware e software e dalla dinamicità delle loro interazioni.
I problemi di interperabilità secondo alcune stime possono ridurre i potenziali introiti fino al 40%. Allo stesso tempo, nuove opportunità di business possono nascere offrendo soluzioni che consentono a sistemi IoT diversi di comunicare fra loro. Anche se gli ecosistemi cloud possono mitigare alcuni dei problemi di interoperabilità con l'utilizzo di tecnologie web (es. api REST, JSON , web sockets ecc...) si basano spesso su architetture "a silo" con un vendor lock-in esplicito o implicito.
Inoltre le soluzioni basate su ecosistemi cloud utilizzando un approccio *sensor-to-cloud* dove i dispositivi sono gestiti utilizzando una connettività basata sul cloud, limitando nuovamente l'estensibilità della soluzione.

Lo standard Web of things del consorzio W3C rappresenza una soluzione recente e promettente per sbloccare il potenziale dell'IoT consentendo l'interoperabilità fra varie piattaforme Iot.
Il supporto per l'interoperabilità è gestito a livello applicativo dfinendo un'interfaccia standard per i componenti IoT (fisica o virtuale) nota come *thing description* che definisce formalmente le capacità e le potenzialità della web thing.
Nonostante la sua recente comparsa, alcune applicazioni interessanti dello standard WoT sono state proposte per diversi domini IoT. Allo stesso tempo però l'implementazione di riferimento del WoT non supporta la mobilità dei componenti software fra i nodi del cloud o dell'edge.
Questo perchè il runtime di una WT va deployato staticamente su un device.
In questo paper affronteremo le due problematiche dell'IoT menzionate in precedenza, ovvero:

- Migrazione dei servizi
- Interoperabilità dei servizi

da una prospettiva WoT: in particolare vogliamo estendere le funzionalità del WoT per gli ambienti IoT supportando l'orghestrazione dinamica e la mobilità delle WT's fra tutte le risorse computazionali disponibili di tutto lo spettro IoT (nodi dell'edge/fog/cloud)
La migrazione delle WT offre nuove opportunità rispetto ad approcci di mobilità software nella letteratura MEC.
Infatti visto che le interazione fra le WTs sono descritte attraverso interfacce software uniformi (ad esempio le TD, le things description) è possibile progettare delle policy di migrazione adattive e precise;
queste policy possono migrare gruppi di WT per soddisfarre i requisiti QoS di sistema considerando le condizioni della rete del mondo reale e del calcolo compuutaizonale, con molta meno complessità computazionale per il mnoitoraggio del servizio che altre soluzioni ad-hoc.
Allo stesso tempo però la mobilità di una WT da un nodo all'altro potrebbe avere un'impatto sulle operazioni di altri WT che la stavano usando.
Quindi le nuove soluzioni devono essere deployate per gestire l'handoff della WT e per garantire la consistenza di tutto il sistema.
Questo paper affronta domande di ricerca legate sia ai *meccanismi di migrazione* delle WT che alle *politiche di migrazione*, es.

- Come rendere possibile la migrazione senza soluzione di continuità di una WT fra due nodi?
- Come ottimizzare le performance di un deployment WoT orchestrando le allocazioni delle WT nel continuo cloud edge?

Per affrontare queste questioni proponiamo il Migratable Web of Things (M-Wot) un nuovo framework architetturale che supporta l'allocazione dinamica delle WT definite dal W3C su i nodi diclcolo disopnibili.
Specificatament, investighiamo come abilitare la migrazione delle WT preservandone lo steto, gestendo la procedura di handoff sui consumatore della WT.
Allo stesso tempo contempliamo la presenza di un servizio orchestratore per il WoT, che è in grado di monitorare le interazioni fra le Wt e di calcolare l'allocazione ottimale delle WT fra i nodi, basandosi su policy ad alto livello (es. si vuole massimizzare la località dei dati, minimizzare la latenza ecc...).
In maniera più dettagliata, questo studio porta sostanzialmente tre contribuzioni:

- Discutiamo dei vantaggi dei meccanismi di migrazione delle WT su due casi d'uso IoT, e poi i componenti dell'architettura M-Wot.
- Formuliamo il problema dell'allocazione delle WT come problema di ottimo multi-obbiettivo. Successivamente proporremo un'euristica centralizzata che mira a bilanciare la comunicazione fra gli host (generata dalle interazioni fra le WTs) e il carico computazionale di ogni host.
- Validiamo le operazioni del M-Wot attraverso due ambienti di test.

Prima valuteremo la performance di diverse policy di allocazione in scenari di edge-computing dove variamo il numero delle WT e il numero di interazioni fra di esse. Successivamente investigheremo l'efficacia del framework M-Wot in uno scenario di monitoraggio IoT dove i servizi di diagnostica in tempo reale sono migrati dinamicamente dal cloud ai nodi di edge in base alle condizioni attuali.

L'analisi della valutazione dimostrerà che l'euristica proposta può bilanciare la comunicazione fra gli host e il carico computazionale in maniera efficace se confrontato con policy di tipo greedy.
Inoltre, nel caso d'uso del monitoraggio Iot la soluzione M-Wot è in grado di ridurre in maniera efficace la latenza di diagnostica rispetto ad un'approccio senza migrazioni allo stato dell'arte.
Il resto del paper è strutturato come segue: la sezione 2 riassume gli approcci di migrazione dei servizi Iot e l'architettura Wot del W3C.

