Alma Mater Studiorum
Università di Bologna

Scuola di ingegneria e architettura
Corso di Laurea in Ingegneria Elettronica, Informatica e Telecomunicazioni

COORDINATION nel CLOUD: ELASTICITA' in RESPECT

Elaborata nel corso di: Sistemi Distribuiti

Tesi di Laurea di:
Francesco Serafini

Relatore:
Prof. ANDREA OMICINI

Co-relatori:
STEFANO MARIANI

III Sessione
Anno Accademico 2014-2015

Indice

Introduzione

La necessità di avere alte prestazioni tramite applicazione parallele in ambienti aperti e distribuiti ha portato allo sviluppo del concetto di coordinazione. Mentre la necessità, nei sistemi cloud, di gestire risorse e processi limitati offerti ad un numero illimitato di utenti ha portato allo sviluppo del concetto di elasticità.

Obiettivo di questa tesi è quello di mostrare come questi due concetti nati per scopi diversi abbiano in realtà diversi punti di contatto e di come è possibile vedere un sistema coordinato come un particolare tipo di sistema elastico e viceversa.

Per farlo questa tesi sarà così organizzata:

- Nel primo capitolo "Concetti introduttivi" approfondiremo quelli che sono i due concetti chiave di questa tesi, ovverosi la coordinazione e l'elasticità, vedendo brevemente in che contesti sono nati e quale lo scopo che si prefiggevano

- Nel secondo capitolo "Elasticità consapevole della coordinazione" analizzeremo due modelli che hanno sviluppato i concetti di coordinazione ed elasticità ovvero TuCSoN e SYBL e come questi abbiano delle similitudini tali che sia possibile pensare ad un sistema che sia

contemporaneamente sia elastico che coordinato e come sia possibile realizzarlo.

- Nel terzo capitolo “Estensione al Bridge rSYBL-ReSpecT” vedremo in che modo è stato esteso il modulo di rSYBL che permette di realizzare l’elasticità consapevole della coordinazione.
- Nell’appendice vedremo le sezioni di codice più rilevanti.

Capitolo 1

Concetti introduttivi

In questo primo capitolo introduciamo brevemente quelli che sono i due concetti chiave di questa tesi, ovvero la coordinazione e l'elasticità. Vedremo in che contesto sono nati e quali obiettivi si volevano perseguire. Ci soffermeremo maggiormente sul concetto di elasticità.

1 Coordinazione

La coordinazione [1] nasce nei sistemi chiusi con lo scopo di creare applicazioni parallele ed ottimizzare la velocità di elaborazione e di calcolo. Uno dei primi esempi è il modello LINDA dove era noto fin dall'inizio quali entità sarebbero state coinvonte, e anche il medium di coordinazione che permetteva la coordinazione tra le varie entità era legato all'applicazione e alla struttura sottostante. Con l'aumentare della complessità di calcolo, l'eterogeneità dei sistemi, la realizzazione del medium di coordinazione è diventata sempre più di primaria importanza tanto che da un'idea di coordinazione come linguaggio si è passati all'idea di coordinazione come servizio.

1.1 Modello di coordinazione

Un sistema che adotta un modello di coordinazione viene chiamato sistema coordinato, composto da diverse tecnologie ed entità che cooperano per un obiettivo comune. Questi modelli possono essere descritti tramite:

- le entità di coordinazione, ovverosia gli "agenti" che verranno coinvolti nei processi di coordinazione
- i medium di coordinazione, tutti quei componenti che rendono possibili le interazioni tra gli agenti
- le leggi di coordinazione che descrivono in che modo gli agenti possono coordinarsi attraverso e tramite i medium

1.2 Architettura logica

In questi modelli sono presenti tre spazi distinti che rappresentano l'architettura logica del sistema.

spazio coordinato: In questo spazio sono presenti tutte le entità coordinate del sistema, ognuna è indipendente e non conosce le altre ed è utilizzato dalle attività di coordinazione per tener traccia degli agenti che hanno fatto delle richieste e che devono ricevere delle risposte. Ciascuna entità può fare delle richieste di coordinazione, ricevere risposte di coordinazione o compiere calcoli interni

spazio di interazione : In questo spazio vengono gestite le richieste e le risposte delle entità coordinate. Rende possibile lo scambio e la comunicazione tra lo spazio coordinato e lo spazio di coordinazione creando eventi di comunicazione (richieste e risposte) che verranno consumati dai coordinabili. E' composto da un insieme di medium di coordinazione visti come astrazione che trasportano attività di coordinazione

Spazio di coordinazione: Lo spazio dove vengono effettuati il consumo di eventi di richiesta e vengono prodotti gli eventi di risposta. Questi eventi vengono consumati da mezzi di coordinazione che si trovano in questo spazio solo se corrispondono a certe condizioni. Avvenuto questo consumo il mezzo di coordinazione può generare nuovi eventi che si materializzano nello spazio di interazione

1.3 ReSpecT

ReSpecT è stato originariamente concepito per superare i limiti dei tradizionali linguaggi basati su spazio di tuple come Linda e che permette di implementare nuove primitive, modificare la semantica delle stesse e modificare le leggi di coordinazione esistenti. Tutto questo in fase di esecuzione.

