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

Michele Beccari 856608

2023

Indice I

Indice II

Introduzione

Lo standard Web of things

Standard promosso dal W3C per progettare sistemi IoT interoperabili in grado di gestire l'eterogeneità delle piattaforme software e dei dispositivi

Cosa consente di fare?

Consente la creazione di scenari di IoT caratterizzati da una moltitudine di Web Things (WTs) che comunicano secondo delle interfacce software ben definite

Tuttavia...

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.

Introduzione

Obbiettivo del 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.

Migratable WoT (M-WoT)

Vogliamo sfruttare lo standard WoT e in particolare la sua capacità di standardizzare le interfacce software delle WT per proporre il 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

Introduzione

Gli ambienti IoT sono ambienti dinamici

Le applicazioni software IoT devono adattarsi a cambiamenti rapidi:

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

Per questo motivo

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

Introduzione

Inoltre

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.

Per questo motivo

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

Introduzione

MEC

L'architettura mobile edge computing (MEC) cerca di eseguire i processi in prossimità delle sorgenti dati.
Questo avviene utilizzando:

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

Introduzione

Problemi di interoperabilità

La maggior parte degli ambienti IoT sono caratterizzati dall'eterogeneità dei componenti hardware e software e dalla dinamicità delle loro interazioni.

Possibili soluzioni

- Nuove opportunità di business possono nascere offrendo soluzioni che consentono a sistemi IoT diversi di comunicare fra loro.
- Gli ecosistemi cloud possono mitigare alcuni dei problemi di interoperabilità con l'utilizzo di tecnologie web (es. api REST, JSON , web sockets ecc...)

Ma...

Spesso le soluzioni si basano spesso su architetture "a silo" con un vendor lock-in esplicito o implicito.

Lo standard WoT del W3C - Che cos'è

Che cos'è

Standard pubblico nato nel 2015 con l'obbiettivo di definire un insieme di standard di riferimento che consentisse l'interoperabilità fra diversi sistemi IoT.

Le novità

Definizione della Web thing (WT), che rappresenta l'astrazione di un'entità fisica o virtuale.
I metadati e l'interfaccia dell'entità possono essere descritti formalmente da una WoT thing description (TD)

Thing description

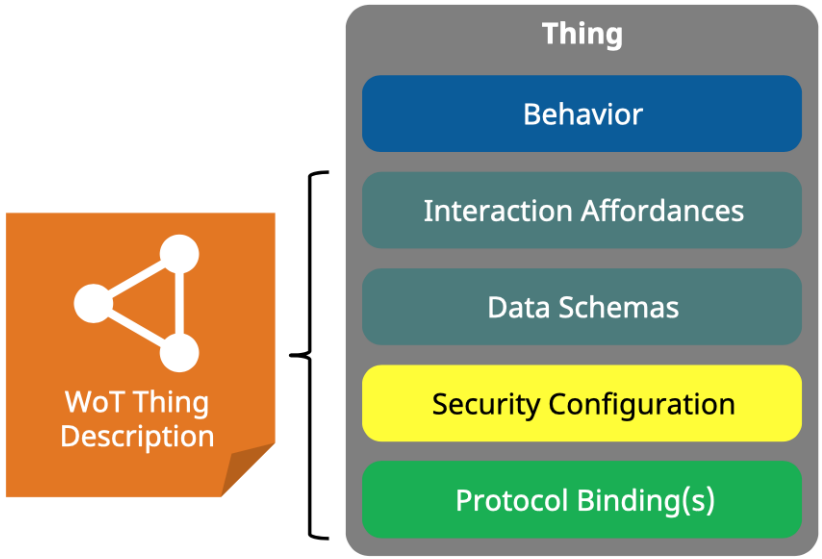
Interfaccia standard per i componenti IoT (fisici o virtuali) che definisce formalmente le capacità e le potenzialità delle web thing.