La sezione 3 evidenzia le novità del framework M-Wot e la sua adeguatezza nei casi d'uso IoT selezionati.
La sezione quattro descrive l'architettura M-Wot e i componenti che ne consentono il funzionamento.
La sezione 5 discute del deploymente del WoT come problema di ottimo multi ovviettivo e propone un'euristica basata su un grafo per allocare le WT ai nodi.
La sezione 6 abbozza l'effettiva implementazione del M-Wot.
I risultati sperimentali sono presentati nella sezione 7.
La sezione 8 trae le conclusioni e discute i futuri sviluppi

3 Lavori correlati

Per quanto ne sappiamo, il problema dell'allocazione dinamica e della migrazione live delle WT può essere considerato relativamente nuovo nella letteratura dei sistemi WoT.
Allo stesso tempo ci sono molti paper scientifici che affrontano la migrazioni di servizi software fra i nodi del cloud e i nodi edge per supportare la mobilità fisica dei dispositivi IoT.
Per questo motivo divideremo i lavori correlati in due sezioni.
Prima nella sezione 2-A rivedremo velocemente l'architettura WoT, motivati dalla novità dello standard e dalla necessità di notrudurre la terminoligia utilizzata nel paper; discuteremo anche i (pochi) strumenti e applicazioni sviluppati fin'ora.
Successivamente nella sezione 2-B rivedremo le architetture e le tecnologie che consentono la migrazione dei servizi IoT, concentrandoci principalmente su approcci Mobile Edge Computing (MEC).

3.1 W3C web of things

Il gruppo WoT del W3C ha iniziato le sue attività nel 2015 con l'obbiettivo di definire un insieme di standard di riferimento che consentisse l'interoperabilità fra diversi sistemi IoT.
Il cuore della proposta è la definizione della *Web thing* (WT), che rappresenta l'astrazione di un'entità fisica o virtuale. I metadati e l'interfaccia possono essere descritti formarlmente da un WoT thing description (TD)
. L'architettura di una WT comprende quattro blocchi di nostro interesse:

- Interaction affordances
- Security Configuration
- Protocol bindings
- Behaviour

come mostrato nella figura 1.
I primi tre blocchi sono inclusi nella TD: l'ultimo può essere descritto come una sequenza di di metadati standardizzati comprensibili dalla macchina che consentono ai consumatori di scoprire ed interpretare le capacità di una WT per poterci interagire.
Più nel dettaglio:

- L' *Interaction afforances* (semplicemente affordances da qui in avanti) fornisce un modello astratto di come i consumatori posso interagire ocn la WT,in termini di proprietà (es. le variabili di stato della WT), azioni (es. i comandi che possono essere invocati sulla WT) ed eventi (es. gli eventi inviati dall WT).
- I *protocol bindings* definiscono la mappatura fra le afforances astratte e le strategie di rete (es. i protocolli) hce possono essere utilizzati per interagire con la WT
- La *Security configuration* definisce i meccanismi di controllo dell'accesso alle afforances

La TD può essere codificata con i mezzi del linguaggio JSON-LD, includendo quindi una descrizione semantica.
Infine, il *behaviuour* è l'implementazione della WT,include le affordances (il codice delle azioni).
Tutti i blocchi di cui sopra sono eseguiti all'interno di un runtime software detto Servient che può operare sia come server che come client.
Nel primo il Servient operi come server il si dice che il servient hosta ed *espone* le cose, ovvero crea un oggetto a run-time che serve le richieste verso la WT che sta hostand (come acceddere alle proprietà esposte, alle azioni e agli eventi).
Nel caso in cui il servient operi come client si dice che il servient *consuma* le cose. In questo caso il servient processa la TD, genera una rappresentazione a runtime chiamata la "consumed thing" e la rende disponibile ai client che stanno interagendo con la WT remota.
Data la recente apperizione dello standard, la letteratyra sul WoT è ancora scarsa e limitata a poche applicazioni e strumenti di supporto.
La mappatura di dispositivi IoT del mondo reale a Web Things 23C è stata esplorata in vari paper per i cellulari, l'industria automotive e i le reti di sensori wireless.
Specificatamente nell'ultimo caso gli autori hanno dimostrato come deployare delle applicazioni interoperabili basate si WoT in grado di gestire sensori eterogenei con tre diverse tecnologie di accesso wireless (Wi-Fi, BLE e Zigbee).
Riguardo gli strumenti, oltre a quelli che implementano lo standard W3C WoT in diversi linguaggi di programmazione (es. javascript e python) citiamo la piattaforma WoT store che supporta la gestione senza soluzione di continuità di WT e applicazioni miste in grado di consumare più WT eterogenee allo stesso tempo.