E' possible comunicare, tramite agenti o righe di comando, con il centro di tuple con le primitive out_s, rd_s, in_s,rdp_s,inp_s,no_s, nop_s, get_s, set_s che hanno un perfetto matching con le funzionalità delle corrispondenti primitive tucson che vedremo in seguito. Queste primitive inseriscono o tolgono tuple strutturate in questo modo *reaction(Event, Guards, Body)*:

Events (eventi) Rappresenta tutto quello che può succedere all'interno del sistema coordinato o una qualunque primitiva TuCSoN

Guards (guardie) Le condizioni che devono valere sull'evento quando si innesca la reazione ed eventualmente attivare un Body. Le condizioni possono essere:

'invocation' se ci troviamo in fase di invocazione

'completion' se ci troviamo in fase di completamento

'success' se la primitiva ha avuto successo

'failure' la primitiva è fallita

'endo' la causa dell'evento è il centro di tuple corrente

'exo' la causa dell'evento non è il centro di tuple corrente

'intra' il bersaglio dell'operazione è il corrente centro di tuple

'inter' il bersaglio dell'operazione non è il corrente centro di tuple

'from_agent' la sorgente dell'evento è un agente

'from_tc' la sorgente dell'evento è un centro di tuple

Body computazioni da intraprendere in risposta a eventi se e solo se valgono le guardie. Può essere una qualunque primitiva TuCSoN, o una computazione in Prolog

2 Elasticità

Nei sistemi cloud [2] per poter fornire processi automatici occorre che l'approvvigionamento e le richieste delle risorse avvengano in maniera dinamica e che nel contempo sia garantita una certa qualità del servizio.

Il concetto di elasticità raccoglie una delle essenze della computazione cloud, ovvero che quando risorse limitate sono disponibili per potenzialmente un uso illimitato chi le fornise deve poterle maneggiarle in maniera 'elastica' provvedendo a fornirle quando necessario. I processi che stanno alla base di questo devono quindi diventare elastici. Pensare di vedere questo problema solo da parte del gestore delle risorse è restrittivo. Le richiesta delle risorse non sono infatti più determinate solo dalle applicazioni che le usano. I processi elastici pertanto arricchiscono le proprietà dei processi computazionali in constesti di cloud computing. Le principali propietà per modellare l'economia dei processi elastici e le dinamiche fisiche sono le risorse (sia hardware che software), i costi e la qualità del servizio.

2.1 Costi

I costi dell'elasticità indicano come cambia l'approvvigionamento delle risorse quando cambia il loro costo. I fornitori di servizi definiscono un modello di prezzo per l'erogazione di servizi cloud. In questo contesto l'elasticità si riferisce anche all'utility computing, dove le risorse così come i servizi computazionali sono approvvigionati da macchine virtuali, i dati vengono trasmessi sul network e i servizi di archiviazione dati sono forniti su diverse gerarchie di archiviazione.
Nel definire un modello di prezzi per l'utility computing occorre che a tempo di progettazione si considerino anche i costi sostenuti per mantenere la capacità di calcolo che includono l'acquisto, l'approvvigionamento, e il mantenimento di processori, memorie, hard disk e il network, la frequenza di clock desiderata per il sistema se questo è virtuale, la quantità di memoria, la dimensione dei dischi, e il costo delle trasmissione dati. Sulla base di questo i fornitori possono sviluppare modelli di prezzi dinamici basati sui concetti di costi dell'elasticità.
In Amazon Cloud questa cosa già esiste e può avvenire in due modi distinti:

- su istanze a richiesta, ovvero un modello pay per use on-demand dove i clienti sono liberi di pianificare i cambiamenti e chiedere un aumento del servizio aumentando i costi, o viceversa, diminuire il servizio abbassando i costi

- su istanze occasionali (spot) che avvengono quando si verificano variazione dei costi dei servizi nel tempo, in base allo stato di domanda e offerta o di altri fattori considerati dai servizi cloud di Amazon. In quest'ultimo caso gli utenti stabiliscono un prezzo massimo che sono disposti a pagare per usufruire di quei servizi e farli funzionare fino a quando il prezzo imposto da Amazon è pari o minore di quello del-

l'offerta, o fino a quando l'istanza è terminata, o il prezzo cresce al di sopra di quanto l'utente è disposto ad offire

2.2 Qualità

La qualità dell'elasticità misura quanto è sensibile la qualità al cambiamento nell'uso delle risorse. Il problema principale è quello di disporre di una funzione misurabile che calcoli la richiesta di risorse per mantenere una determinata qualità del servizio per esempio il mantenimento di una determinata velocità di esecuzione. In questo caso, il risultato di un servizio è possibile determinarlo a priori e si ottieni con qualunque velocità nell'escuzione ma questa varia in base alle risorse disponibili. Il tempo di risposta non è l'unico criterio di qualità utilizzabile. Altra possibilità è la qualità dei risultati. Per esempio una query su archivi molto grandi in grado di fornire risposte esatte potrebbe richiedere un tempo elevato e ad esse sono preferibili risposte più veloci e approssimative dato che non sempre quelle esatte sono necessarie.