Lo standard WoT del W3C - Architettura

L'architettura

L'architettura di una WT comprende quattro blocchi di nostro interesse:

- Interaction affordances
- Security Configuration
- Protocol bindings
- Behaviour



I blocchi della WT

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

Lo standard WoT del W3C - Architettura

In dettaglio

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

Interaction Affordances

Fornisce un modello astratto di come i consumatori possono interagire con la WT, in termini di proprietà, azioni ed eventi.

Protocol Bindings

Definiscono la mappatura tra le affordances astratte e le strategie di rete (protocolli) che possono essere utilizzate per interagire con la WT.

Lo standard WoT del W3C - Architettura

Security Configuration

Definisce i meccanismi di controllo dell'accesso alle affordances.

Behaviour

Consiste nelll'implementazione della WT,incluse le affordances (il codice delle azioni)

Servient

Runtime all'interno del quale vengono eseguiti tutti i blocchi della WT, può operare in modalità server o in modalità client.

Lo standard WoT del W3C - Servient

Modalità server

Il servient hosta ed *espone* le cose, ovvero crea un oggetto a run-time che serve le richieste verso la WT che sta hostando (come accedere alle proprietà esposte, alle azioni e agli eventi)

Modalità client

Il servient *consuma* le cose. In questo caso il servient processa le TD, genera una rappresentazione a runtime chiamata la "consumed thing" e la rende disponibile ai client che stanno interagendo con la WT remota.

Lo standard WoT del W3C - Limiti e obbiettivi

I limiti

L'implementazione di riferimento del WoT non supporta la mobilità dei componenti software fra i nodi del cloud o dell'edge

Gli obbiettivi del paper

Vogliamo estendere le funzionalità del WoT per gli ambienti IoT supportando l'orchestrazione dinamica e la mobilità delle WT's fra tutte le risorse computazionali disponibili di tutto lo spettro IoT (nodi dell'edge/fog/cloud).

- 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?

M-WOT - migrazione

Migrazione di una WT

Definiamo la migrazione WT come la capacità di associare dinamicamente un WT a diversi nodi, stoppando l'esecuzione al nodo sorgente e riprendendola al nodo di destinazione.

Stato della WT

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.

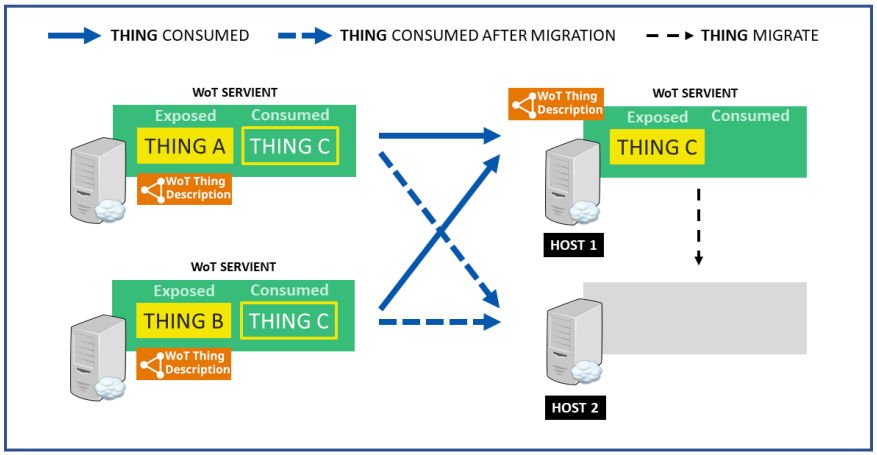
In particolare

Tutti i valori delle Properties e le informazioni che descrivono il contesto di computazione della WT doverebbero essere considerati parte dello stato quindi migrati.

M-WOT - scenario d'esempio

Lo scenario

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 a lato; ogni nodo è abilitato al WoT, ovvero può hostare uno o più servient e ogni servient contiene una singla WT in stato di esecuzione.