3.2 Migrazione servizi IOT

Una moltitudine di soluzioni sono state proposte per consentire la migrazione di servizi senza soluzione di continuità gra i nodi di un sistema distribuito.
Possiamo classificare gli approcci esistenti in due grandi categorie:

- migrazione statica
- migrazione dinamica

Nel primo caso la migrazione del software è usata come sinonimo di modernizzazione del software, ovvero il processo di adattare le capacità di un sistema esistente per poterlo deployare in un nuovo ambiente; il lettore può fare riferimento ad un certo paper per un sondaggio esaustivo sulla migrazione di sistemi legacy verso software cloud-based.
Nel secondo caso, ovvero quando si parla di migrazione dinamica, si fa riferimento al processo di delegare l'esecuzione a run-time di servizi software da un nodo all'altro.
Ci contreremo sulla migrazione dinamica perchè è più rilevante per lo scopo di questo paper.
Nel dominio IoT possiamo introdurre un ulteriore distizione fra approcci di migrazione indotti dall'utente e approcci di migrazione introdotti dalla mobilità.
Il primo caso include diversi studi su come consentire alle applicazioni mobili di migrare senza soluzione di continuità fra i nodi durante le normali operazioni.
L'obbiettivo finale è quello di offire la migliore quality of experience agli utenti mentre passano da un device all'altro.
Per questo scopo in un paper descrivono il middleware TRAMP per la mobilità precisa delle applicazioni multimediali; la decisione di migrazione è definita manualmente dagli utenti.
Negli approcci indotti dalla mobilità la migrazione software è ottenuta assicurandosi che la gestione e il processing dei dati sia sempre il più vicino possibile alla posizione corrente del dispositivo.
Un modello concettuale di questo tipo è denotato come Mobile Edge Cloud, anche se presenta divere sovrapposizioni con altre architetture allo stato dell'arte, come quella cloudlet, follow me cloud e fog computing.
Il fog computing ha diverse definizioni: in questo paper facciamo riferimento alla proposta in un paper, che definisce il fog clopud come "uno strato di risorse che è fra il dispositivi sull'edge e i data center

nel coud, con caratteristiche che possono somigiliare ad emntrambi.
Un'illustrazione dettagliata delle tecniche di migrazione dei servizi può essere trovata nel paper 5; qui, le sfide uniche del MEC comparate alla migrazione live per i datacenter e alle procedure di handover nelle reti cellulari vengono evidenziate
In maniera simile nel paper 28 gli autori propongono il concetto di Companion Foc Computing (CFC), un architettura software composta di strati distribuiti, uno in esecuzione sul dispositivo mobile e una sul fog server; quest'ultimo è allocato dinamicamente sui nodi di un infrastruttura fog per minimizzare la distanza dalla posizione corrente del dispositivo.
In maniera analoga, lo studio in 29 propone una architettura di rete basata su cloudlet includendo un'algoritmo cooperativo per la mobilità del carico di lavoro fra i nodi del cluster.
In generale le piattaforme correlate al MEC devono risolvere due problemi principali:

- Come definire la strategia di migrazione servizi, considerando lo stato corrente di utilizzo dei nodi oltre che la QoS dell'applicazione IOt
- Come implementare la mobilità software, gestendo anche la migrazione dello stato di esecuzione.

Rispetto al primo problema (la policy di migrazione) la maggior parte delle policy di migrazione che tengono in conto della qualità del servizio si concentrano sul ritardo come il principale indicatore di performance (30) e utilizzano dei modelli con processi decisionali di Markov (MDP) per descrivere l'evoluzione del sistema nel tempo (ovvero la mobilità del dispositivo e le conseguenti azioni di mobilità del servizio).
Visto che i pattern di mobilità possono essere difficili da raccogliere in anticipo, un numero crescente di studi sta investigando le applicazioni di tecnici di machine learning per il calcolo della policy di migrazione ottimal; un'esempio può essere trovato in [3], dove l'utilizzo di tecniche di deep reinforcement learning (DRL) è dimostrato che masssimizzi la reward dell'utente, definita come la differenza fra la QoS e il costo di migrazione.
Fra gli studi che non si concentrano sul ritardo, citiamo la piattaforma auto-organizzante per la gestione del servizio per le smart city proposta in [31] dove la metrica ETX (expected transmission count) è utilizzata per determinare l'allocazione ottimale dei servizi IoT nei fog nodes.
Per quanto riguarda il secondo problema (ovvero la mobilità del software) le macchine virtuali (VM) e i container sono le tecniche più investigate per implementare la migrazione dei servizi con o senza stato.
La migrazione delle VM secondo la mobilità dei dispositivi prevista è considerata in [9]; inoltre per ridurre l'overhead di rete introdotto dal trasferimento della VM viene applicata una tecnica di sintetizzazione di container consentendo ad un nodo fog di riprendere l'esecuzione di una VM applicando dei delta ad un'immagine base.
Le possibilità di effettuare migrazione orizzontali (roaming) o verticali (offloading) di funzioni IoT basate su container dovker è dimostrata in [10].
Da un punto di vista delle performance l'implementazione basata sui container è spesso considerata più adatata per la virtualizzazione ai bordi della rete rispetto all'approccio VM-Bases [32].
Questo è confermato da diversi studi sperimentali, includo lo studio [33] che investiga l'implementazione di meccanismi di virtualizzazione per la gestione dei dati IoT e dimostra che l'impatto energetico su computer single-board è trascurabile.
Un alternativa all'utilizzo delle macchine virtuali e dei container è costituita dalla migrazione del codice attivo, per questo scopo il framework ThingMigrate consente la migrazione di processi Javascript attivi fra diverse macchine utilizzando dei meccanismi di iniezione per tracciare lo stato locale di ogni funzione.
Rispetto agli studi citati fino ad'ora la migrazione delle WT affrontata in questo paper può essere considerata un'istanza speciale di una migrazione dinamica basata sugli agenti [35]; allo stesso tempo presenta nuove opportunità oltre che nuove sfide tecnichce che sono discusse in dettaglio nella sezione che segue.