2.3 Modello concettuale

Un possibile modello concettuale deve tenere conto di:

proprietà fisiche dell'elasticità : un Processo Elastico deve essere in grado di decidere come utilizzare le risorse esistenti traendone il massimo vantaggio. L'ambiente, le risorse (computazionali, dati e risorse di rete) i loro modelli di qualità e di costo risultano essere dinamici. Sulla base di qualità e costi, un processo elastico potrebbe o utilizzare diverse risorse e attività di trasformazione o risorse simili per ottenere risultati diversi. Questo comportamento riflette le proprietà fische dell'elasticità

proprietà economiche dell'elasticità : in primo luogo, bisogna considerare i costi del processo elastico e distinguerli delle risorse per la costruzione

del processo elastico (una macchina computazionale, anche virtuale, le risorse di rete o servizi software). I fornitori del servizio cloud rendono le risorse disponibili, e ogni risorsa ha tra le sue proprietà anche qualità e costo. Tra le caratteristiche delle proprietà economiche dell'elasticità ci sono risorse, costi, qualità ed elasticità. Un processo elastico utilizza risorse fornite da qualsiasi fornitore in qualsiasi luogo e in qualunque momento, purché soddisfino i vincoli richiesti dai processi come i costi minimi di spesa. Un processo elastico è in grado di considerare diversi modelli di costi e presentarli al consumatore.

principi di Funzionamento e modellazione : un processo elastico deve essere in grado di

- monitorare, gestire e descrivere le prorietà dinamiche
- sulla base della qualità potersi raffinare dinamicamente
- determinare il costo migliore tra più modelli di costi delle risorse
- fornire elasticità anche prendendo risorse tra più fornitori

Nel caso più semplice, il processo elastico è utilizzato da un unico consumatore e utilizza un unico fornitore. Nel caso più estremo, un processo elastico serve N consumatori, che imporrebbero un certo numero di vincoli, e utilizza M fornitori Un processo elastico deve modellare la sua funzione come una proprietà statica. I risultati dei processi elastici si basano sui requisiti di costi e qualità che diventano vincoli da rispettare. Questo modello influenza le risorse dell'elasticità. Inoltre, la modellazione può anche descrivere come un processo elastico comunica con eventuali altri processi. E tutto questo può essere applicato anche su più livelli.

2.4 Cosa occorre

Per poter ottenere questo tipo di elasticità occorre:

poter specificare vincoli e preferenze : si vuole definire un'insieme di processi con vincoli e preferenze a livello di costi, qualità e risorse impiegate e si vorrebbe poter controllare il comportamento del sistema con un linguaggio o un'interfaccia intuitiva

auto descrizione delle risorse : le risorse hanno delle proprietà e devono essere in grado di renderle comprensibili a qualunque processo elastico che richieda di conoscere le proprietà altrui.
Ogni risorsa deve fare i conti con un elevato grado di eterogeneità per descrivere sé stessa dato che in contesti elastici potrebbe essere richiesta da chiunque. E' possibile pensare alla creazione di diversi livelli di dettaglio a seconda delle esigenze, e alcune informazioni saranno facoltative, ma la descrizione deve essere comprensibili a chiunque

meccanismo di ragionamento elastico : un processo elastico deve essere dotato di un meccanismo di ragionamento elastico per decidere in che modo utilizzare le risorse nella maniera ottimale. Questo meccanismo può essere considerato come un sistema di ottimizzazione che ottiene risorse e informazioni sui costi dall'ambiente

riusabilità ed adattamento dell'esecuzioni : esistono già dei processi di adattamento ma non considerano combinazioni tra risorse, costi e qualità. Le tecniche di raffinamento esistenti si concentrano sulla qualità delle prestazioni, ma non sulla qualità del risultato. L'esecuzione dei processi non può essere lenta o addirittura statica. Deve concentrarsi sul monitoraggio continuo e sulla ri-pianificazione. In questi grandi ambienti complessi sono necessari approcci decisionali approssimativi basati sulla

probabilità e su informazioni parziali. I processi elastici consentono l'esecuzione di processi di adattamento e possono reagire ai cambiamenti dell'ambiente

un formalismo per i sistemi di processi elastici : un formalismo potrebbe contribuire alla modellazione e alla comprensione dei processi elastici. Un sistema di questo tipo deve essere costruito su un insieme ben definito di operatori. Questi operatori dovrebbero concentrarsi sulle caratteristiche dei processi elastici e sulla loro composizione

Capitolo 2

Elasticità consapevole della coordinazione

In questo secondo capitolo parleremo brevemente di TuCSoN, come modello di coordinazione, e in maniera più articolata di SYBL, come modello di elasticità. Inoltre, a partire dalle similitudini tra i due modelli, introdurremmo anche il concetto di Elasticità Consapevole della Coordinazione e di come è possibile crearne un modello a partire da TuCSoN e SYBL.