M-WOT - challenge

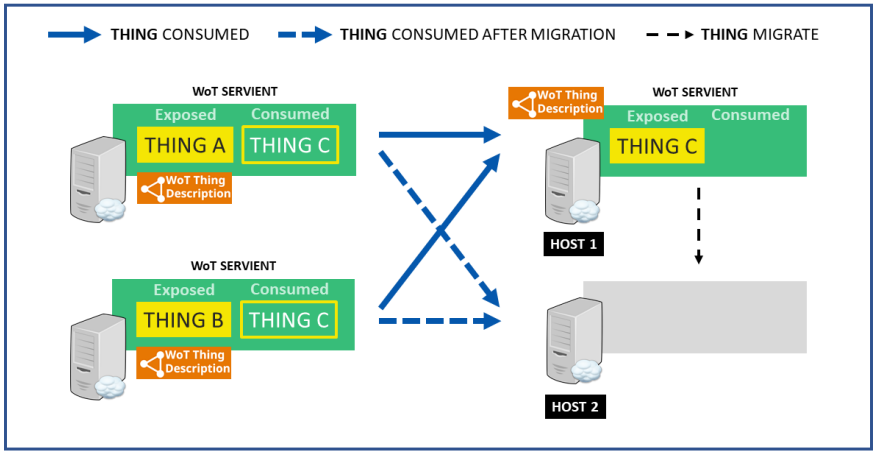
challenge: Thing handoff management

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.

M-WOT - challenge

Thing handoff management

Nella figura 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.



Quindi

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 nuovamente 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.

M-WOT - advantage

L'idea

Un framework per la migrazione WoT potrebbe supportare la mobilità di gruppi di componenti software come conseguenza delle dipendenze attive di dati (le interazioni) fra le WT

Grazie allo standard WoT

Ogni WT espone le proprie affordances attraverso la TD in maniera standardizzata

Questo constente di

Costruire un grafico delle dipendenze real-time fra tutte le WT di un sistema WoT distribuito.

M-WOT - advantage

Utilizzando il grafo

Diventa possibile progettare policy di allocazione che determinano migrazioni di gruppo di sottoinsiemi di WTs che interagiscono fra di loro per massimizzare la località dei dati.

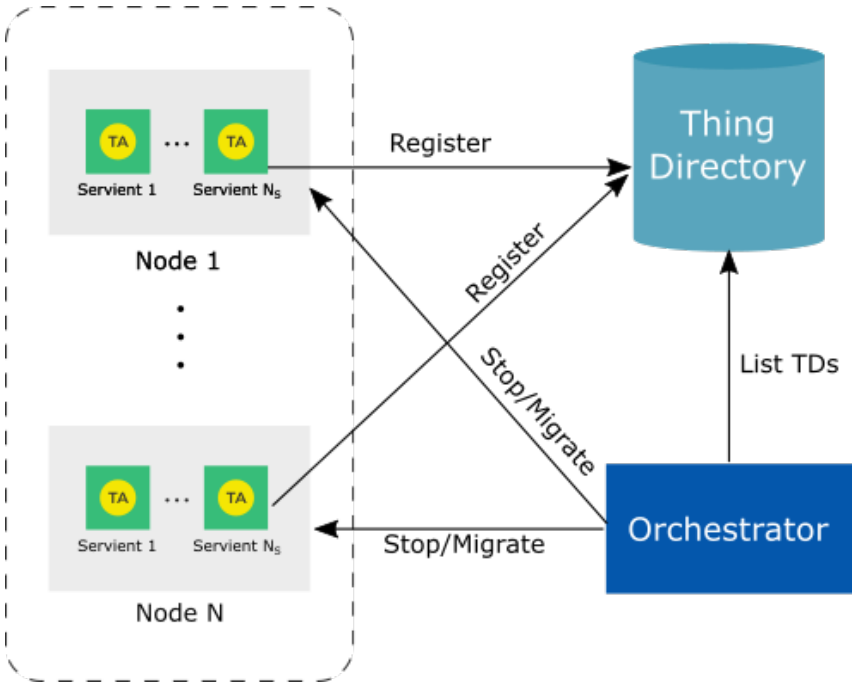
M-WOT - architettura

I componenti del deployment

Consideriamo un insieme di servients del WoT deployati su diversi nodi, ogni servient hosta esattamente una WT.