3.3 M-WOT definizioni preliminari e motivazioni

Consideriamo uno scenario distribuito composto da un insieme di nodi di calcolo distribuiti lungo tutto lo stack dello spettro Iot (dall'edge al cloud) come mostrato nella figura 2; ogni nodo è abilitato al WoT, ovvero puà hostare uno o più servient (l'ambiente di runtime dell'architettura WoT) e ogni servient contiene una singla WT in stato di esecuzione.
Definiamo la migrazione WT come la capacità di delegare dinamicamente un WT fra diversi nodi, stoppando l'esecuzione al nodo sorgente e riprendendola al nodo di destinazione.
Presumiamo che il processo di migrazione sia stateful, ovvero lo stato interno di una WT e la su TD doverebbero essere spostati e aggiornati insieme al codice.
In particolare tutti i valori delle Properties e le informazioni che descrivono il contesto computazione della WT doverebbero essere considerati parte dello stato quindi migrati.
Rispeto agli approci di migrazione classici (VM, container, agent based) visti in precedenza la migrazione della WT presenta dei vantaggi e delle nuove sfide di ricerca da affrontare:

- Challenge: thing handoff management. Il WoT consente le interazione senza soluzione di continuità fra software eterogenei durante le operazioni di consumo della WT; se una WT migra su un altro nodo, tutte le altre WTs che la stavano consumando devono essere notificare per poter aggiornare i loro oggetti consumati e puntare al nuovo indirizzo della TD.
  Questo caso d'uso è rappresentato nelal figura 2, dove entrambe le WT A e B stanno consumando la WT C; La WT C verrà migrata dall'host 1 all'host 2 in un'istante futuro.
  Per questo motivo una procedura di signalling adeguata deve essere effetuata per informare A e B di quando l'attivazione della WT C sull'host 2 è stata completata, in modo tale che possano consumare nuovalmente la TD della WT C.
  Inoltre il processo di migrazione introduce un'intervallo di handoff durante il quale la WT C potrebbe non essere in grado di processare le invocazioni remote dalla WT A e B, la durata di questo handoff è chiaramente un parametro critico che determina la performance del sistema-

- Advantage: support to group migrations.
  Continuando rispetto al punto precedente, un framework per la migrazione WoT potrebbe supportare la mobilità di gruppi di componenti software (rispetto ad un singolo servizio come accade negli approcci MEC) come conseguenza delle dipendenze attive di dati (le interazioni) fra le WT, a fianco della mobilità fisica dei dispositivi IoT.
  Infatti ogni WT espone le proprie afforances attraverso la TD in maniera standardizzata; come risultato è possibile costruire un grafico delle dipendenze real-time fra tutte le WT di un sistema WoT distribuito e conseguentemente progettare policy di allocazione che determinano migrazioni di gruppo di sottoinsiemi di WTs che interagiscono fra di loro per massimizzare la località dei dati.
  Chiaramente le policy migrazioni basate su gruppi potrebbero anche essere deployate su altre architettura a micro servizi; tutta via, per il caso del M-WoT questa feature potrebbe essere supportata in un modo agnostico rispetto al proctcollo visto che le interazioni fra le WTs occorono secondo un'interfaccia standard e quindi potrebbero essere gestite attraverso lo strato di monitoring del M-WOT descritto nella sezione IV-C.

Le figure 3(a) e 3(b) mostrano due possibili casi d'uso della migrazione WoT, collegate a due modelli concettuali di WTs differenti: la migrazione del servizio di processamento dei dati e la migrazioni dei digital twin.
In maniera più specifica, la figura 3a rappresenta un'applicazione di structurual health monitoring basata su tecnologie IoT/WoT per come è stata proprosta insiene ad altri dal progetto MAC4PRO.
Assumiamo che il sistema di monitoraggio possa lavorare in due modalità diverse: normale e critica, dentando due requisiti QoS per il rilevamento dei rischi.
All'estremità dell'edge ci sono i sensori (es. degli accelerometri) che monitorano le vibrazioni deòlla costruzione nel coros del tempo.
I dati dei sensori sono resi disponibili attraverso la SWT (sensor web thing) che fornisce funzionalità di querying dei dati e querying e aggiornamento dello stato del dispositivo.