1 TuCSoN

TuCSon (Tuple Centres Spread over the Network) è un modello e una tecnologia [3] che permette la coordinazione di processi distribuiti e di agenti autonomi, intelligenti e mobili. Si basa sull'astrazione chiamata "centro di tuple" uno spazio programmabile tramite tuple, scritte con un linguaggio specifico il ReSpecT (Reaction Specification Tuples), che descrivono leggi specifiche di coordinazione, distribuite sui nodi di una data rete. Le entità principali di questo sistema sono:

nodi TuCSoN : rappresentano l'astrazione topologica che contiene il centro di tuple. Sono identificati univocamente tramite la coppia (NetworkId, PortNo) dove NewtorkId rappresenta l'indirizzo IP o il DNS del device che ospita il nodo e PortNo è il numero della porte su cui risiede il servizio di coordinazione (di base la porta è la 20504). La sintassi per identificare un nodo TucSoN è *'netid:portno'* .

centri di tuple ReSpecT : sono il medium di coordinazione, forniscono lo spazio condiviso per la comunicazione e lo spazio di comportamento programmabile per la coordinazione. Programmabili tramite ReSpecT possono reagire a determinati eventi. Viene identificato univocamente da *'tname @ netid:portno'* dove tname è il nome del centro di tuple.

Gli Agenti TuCSoN : sono le entità coordinate, sono attività pro-attive ed intelligenti che attraverso primitive TuCSoN possono inviare e ricevere tuple e dare o togliere a quest'ultime determinati comportamenti. Quando entrano in un sistema TuCSoN gli viene assegnato un identificativo (UUID). Il suo nome completo sarà *'aname : uuid'* dove aname è il nome dell'agente.

Tucson fornisce anche un linguaggio di coordinazione definito da primitive di coordinazione tramite le quali gli agenti possono interagire con i centri di tuple. Ogni operazione è divisa in due fasi, quella di *invocazione* (invio della richiesta con tutte le informazioni necessarie) e quella di *completamento* (il risultato viene restituito dal centro di tuple all'agente che l'aveva chiesto). La sintassi per eseguire un operazione su un dato centro di tuple è *tcid ? op* dove tcid è l'identificativo di un centro di tuple.

Le operazioni base ammissibili sono:

out (Tuple) inserisce una tupla nello spazio di tuple specificato.

rd (TupleTemplate) ricerca una tupla nello spazio di tuple che corrisponda col template. Se ha successo il risultato è una tupla, altrimenti l'operazione resta in sospeso fino al suo compimento

in (TupleTemplate) stesso funzionamento della rd, solo che una volta trovata una tupla la rimuove dal centro di tuple

rdp (TupleTemplate) stesso funzionamento della rd, ma se trova una tupla che corrisponda al template fornito restitusce quest'ultima altrimenti l'operazione fallisce, non è quindi un'operazione bloccante

inp (TupleTemplate) stesso funsionamento della rdp, ma se l'operazione ha successo rimuove dal centro di tuple la tupla trovata, non è quindi un'operazione bloccante

no (TupleTemplate) verifica l'esistenza o meno di una tupla nel centro di tuple. Finchè ne sarà presente anche solo una l'operazione viene sospesa

nop (TupleTemplate) stesso funzionamento di no (TupleTemplate) con la differenza che se è presente una tupla che corrisponde al template fornito l'operazione fallisce

get restituisce tutte le tuple presenti nello spazio di tuple

set(Tuples) inserisce tutte le tuple passate in ingresso nel centro di tuple

2 Sybl

2.1 Direttive di programmazione

Le direttive di programmazione [4] sono sempre più in uso in linguaggi di programmazione in parallelo. Permettono agli sviluppatori di controllare il parallelismo separando la logica della programmazione del comportamento

computazionale. Le principali direttive di programmazione controllano tra le altre cose il numero di processori usati e la distribuzione dei dati. L'elasticità come abbiamo detto non è limitata solo alle risorse ma anche ai costi e alla qualità quindi è possibile pensare di espandere delle direttive in modo che coinvolgano anche costi e qualità.
Nei sistemi cloud esistono degli strumenti e linguaggi specifici che sviluppano e configurano le applicazioni, tuttavia non includono vincoli riguardanti elasticità. E' possibile pensare di applicare i principi delle direttive di programmazione per gestire l'elasticità intrinseca degli scenari di computazione cloud nei quali è necessario controllare che vengano rispettati alcuni vincoli riguardanti risorse, costi e qualità. Le direttive di programmazione per l'elasticità dovrebbero permeterre agli sviluppatori di specificare proprietà a tempo di esecuzione relative a risorse, costi e qualità tramite