Le differenze

In maniera differente rispetto ad un deployment WoT legacy, che si presume essere statico, il M-Wot consente la mobilità delle WT fra nodi diversi.

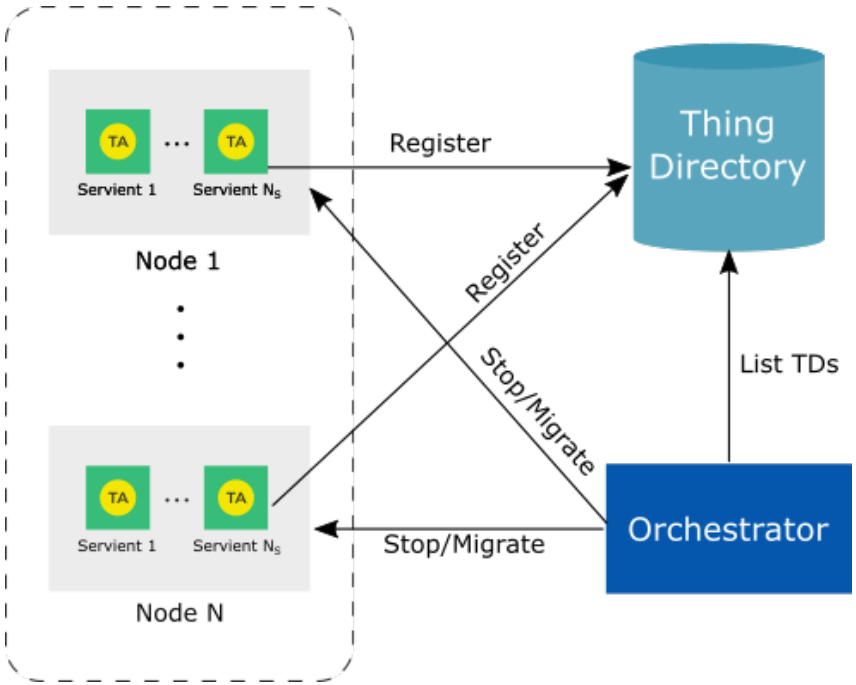


M-WOT - architettura

Per gestire la mobilità

Il M-WoT offre due nuovi componenti, che non migrano.

- L'orchestrator
- Thing directory



M-WOT - thing directory

Il thing directory

Serve come registro delle risorse del M-WoT, ovvero delle thing descriptors attive.
Svolge due funzionalità:

- Servizio di discovery
- Supporta le notifiche push

Servizio di discovery

Quando viene interrogato dai client (soprattutto dall'orchestrator) ritorna la lista dei TD che rispettano i parametri di interrogazione

Supporto alle notifiche push

Invia notifiche verso i servient, quando specifici eventi di sistema sono rilevati (ad esempio quando una WT comleta la procedura di handoff.)

M-WOT - orchestrator

L' orchestrator

Costituisce il componente core dell'architettura M-WoT.