L'analisi dei dati dei sensori è gesista dalle WTs migrabili T_1 , T_2, T_3 e T_4 che implementano rispettivamente le funzionalità di data fusion, data cleaning, data alerting, data forecasting.
Nella modalità normale T_1 e T_2 sono in esecuzione su nodi fog in prossimità della struttura monitorata, mentre T_3 e T_4 sono hostati in remoto sul glod; questo introduce della latenza di rete nel rilevare anomalie o situazioni di pericolo (calcolate da T_3) ma allo stesso tempo minimizzano il carico sui nodi fog.
Ad un certo punto dell'esecuzione del sistema presumiamo che anomali dei dati consecutive sono notate all'interno dei dati (T_2) per qui i sistema di monitoraggio passa dalla modalità *normale* alla modalità *critica*; questa azioni potrebbe anche richiedere un grado di responsiveness più alto per il sistema di diagnostica.
Nell'ambiente M-WOT il cambio di modalità può essere gestito automaticamente migrando il servizio T_3 dal cloud ai nodi fog (o viceversa quando la modalità torna ad essere *normale*) segna nessun bisogno di una configurazione manuale e senza introddure alcun meccanismo di signaling esplicito fra le WTs coinvolte (ovvero T_2 e T_3).
La figura 3(b) rappresenta il secondo caso d'uso dove la migrazione coinvolge i digital twins del WoT. I digital twin sono definiti nello standard W3C come una rappresentazione virtuale di un dispositivo o di un gruppo di dispositivi che risiedono sul cloud o su un nodo dell'edge (...) possono modellare un singolo dispositivo o possono aggregare più dispositivi in una rappresentazione virtuale dei dispositivi combinati.
A questo proposito, consideriamo un applicazione WoT per l'industria automotive come quella proposta in [17]; una WT è associata ad ogni componente all'interno dei veicoli per consentire accesso senza soluzione di continuità e interazioni con i segnali dell'auto.
Similmente a [17] le sensor web things sono incaricate di recuperare i dati dall'hardware del veicolo.
Inoltre presumiamo la presenza si una Veichle web thing, definita come il digital twin del veicolo per intero; la VWT è l'unico punto di accesso ad un sottoinsieme di proprietà azione ed eventi delle SWT, ma espone anche nuove affordances derivate dall'analisi e dall'elaborazione dei dati di più sensori es. per la diagnostica in tempo reale del veicolo.
A causa dei requisiti enerfitivi la VWT è hostata al di fuori del veicolo, su un fog node di proprietà del comune
. Mentre il veicolo su muove nei limiti dello scenario, la sua VWT veiene generata automaticamante sul nodo fog più vicino, in maniera simile alle applicazioni MEC, anche se qui è la mobilità fisica del dispositivo ad indurre la mobilità del digital twin della WT.
In aggiunta immaginiamo uno scenario in un acittà con molti VMTs diversi fra loro, associati a diversi tipi di veicoli (es. auto, bici, autobus eccc...); le VMT sono successivamente consumate dalle City web things nel cloud per fornire servizi avanzati legati alla mobilità, come smart parking, monitoraggio del traffico, routing multi modale eccc...
Evidenziamo il fatto che il numero di VMT può essere molto dinamico nel tempo, ovvero nuove things potrebbero essere create o eliminate come effetto della mobiilità terrestre; in maniera simile il carico compuitazionale per l'esecuzione delle VMT e della CWT potrebbe variare nel tempo.
Nel nostro ambiente M-WoT le VMT sono allocate dinamicamente fra i nodi del cloud e del fog quando compaiono nel sistema; inoltre, policy di load balancing multi obbiettivo possono essere utilizzate, ad esseempio per minimizzare la distanza dall'orgine dei dati (es. il veicolo) mentre si massimizza l'utilizzo delle risorse di calcolo dei nodi fog e cloud.

4 Architettura M-WOT

L'architettura sotware del M-WOT è rappresentata nella figura 4. Presumiamo un'insieme di servients del WoT deployati su diversi nodi, ogni servient hosta esattamente una WT.
In maniera differente rispetto ad un deploymente WoT legacy, che si presume essere statico, il M-Wot consente la mobilità delle WT fra nodi diversi.
A questo scopo il M-WoT offre due nuovi componenti, l'orchestrator e il thing directory; questi moduli non migrano e possono essere deployati o sull'edge (se i requisiti computazionali sono soddisfatti) o

sui server nel cloud.
Inoltre, uno strato di monitoraggio è stato aggiunto allo stack dei servient.
In questo paragrado descriveremo in dettaglio la struttura interna dei tre moduli, mentre nella sezione IV-D chiarificheremo le operazioni dei modili quando avviene il processo di migrazione di una WT.

4.0.1 Thing directory

Il thing directory serve come registro delle risorse del M-WoT ovvero delle thing descriptors attive.
Più nel dettaglio, assumiamo due tipi di TD, una associata alle WT e una ai servient: queste ultime descrivono le capacità dell'ambiente di runtime e sono utilizzate per abilitare le funzionalità dello strato di monitoraggio descritto nella sezione IV-C.
Una volta attivato ogni servient registra la propria TD e la TD della WT che hosta.
Il thing directory gioca due ruoli principali.
Funziona come servizio di discovery, quando viene interrogato dai client returna la lista dei TD che rispettano i parametri di interrogazione, come risultato l'orcheestratore può essere al corrente della lista dei servients disponibili al momento nello scenario WoT.
Come secondo ruolo supporta notifiche push verso le WTs o verso i servient, quando specifici eventi di sistema sono rilevati, ad esempio quando una WT comleta la procedura di handoff.
A questo scopo presumiamo che la WT T_1 sia stata consumata dalla T_2 che accede periodicamente ad una delle sue proprietà.
Se T_1 è migrata su un nodo diverso la pipeline dei dati si rompe a meno che T_2 sia notificata dell'evento di mobilità e della nuova posizione del servizio.
Il processo di notifica è illustrato nel diagramma di sequenza della figura 7, discussa successivamente nella sezione IV-D.
Alternativamente si potrebbe utilizzare un meccanismo di polling (coinvoltendo T_1 e TDir nel nostro esempio).
Tuttavia questo approcio potrebbe introdurre dell'overhead di rete significativo con conseguente spreco di banda, per cui non è stato considerato nella nostra soluzione.