monitoring: un monitoraggio continuo che controlla lo stato corrente della computazione, determinando le risorse usate, i costi maturati, la qualità del servizio e informandone l'utilizzatore

constraints: alcuni vincoli che specificano le condizioni che devono valere per quanto riguarda risorse, costi e qualità. Queste direttive dipendono dalle informazioni fornite dalle direttive di monitoraggio e possono abilitare strategie da eseguire qualora queste condizioni valgano o che queste siano violate

strategies: strategie specifiche che consentono agli sviluppatori di esprimere azioni per gestire il comportamenti del programma sulla base dei dati monitorati e dei vincoli che devono essere rispettati, influenzando l'ambiente di elaborazione ed eseguendo eventuali operazioni. Un gestore di strategia può attivare strategie o su richiesta (qualora venga chiesto al sistema di compiere una determinata operazione indipendente dai

vincoli o dal monitoraggio che si sta eseguendo) o come risposta a un cambiamento al valore di un vincolo

Possiamo quindi creare 3 classi di direttive di programmazione dette MONITORING, CONSTRAINT e STRATEGIES e in ognuna di esse gli sviluppatori possono specificare alcune proprietà, come è possibile vedere nella figura 2.1, usando o direttive predefinite o definite a tempo di esecuzione. SYBL (Simple-Yet-Beautiful Leanguage) nasce con lo scopo di specificare le principali direttive per le applicazioni cloud. rSYBL è l'interprete in grado di eseguire a runtime le funzioni SYBL.

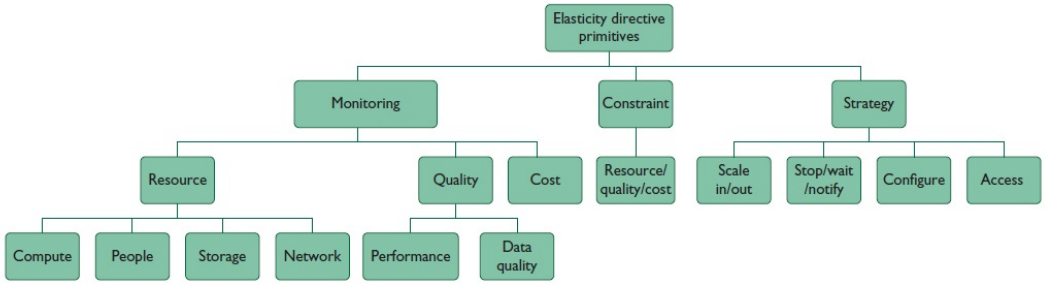

Figura 2.1: Una prima gerarchia che mostra le tipologie di direttive e le richieste che possono essere fatte per ciascuna di esse

2.2 Direttive

SYBL definisce un'insieme di funzioni runtime che possono gestire e controllare le proprietà dell'ambiente di computazione. SYBL è composto da un insieme di direttive che gli sviluppatori possono usare o singolarmente o in combinazione tra di loro per eseguire compiti più complessi. In maniera astratta le direttive SYBL cominciano con *#SYBL* seguite dal nome della classe di direttive (MONITORING, CONSTRAINT e STRATEGY) e dalle diverse clausole di direttiva che permettono di usare funzioni runtime e defi-

nite dagli utenti con altre variabili, funzioni e clausole. Possiamo utilizzare queste primitive di elasticità sia in linguaggi di programmazione general-purpose sia in quelli di configurazione di sistema per controllare che vengano utilizzate risorse computazionali e finanziarie, per raggiungere la qualità desiderata o per specificare in che modo configuare il sistema in un ambiente elastico. Nei linguaggi general-purpose è possibile inserire le direttive nel supporto di annotazioni specifico di quel linguaggio.

3 Integrazione

3.1 Motivazioni

I controllori di elasticità [5] responsabili di eseguire le direttive di elasticità, sono direttamente responsabili del supporto e dell'applicazioen dell'elasticità nel cloud, tramite il monitoraggio e il controllo dei cambiamenti che avvengono nei costi, qualità e risorse e delle dipendenze che intercorrono tra di loro in ambienti potenzialmente vasti ed eterogenei. Per compiere queste operazioni occorre che siano coordinate. I controllori di elasticità per come sono attualmente implementati realizzano una coordinazione tramite processi goffa, ingombranti e soggetta ad errori sia da parte dello sviluppatore che dai componenti software infatti:

- i meccanismi di esecuzione necessitano di parametri (per esempio il tempo di risposta, il numero di tentativi prima di riuscire a svolgere una determinata funzione) che spesso sono hard-coded o configurabili solo a tempo di progettazione

- la sincronizzazione tra questi meccanismi è carente (non si sa se un'operazione ha avuto successo o meno, tempi di attesa indefiniti o hard-coded, ecc...)