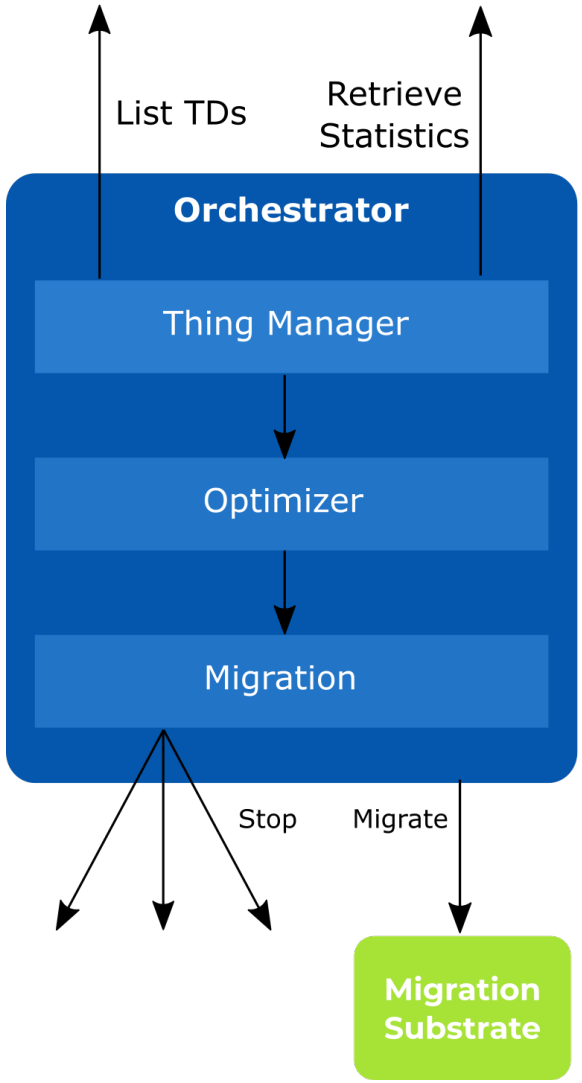
Per tutta la vita del sistema

1. Usa la TDir per recuperare la lista dei servient attivi (ovvero delle loro TD)
2. Interroga ogni servient attraverso la sua interfaccia WoT per raccogliere statistiche live (es. uso della CPU o traffico di rete)
3. Basandosi sui valori delle metriche ricevute e sulle politiche di ottimizzazioni in uso determina il piano di allocazione dei servient sui nodi.
4. Trasferisce il piano di allocazione ad un layer sottostante (esterno al M-Wot) chiamato in genere il *Migration substrate* (incaricato di implementare la mobilità fisica del software fra i nodi sorgenti e i nodi di destinazione)

M-WOT - orchestrator - struttura interna

La struttura interna

Per favorire l'estensibilità della piattaforma la struttura dell'orchestrator è stata divisa in tre sottomoduli principali



M-WOT - orchestrator - struttura interna

Thing manager

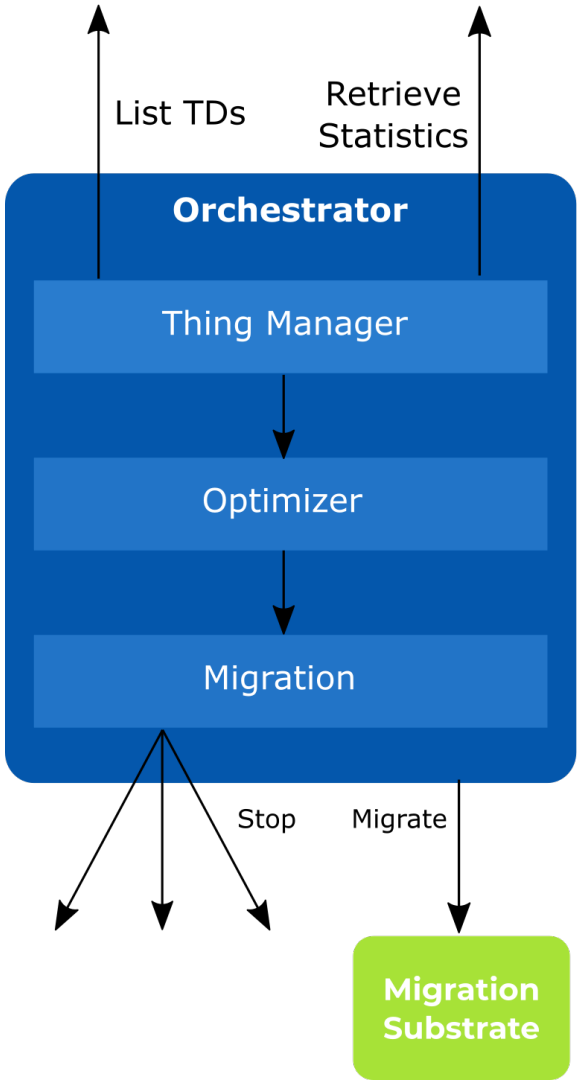
Polla periodicamente i dati dalla TDir per gestire la lista dei servient attivi e le rispettive TD.