4.0.2 WT orchestrator

L'orchestrator costituisce il componente core dell'architettura M-WoT.
Come spiegato in precedenza sfrutta la TDir per recuperare la lista dei servient attivi (ovvero delle loro TD).
Successivamente interroga periodicamente ogni servient attraverso al sua interfaccia WoT per raccogliere statistiche live, come l'utilizzo della CPU e il traffico di rete generato dalle interazioni WT.
Basandosi sui valori delle metriche ricevute esule politiche di ottimizzazioni in uso, l'orchestrator determina l'allocazione adatta delle Wts e/o dei servient sui nodi.
Il piano di allocazione è poi trasferito ad un layer sottostante (esterno al M-Wot) chiamato in genere qui il *Migration substrate* che è incaricato di implementare la mobilità fisica del software fra i nodi sorgenti e i nodi di desintazione.
I passi appena citati sono eseguiti continuamente dall'orchestrator durante la vita del sistema; come risultato la dinamicità dell'ambiente WoT/IoT riguardo alla creazione /distruzione delle WT, variazioni nell'utilizzo della banda e aggiornamenti a rutime delle policy sono supportati a pieno.
Inoltre, per favorire l'estensibilità della piattaforma la struttura dell'orchestrator è stata divisa in tre sottomoduli principali che riflettono la pipeline interna dei dati:

1. Thing manager: polla periodicamente i dati dalla TDir per gestire la lista dei servient/Wts attivi e le rispettive TD.
   La lsita è utilizzata per raccogliere report periodici da ogni servient.

2. Optimizer: esegue la policy di allocazione per le WT/servient.
   Allo stato corrente dell'implementazione il modulo hosta l'algoritmo di ottimizzazione sbasato su grafi definito nella sezione V e altre policy greedy valutate nella sezione VII.

Tuttavia sottolineiamo il fatto che ogni policy che imlementa l'interfaccia verso lo lo strato superiore (ovvero il thing manager) e lo strato inferiore (ovvero la migrazione) può essere installata ed utilizzata

3. Migration: riceve il piano di deployment dell'optimizer e implementa le procedure di handoff per le WT.
   Prima stoppa l'esecuzione delle WT per migrarle verso i loro nodi, poi, attraverso connettori specifici comanda azioni al Migration Substrate per abilitare il trasferimento fisico dei servient (e delle wt hostate) dai nodi sorgenti ai nodi di destinazionie.

L'architettura M-Wot non dipende su nessuna tecnologia specifica di mobilità del software.
Abbiamo invece introdotto uno strato di astrazione - chiamato il Migration substrate - che può adottare una qualsiasi soluzione allo stato dell'arte (attraverso dei migration connectors adatti) come container docker, macchine virtuali o processi javascript.
Questo connettori attuerano il piano restituito dall'optimizer che utilizzeranno come input.
Concretamente l'implementazione corrente si affida a Docker Swarm come migration connector di default, come meglio dettagliato nella sezione VI.

4.0.3 Servient M-WoT

Infine il framework WoT introduce delle piccole modifiche al runtime Sservient per fornire all'optimizer dati in tempo reale sulle performance del sistema.
Più specificamente uno strato di API di monitoraggio è stato introditti fra l'applicazione WT e il runtime di scripting come raffigurato nella figura 6.
Questo strato è incaricato di interecettare le invocazione all'api di scripting e di generare periodicametne dei Thinkg Report (Trs).
Questi ultimi opssono essere considerati come uno snapshot dell'esecuzione corrente del servient/WT e contengono i valori delle metriche (sia del servient che della WT) richieste dall'optimizer;
Nell'appendice riporteremo un frammento della struttura della TR in uso.
Il monitoring layer esponde tutti dati raccolti attraverso un'azione apposita nelle afforfance che è stata aggiunta alla TD del serveint; invocandola l'Orchestrator può richiedere una nuova richiesta di generazione di TR.