- il flusso di controllo passa dal controllore di elasticità alle API cloud di più basso livello, esponendo il tutto a fallimenti o errori, ostacolando la portabilità e il riuso del codice

Flessibilità, sicurezza, incapsulamento e separazione delle responsabilità sono ostacolati perché i controllori di elasticità si scontrano con alcune problematiche legate alla coordinazione. Un'integrazione tra elasticità e coordinazione può essere fatta su tre diversi livelli:

- a livello di meta-modello (concettuale) definendo un'astrazione da usare mentre si pensa alla soluzione del problema che abbiamo davanti (nel nostro caso fornendo una elasticità consapevole della coordinazione)
- a livello di modello (linguaggio) definendo in che modo la soluzione al problema può essere definita a partire dall'astrazione fornita del meta-modello (tramite direttive elastiche e leggi di coordinazione)
- a livello di tecnologia (infrastruttura) definendo in che modo le astrazioni e il linguaggio possono essere supportati da una nuova architettura e a tempo di esecuzione (tramite il controllore di elasticità integrato con un medium di coordinazione)

L'elasticità nei sistemi cloud sta diventando una questione sempre più pressante, tanto che per agevolarla i fornitori dei servizi cloud hanno sviluppato architetture specifiche per poterla implementare. Anche la coordinazione appare necessaria nella programmazione orientata agli agenti e in generale in ogni tipo di programmazione che preveda una coordinazione temporale o spaziale. Si potrebbero aprire scenari in cui agenti debbano operare su sistemi cloud elastici o di sistemi elastici che abbiano bisogno di coordinazione. Per potere pensare di integrare insieme elasticità e coordinazione dobbiamo conciliare alcune nozioni di questi due mondi. In questo modo avremmo diversi benefici:

- un supporto a tempo di esecuzione per la delega e la separazione delle responsabilità che permetterebbe ai componenti di concentrarsi su un aspetto specifico della computazione delegando l'altro a chi è più competente
- aumentare la sicurezza nelle interazioni tra i controllori di elasticità e i componenti, servizi o plugin cloud tramite primitive e leggi di coordinazione ben definite
- aumentare l'availability dei controllori di elasticità delegando ad un componente dedicato tutte le operazioni relative alla coordinazione
- agevolare il processo di sviluppo, tramite la separazione di doveri e responsabilità tra sviluppatori di elasticità e di coordinazione

3.2 Integrazione

Per garantire l'elasticità i sistemi cloud devono essere:

- composti da servizi elastici e soggeti a controllori dell'elasticità programmati per il sistema
- supportati da un'infrastruttura che abbia le capacità di monitorare e controllare le risorse computazionali
- dotati di un linguaggio di programmazione per le direttive di elasticità da applicare a tempo di esecuzione

Per garantire la coordinazione un sistema deve essere:

- composto da agenti (coordinabili) soggetti a leggi di coordinazione.
- supportato da una infrastruttura che ospiti il medium di coordinazione

- dotato di un linguaggio che permetta di specificare nuove leggi di coordinazione a tempo di esecuzione

E' possibile fare un primo confronto tra le astrazione di elasticità e coordinazione

$$\text{Elasticità} \longleftrightarrow \text{Coordinazione}$$
$$\text{Risorse/Servizi Elastici} \longleftrightarrow \text{Agenti (Coordinabili)}$$
$$\text{Infrastruttura elastica} \longleftrightarrow \text{Media di Coordinazione}$$
$$\text{Direttive di Programmazione} \longleftrightarrow \text{Leggi di Coordinazione}$$

In astratto un sistema elastico può essere visto come un sistema coordinato in cui i servizi elastici e le risorse computazionali sono entità soggette a processi di coordinazione (sono quindi coordinabili). Questi processi sono supportati da un'infrastruttura elastica e di coordinazione composta da un'insieme di medium di coordinazione e componenti elastici distribuiti. Questi ultimi sono responsabili dell'applicazione a tempo di esecuzione e della programmabilità delle direttive di coordinazione elastiche desiderate.

3.3 Linguaggio e interoperabilità

E' possibile individuare alcune similitudini tra le direttive di programmazione elastiche e le leggi di coordinazione. Infatti le direttive di programmazione sono generalmente composte da primitive di monitoraggio, vincoli e strategie che descrivono in che modo i servizi e le risorse dovrebbero comportarsi, mentre le leggi di coordinazione descrivono in che modo dovrebbero comportarsi i coordinabili. Nelle leggi di coordinazione è necessario essere in grado di osservare lo stato del sistema, le dinamiche di interazione dello spazio ed eseguire operazioni su questo spazio in modo da modificare, qualora fosse opportuno, lo stato e le dinamiche stesse.