Optimizer

Esegue la policy di allocazione per i servient.

Migration

Riceve il piano di deployment dell'optimizer e implementa le procedure di handoff per i servient.



M-WOT - servient

Servient M-WoT

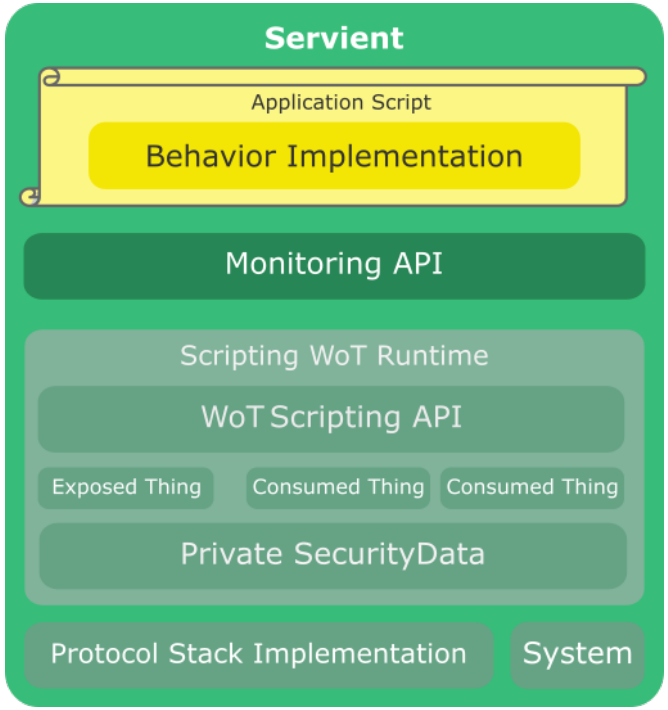
Presenta modifiche al runtime servient per fornire all'optimizer dati in tempo reale sulle performance del sistema

Strato di API di monitoraggio

Esegue la policy di allocazione per i servient.

Migration

Riceve il piano di deployment dell'optimizer e implementa le procedure di handoff per i servient.



M-WOT - servient

In particolare

Il nuovo strato di API di monitoraggio è incaricato di intercettare le invocazione all'api di scripting e di generare periodicamente dei *Thing Report* (Trs).

Thing Report (TR)

Snapshot dell'esecuzione corrente del servient che contengono i valori delle metriche del servient richieste dall'optimizer

M-WOT - esempio di migrazione I

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 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

Successivamente

L'optimizer viene eseguito: una nuova allocazione viene prodotta dove T_A deve essere spostato sul nodo N_2.

M-WOT - esempio di migrazione II

È necessario migrare T_A da N_1 ad N_2

1. L'esecuzione corrente di T_A viene stoppata dall'orchestrator
2. Il contesto applicativo di T_A è salvato come metadati all'intendo di TDir
3. L'orchestrator invia una richiesta al migration substrate per far muovere T_A/S_A al nodo di destinazione (N_2)

M-WOT - esempio di migrazione III

Dopo che S_A è stato rigenerato su N_2

1. Registra la sua nuova TD (con gli indirizzi di rete delle sue affordances aggiornate) nel TDir
2. Interroga la TDir per recuperare il contesto della T_A, quest'ultimo viene deserializzato e iniettato come oggetto globale nello script applicativo della T_A
3. T_A inizia il processo di inizializzazione e si espone facendo partire la registrazione della propria TD all'interno della TDir
4. T_A riprende nello stato in cui era prima di essere stoppata ed è considerata completamente migrata.