4.1 Esempio di migrazione

Per riassumere le operazioni dei tre componenti presentati fino ad ora, vediamo un esempio di un processo di migrazione di una WT.
Consideriamo due WT/Servient, rispettivamente T_A/S_A e T_B/S_B (con T_A in esecuzione su S_A e T_B su S_B) hostati sui nodi N_1 ed N_2.
Assumiamo anche che T_B abbia consumato T_A e che stia leggendo periodicamente alcune delle sue proprietà.
All'istante t il thing manager interroga S_A e S_B per raccogliere le TR; questo è implementato consumando le TD dei servient e inviando un comando retrieveReport (dettagli nella sezione VI).
Successivamente l'otimizer viene eseguito: una nuova allocazione viene prodotta dove T_A deve essere spostato sul nodo N_2.
La sequenza di operazioni che effettuano la migrazione di T_A da N_1 ad N_2 è mostrata nella figura 7.
Per prima cosa l'esecuzione corrente di T_A viene stoppata: questo viene fatto dall'orchestrator (e più specificamente dal sottomodulo migration) invocando l'azione stop su S_A, che di conseguenza stoppa l'applicazione della WT, pulisce le risorse di sistema recupera il contesto dei dati applicativi (lo stato corrente) e lo ritorna all'orchestrator.
Di conseguenza il contesto applicativo di T_A è salvato come metadati all'intendo di TDir per essere utilizzato successivamente.
In sequito l'orchestrator (utilizzando un connector adatto) invia una richiesta al migration substrate (es docker swarm) per far muovere T_A/S_A al nodo di destinazione (N_2).

Dopo che S_A è stato rigenerato registra la sua nuova TD (con gli indirizzi di rete delle sue affordances aggiornate) nel TDir.
Di seguito interroga la TDir per recuperare il contesto della T_A, quest'ultimo viene deserializzato e iniettato come oggetto globale nello script applicativo della T_A4
Infine T_A inizia il processo di inizializzazione e si espone facendo partire la registrazione della propria TD all'interno della TDir.
A questo punto T_A riprende nell ostato in cui era prima di essere stoppata ed è considerata completamente migrata.
La TDir invia una notifica a T_B riguardo la procedura di handoff, T_B recupera la nuova TD di T_A dal TDir e la consuma di nuovo per poter puntare alla posizione del servizio aggiornata.
Infine T_B ricomincia ad interagire con T_A e ad accedere alle sue affordances.

5 M-WOT migration policy

In questa sezione caratterizzeremo formalmente le operazioni dell'optimizer come problema di ottimo multi ovviettivo.
Per lo scopo di questo studio consideriamo un processo di ottimizzazioni a due passaggi che considera il problema del load balancing (ovvero quanto carico ha ogni host) e l'overhead delle comunicazioni di rete (ovvero quanti dati vengono scambiati fra gli host).
Il problema di ottimizzazione è definito formalmente nella sezione V-A.
Successivamente un'euristica basata su grafi viene proposta nella sezione V-B, e la sua complessità computazionale è definita formalmente nella sezione V-C.
La tabella 1 riporta la lista delle variabili nella sezione V-A.

5.1 Formulazione del problema

A prescindere dal caso d'uso target, consideriamo un generico deployment WoT con N_{WT} WT attive.
Il sistema si evolve su una serie di time-slot $T = \{t_0, t_1...\}$; ogni slot ha una durata di t_{slot} secondi ed è uguale all'intervallo fra le esecuzioni consecutive della policy di migrazione.
Poniamo $WT = \{wt_1, wt_2, ...wt_{N_{WT}}\}$ come l'insieme delle WT, che possono essere eterogenee in termini di modello dei dati (es. le affordances).
Senza perdere in generalità poniamo $A_i = \{a_i^1, a_i^2, ...a_i^{N_{A_i}}\}$ le affordances esposte dalla wt_i nella sua TD, ogni affordance può rappresentare una proprietà, un'azione oppure un evento.
Presumiamo che A_i sia statico, ovvero wt_i non può aggiornare la propria TD a runtime (es. non può definire nuove proprietà).
Poniamo H l'insieme dei nodi di host, con $H = \{h_1, h_2, ...h_{N_H}\}$ e assumiamo che i nodi siano eterogenei in termini di hardware.
Ogni nodo potrebbe infatti avere una potenza di calcolo diversa, senza perdere in generalità il nostro modello attraverso un'indice di potenza computazionale $\gamma(h_l), \forall h_l \in H$ che astrae i dettagli dell'hardware ed è definito come il numero massimo di things che possono essere eseguite sull'host.
L'allocazione dei servient agli host è definita dalla funzione policy $P : WT \times T \to H$;
per ogni WT wt_i il valore $P(wt_i, tk) = h_m$ specifica la macchina (ovvero h_m) che la sta hostando all'istante di tempo t_k.
Basandosi sull'output della policy di allocazione, l'insieme $PT_{m,k} \subseteq WT$ denota la lista delle WT che sono hostate dall'host h_m nel time slot t_k, ovvero $PT_{m,k} = \{wt_i \in WT | P(wt_i, t_k) = h_m\}$.
Secondo l'architettura WoT vista nella sezione due, ogni WT wt_i può interagire con un altra WT wt_j se prima la consuma.
Questo cosa è modellata con l'assunzione che, in ogni time slot t_k, wt_i può inviare una lista di richieste $R_{i,j,k} = \{r_{i,j,k}^1, r_{i,j,k}^2...\}$ alla wt_j consumata; ogni richiesta $r_{i,j,k}^y$ fa riferimento ad un affordance di wt_j, e consiste in:

- Lettura o scrittura di una proprietà

- Invocazione di un evento
- Elaborazione di un evento

Vale la pena evidenziare che la notazione sopra presume che la stessa affordance a_i^x potrebbe essere attivata più volte da wt_i durante lo stesso timeslot, ma vengono considerate due richieste diverse (ad esempio la WT wt_i legge due volte la stessa proprietà a_j^x sulla WT wt_i nel time slot t_k).
L'implementazione di ogni richiesta $r_{i,j,k}^y$ comporta dello scambio di dati fra le WT wt_i e wt_j; definiamo $B(r_{i,j,k}^y)$ come i dati scambiati (in byte) fra le due WT, includendo sia gli eventuali parametri passati da wt_i a wt_j sia eventuali valori di ritorno da wt_j a wt_i.
Il valore $B(r_{i,j,k}^y)$ è incluso nel messaggio TR, che è periodicamente mandato da ogni WT all'optimizer come descritto in precedenza nella sezione 4.
Denotiamo con $B(i,j,k) = \sum B(r_{i,j,k}^y)\ \forall r_{i,j,k}^y \in R_{i,j,k}$ il carico totale di comunicazione fra le WT wt_i e wt_j all'istante di tempo t_k.
Chiaramente, $B(i,j,k) = 0$ sia nel caso in cui wt_i non stia consumando wt_j all'istante t_k o nel caso in cui non ci siano interazioni fra loro (ovvero $R_{i,j,k} = \emptyset$).
L'obbiettivo dell'optimizer è determinare la policy che calcola - ad ogni istante di tempo t_k - il trade-off ottimale fra l'utilizzo di risorse computazionali (ovvero il load balancing fra gli host) e la località dei dati (ovvero quanti dati sono trasferiti fra gli host),
A questo scopo definiamo la metrica del network overhead (NO) come il carico di comunicazione fra gli host (in byte) che occorre a causa delle interazioni fra le WT hostate su nodi diversi. Più formalmente:

$$NO(t_k) = \sum_{wt_i \in WT, wt_j \in WT, P(wt_i,t_k) \neq P(wt_j,t_k)} B(i,j,k)$$

. Chiarifichiamo che la metrica $NO(t_k)$ sopra quantifica le comunicazioni *end-to-end* dello strato applicativo fra i nodi del cluster, enerete dalle interazioni fra le diverse WT; non include l'overhead dello strato di rete (es quello casusato da forwarding multi hop di messaggi fra i router).
Questo perché il framework M-Wot è implementato nello strato di applicazione e la conoscenza della topologia della rete sottostante non è nota.
Introduciamo poi la metrica di host fairness (HF) defintia come la differenza fra il nodo più carico ed il nodo meno carico del cluster, ovvero:

$$HF(t_k) = \max_{h_m \in H} L(h_m, t_k) - \min_{h_m \in H} L(h_m, t_k)$$

Qui $L(h_m, t_k)$ definisce il carico computazionale di h_m nel time slot t_k ed è collegato al numero di WT hostate diviso la potenza computazionale, ovvero:

$$L(h_m, t_k) = \frac{|PT_{m,k}|}{\gamma(h_m)}$$

Definiamo $p_{wt_i,h_m}^{t_k}$ come la variabile binaria che indica l'allocazione delle WT, definita $\forall t_k \in T$, $\forall wt_i \in WT$ e $\forall h_m \in H$ come segue:

$$p_{wt_i,h_m}^{t_k} = \begin{cases} 1 \text{ se } P(wt_i, t_k) = h_m \\ 0 \ \textit{altrimenti} \end{cases}$$

Attraverso le metriche NO ed HF introdotte sopra, il problema di migrazione può essere formalmente definito come segue:

$$\min_{p_{wt_i,h_m}^{t_k}} \ NO(t_k)$$

Tale che;

$$L(h_m, t_k) \leq 1 \ \forall h_m \in H$$

$$HF(t_k) \leq \Delta$$

Il primo dei due vincoli serve per assicurarsi che l'allocazione su ogni host non ecceda le capacità computazionali per quell'host ($\gamma(h_m)$).
Nel secondo vincolo Δ è un parametro definito dall'utente che quantifica il trade-off menzionato in precedenza.
È facile notare che le metriche HF ed NO sono legate fra loro: minimizzare il carico di rete si può raggiungere con una policy che alloca tutte le WT hall ostesso host.
Tuttavia questo costituisce il caso peggiore per l'HF.
Per questo motivo abbiamo due scenari estremi:

1. L'obbiettivo del sistema è quello di minimizzare lo scambio di dati sulla rete, a prescindere dalla latenza del servizio.
   Questo potrebbe essere il caso di uno scenario Iot edge-cloud dove lo stakeholder ha interesse nel mimizzare la quantità di dati trasferita ad un'infrastruttura remota per ragioni di privacy.
   In questo caso $\Delta = \infty$

2. L'obbiettivo del sistema è quello di minimizzare la latenza del servizio, evitando la presenza di bottleneck di performance (es host sovvraccaricati) mitigando comunque le comunicazioni fra host.
   In questo caso $\Delta = 1$

Tutte le situazioni intermedie sono modellate personalizzando il parametro Δ, che presumiamo essere l'input del problema di ottimizzazione.

5.2 Euristica proposta