Si può arrivare a questo tipo matching:

Elasticità $\longleftrightarrow$ Coordinazione

Direttive $\longleftrightarrow$ Leggi

Funzioni runtime $\longleftrightarrow$ Primitive di coordinazione

Monitoring $\longleftrightarrow$ Event

Constraint $\longleftrightarrow$ Observation

Strategy $\longleftrightarrow$ Computation

In particolare

Monitoring: Le primitive di monitoraggio dell'elasticità sono progettate per osservare lo stato e le dinamiche di un sistema elastico tramite le proprietà rese osservabili dall'infrastruttura di supporto, i servizi ospiti e le risorse computazionali, un processo che risulta continuo. Gli eventi coordinati invece non sono processi continui, ma eventi singolari catturati dal medium di coordinazione e lì memorizzati.
L'obiettivo di entrambi risulta similare ovvero abilitare il sistema a reagire ad un cambiamento.

Constraint: Le primitive di vincolo controllano se sussistono o meno certe condizioni riguardo allo stato del sistema. Si basano sulle proprietà osservate dalle primitive di monitoraggio e attivano all'occorrenza le primitive di strategia. Sono simili all'osservazione nella coordinazione che verifica che le leggi che si sono innescate possano essere abilitate. Anche i constraint sono controllati in maniera continua mentre le guardie (responsabili dell'osservazioni) una volta che si verificano innescano una determinata reazione (evento singolare), ma l'obiettivo è,in entrambi i casi, attivare l'esecuzione della computazione in seguito a dati eventi.

Strategy: Le primitive di strategia compiono azioni atte a modificare lo stato e le dinamiche del sistema elastico influenzando l'esecuzione dei

servizi, le proprietà delle risorse e il comportamento dell'infrastruttura. Ovviamente questo avviene in risposta a vincoli violati o a condizioni rese valide, controllate in fase di monitoraggio. C'è una forte similitudine anche qui con le leggi di coordinazione che abilita il medium di coordinazione a compiere determinate azioni se sono stati osservati determinati comportamenti.

Usando i linguaggi SYBL e ReSpecT è possibile quindi fare queste similitudini tra i concetti chiave dei due linguaggi:

$$\text{Direttive} \longleftrightarrow \text{Tuple di specifica}$$
$$\text{Funzioni runtime} \longleftrightarrow \text{Primitive}$$
$$\text{MONITORING} \longleftrightarrow \text{Event}$$
$$\text{CONSTRAINT} \longleftrightarrow \text{Guards}$$
$$\text{STRATEGY} \longleftrightarrow \text{Body}$$

3.4 Separazione delle responsabilità

Non basta solo integrare i linguaggi per avere un pieno supporto ed ottenere un'elasticità consapevole della coordinazione.
Si potrebbe pensare di rendere ciascuna direttiva di programmazione di elasticità una legge di coordinazione in grado di interfacciarsi con le API cloud sottostanti. Oppure di fare l'inverso, ovvero rendere le leggi di coordinazione direttive di elasticità in grado di chiamare le API cluod. Oppure di codificarle separatamente e in seguito tradurle in un linguaggio comune di esecuzione in grado di integrare le API dei due linguaggi e di interfacciarsi con le API cloud. Ognuna di queste soluzione comprende delle problematiche e in nessun caso ci si occupa di separazione delle responsabilità.

Meglio pensare ad uno scenario diverso in cui ogni sviluppatore realizzi la propria applicazione in maniera indipendente. Occorre realizzare un compilatore che prende il codice di applicazione e le direttive di programmazione e