M-WOT - esempio di migrazione IV

Dopo che S_A è stato rigenerato su N_2

1. Registra la sua nuova TD (con gli indirizzi di rete delle sue affordances aggiornate) nel TDir
2. Interroga la TDir per recuperare il contesto della T_A, quest'ultimo viene deserializzato e iniettato come oggetto globale nello script applicativo della T_A
3. T_A inizia il processo di inizializzazione e si espone facendo partire la registrazione della propria TD all'interno della TDir
4. T_A riprende nello stato in cui era prima di essere stoppata ed è considerata completamente migrata.

M-WOT - esempio di migrazione V

Infine

1. La TDir invia una notifica a T_B riguardo la procedura di handoff.
2. T_B recupera la nuova TD di T_A dalla TDir e la consuma di nuovo per poter puntare alla posizione del servizio aggiornata
3. T_B ricomincia ad interagire con T_A e ad accedere alle sue affordances.

Migration Policy - scopo

Scopo

Vogliamo rappresentare formalmente le operazioni dell'optimizer come problema di ottimo multi obbiettivo

Come?

Consideriamo un processo di ottimizzazione a due passaggi che considera:

- il problema del load balancing (ovvero quanto carico ha ogni host)
- l'overhead delle comunicazioni di rete (ovvero quanti dati vengono scambiati fra gli host)

Migration Policy - formulazione del problema I

Scenario

Consideriamo un generico deployment WoT con N_{WT} WT attive.

Il sistema evolve nel tempo

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

Il sistema comprende un insieme di WT

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).

Migration Policy - formulazione del problema II

Le WT offrono una serie di affordances

Senza perdere in generalità poniamo

$$A_i = \{a_i^1, a_i^2, ...a_i^{N_{A_i}}\}$$

Ovvero le affordances esposte dalla wt_i nella sua TD, ogni affordance può rappresentare una proprietà, un'azione oppure un evento.

Le affordances sono un insieme statico

Presumiamo che A_i sia statico, ovvero wt_i non può aggiornare la propria TD a runtime (es. non può definire nuove proprietà)

Migration Policy - formulazione del problema III

Il sistema comprende un insieme di host

Poniamo H l'insieme dei nodi di host.

$$H = \{h_1, h_2, ...h_{N_H}\}$$

e assumiamo che i nodi siano eterogenei in termini di hardware.

Gli host sono eterogenei fra loro

Senza perdere in generalità modelliamo la diversa potenza di calcolo 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

Migration Policy - formulazione del problema IV

L'allocazione dei servient agli host

È definita dalla funzione policy

$$P : WT \times T \rightarrow H;$$

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.

Migration Policy - formulazione del problema V

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

$$PT_{m,k} = \{wt_i \in WT | P(wt_i, t_k) = h_m\}$$

.

Migration Policy - formulazione del problema VI

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

$$R_{i,j,k} = \{r^1_{i,j,k}, r^2_{i,j,k} \cdots\}$$

alla wt_j consumata.

L'implementazione di ogni richiesta $r^y_{i,j,k}$ comporta dello scambio di dati fra le WT wt_i e wt_j

definiamo

$$B(r^y_{i,j,k})$$

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.

Migration Policy - formulazione del problema VII

Carico totale

Denotiamo con

$$B(i,j,k) = \sum B(r^{y}_{i,j,k}) \; \forall r^{y}_{i,j,k} \in R_{i,j,k}$$

il carico totale di comunicazione fra le WT wt_i e wt_j all'istante di tempo t_k

Migration Policy - formulazione del problema VIII

Reminder: l'obbiettivo dell'optimizer

L'obbiettivo dell'optimizer è determinare la policy che calcola - ad ogni istante di tempo t_k- il trade-off ottimale fra

- la località dei dati (ovvero quanti dati sono trasferiti fra gli host)
- l'utilizzo di risorse computazionali (ovvero il load balancing fra gli host)

Per questo scopo

Introduciamo due metriche:

1. *NO*
2. *HF*

Migration Policy - formulazione del problema IX

La metrica *HF*

Metrica definita come la differenza fra il nodo più carico ed il nodo meno carico del cluster

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

$L(h_m, t_k)$

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:

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

Migration Policy - formulazione del problema X

Per formalizzare l'appartenenza delle WT agli host

Definiamo $p^{t_k}_{wt_i,h_m}$ 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:

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

Migration Policy - formulazione del problema XI

Formulazione finale

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

$$\min_{p^{t_k}_{wt_i,h_m}} \; NO(t_k)$$

Tale che;

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

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

Migration Policy - note sui vincoli

Primo vincolo

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

Serve per assicurarsi che l'allocazione su ogni host non ecceda le capacità computazionali per quell'host ($\gamma(h_m)$).

Secondo vincolo

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

Δ è un parametro definito dall'utente che quantifica il trade-off menzionato in precedenza fra *NO* ed *HF*.

Migration Policy - Legame fra *NO* ed *HF*

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 allo stesso host.
Tuttavia questo costituisce il caso peggiore per l'HF.
Per questo motivo abbiamo due scenari estremi, $\Delta = \infty$ e $\Delta = 1$

$\Delta = \infty$

L'obbiettivo del sistema è quello di minimizzare lo scambio di dati sulla rete, a prescindere dalla latenza del servizio

$\Delta = 1$

L'obbiettivo del sistema è quello di minimizzare la latenza del servizio, evitando la presenza di bottleneck di performance.

L'euristica

L'euristica proposta

Proponiamo un'euristica basata su grafi che rispetta il vincolo $HF(t_k) \leq \Delta$, mentre rilassa il vincolo $L(h_m, t_k) \leq 1 \ \forall h_m \in H$ e utilizza un approccio greedy per la funzione obbiettivo.

L'Euristica proposta - il grafo I

La soluzione si basa sulla costruzione di un grafo delle dipendenze

$$G(V, E, W, L)$$

- V è l'insieme dei vertici, ogni vertice rappresenta una WT, quindi $V = WT$ e $v_i = wt_i, \forall wt_i \in WT$
- E è l'insieme degli archi, ogni arco $e_l(v_i, v_j)$ connette due vertici $v_i, v_j \in V$ e modella le interazioni fra le due WT.
  Più specificamente esiste l'arco $e_l(v_i, v_j)$ se $B(i, j, k) > 0$ oppure se $B(j, i, k) > 0$.
- $W : E \to \mathcal{R}$ è una funzione peso, che assegna un costo ad ogni arco $e_l(v_i, v_j) \in E$. Qui, il valore $W(e_l(v_i, v_j))$ quantifica la quantità totale di dati scambiati fra le WT, in caso wt_i stia consumando wt_j o viceversa, ovvero $W(e_l(v_i, v_j)) = B(i, j, k) + B(j, i, k)$.

L'Euristica proposta - il grafo II

La soluzione si basa sulla costruzione di un grafo delle dipendenze

- $L : V \rightarrow \mathcal{R}$ è una funzione di carico che assegna un costo ad ogni vertice $v \in V$.
  presumiamo $L(v_i) = 1 \forall v_i \in V$, ovvero presumiamo che tutte le WT producano lo ste sso carico, mentre il carico totale dell'host h_m è il numero delle WT hostate.

L'Euristica proposta - il ragionamento I

1. Calcoliamo l'insieme delle componenti connesse nel grafo delle dipendenze G
2. Ordiniamo le componenti del grafo in base al loro valore di carico e le associamo agli host con un algoritmo round robin
3. In caso il vincolo sulla load fairness è rispettato ($HF(t_k) \leq \Delta$), l'algoritmo termina la sua esecuzione.
4. Altrimenti rompiamo le componenti calcolate fino ad ora migrando iterativamente una WT dall'host più carico all'host meno utilizzato, fino a quando il sulla load fairness è rispettato.

L'Euristica proposta - il ragionamento II

Come viene scelta la *wt* da migrare?

Con un approccio greedy come quella che minimizza l'overhead di rete, calcolato come la differenza fra:

- Il nuovo overhead generato quando stacchiamo wt_i dal suo host di partenza
- Il guadagno di performance sull'host di destinazione, causato dal fatto che wt_i è diventato un servizio locale per quell'host