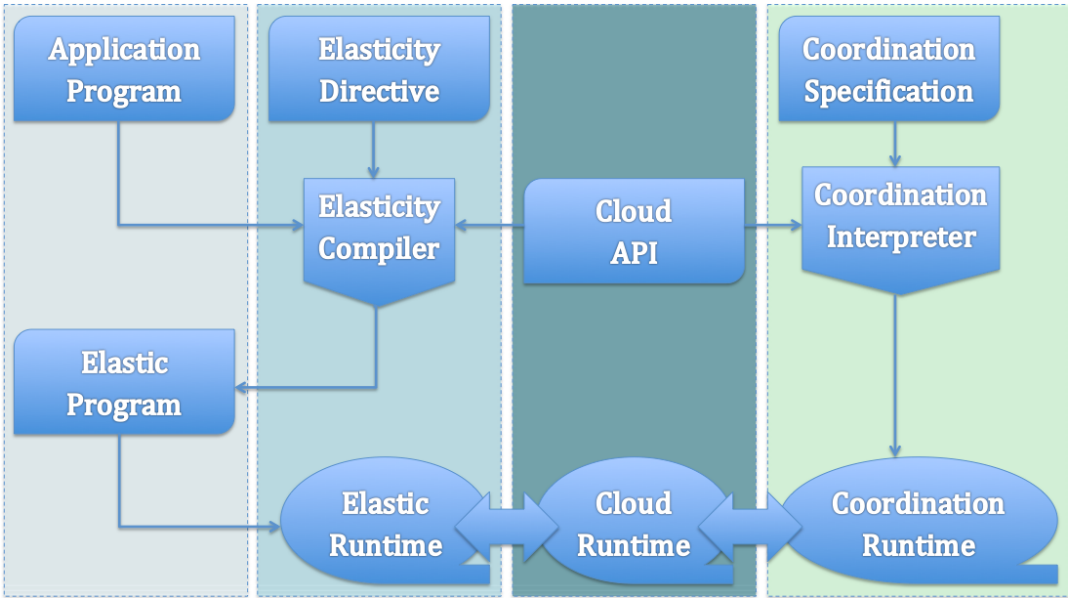

Figura 2.2: Il flusso di sviluppo ed esecuzione di un'applicazione elastica consapevole della coordinazione in grado di eseguire direttive di programmazione elastica in sinergia con le leggi di coordinazione

le integra in un programma elastico che sappia interfacciarsi con le API fornite dall'infrastruttura cloud sottostante e che infine il programma generato possa essere eseguito sull'infrastruttura cloud stessa.

In questo modo non occorre pensare e capire in che modo le direttive di elasticità possano essere tradotte in leggi di coordinazione o viceversa. E' necessaria solo un API di integrazione che permetta alle direttive di elasticità di interagire con le leggi di coordinazione e che rendano possibile lo scambio di infromazioni tra le due parti.

3.5 Linee guida di integrazione

Monitoring API : Il monitoraggio può essere eseguito sia dalle primitive di monitoraggio dell'elasticità, sia dalle leggi di coordinazione. Le primiti-

ve di monitoraggio sono state create proprio con questo scopo. Tuttavia qualora il sistema sia stato pensato come un sistema coordinato con capacità elastiche, e che quindi i monitoraggi più frequenti riguardino la coordinazione, in tal caso le leggi di coordinazione risultano più efficaci. In accordo col principio di separazione delle responsabilità questo aspetto potrebbe essere coperto sia dalle primitive di monitoraggio che dalle leggi di coordinazione e solo quando questi devono scambiarsi informazioni o delegare la computazione allora si deve trovare il modo di integrarle secondo lo schema visto in precedenza. Si può facilmente risolvere questo problema creando tuple ben definiti che deleghino la richiesta e quindi pensare ad un processo elastico che davanti ad un problema di coordinazione deleghi tramite a delle tuple specifiche questo problema ad un agente. E, viceversa, un agente che davanti ad un problema legato all'elasticità deleghi, sempre tramite apposite tuple, il compito ad un processo elastico.

Constraint API: I vincoli verificano quando a seguito di alcuni eventi sussistano o meno certe condizioni. Se questi vincoli sono generati da eventi che devono essere risolti da leggi di coordinazione questi dovrebbero essere tradotti in tuple ben definite che inserite nel medium di coordinazione attivino le leggi di coordinazione. Se il monitoraggio già avviene tramite leggi di coordinazione non occorre fare altro. Altrimenti occore tradurre le direttive di programmazione elastica in qualche modo. L'unico problema che sussiste a questo livello è che le API di integrazione dei vincoli devono avere le capacità di chiamare servizi di coordinazione e di costruire nel caso le apposite tuple.

Strategy API : Qui avvengono la maggior parte delle deleghe a tempo di esecuzione tra elasticità e coordinazione. Anche il compito più semplice relativo all'elastità implica per la coordinazione un problema non bana-

le che può diventare enorme se non si è progettato il sistema con il giusto livello di astrazione. L'esecuzione di strategie avviene generalmente nelle direttive di elasticità, in risposta a vincoli verificati o violati, a seguito di un monitoraggio. Le strategie di elasticità possono delegare compiti di coordinazione a leggi di coordinazione oppurtanamente create che possono a loro volta fare richieste di tipo elastico per richiedere proprietà relative all'elasticità o esecuzione. Come per il monitoraggio, in accordo con la separazione delle responsabilità, si potrebbe pensare, data la complessità, di dividere la responsabilità di applicazione delle strategie tra elasticità e coordinazione. All'occorrenza:

- il controllore di elasticità trasforma una richiesta di coordinazione in una tupla e la inserisce in un medium di coordinazione. Qui si innescano le leggi di coordinazione che avviano il processo desiderato e, una volta terminato, forniscono un risultato (in forma di tupla) al controllore di elasticità per indicare il successo o il fallimento dell'operazione
- Il medium di coordinazione interagisce con il controllore di elasticità che esegue la sua operazione e restituisce un risultato in forma di tupla

Capitolo 3

Estensione al Bridge rSYBL-ReSpecT

In questo terzo capitolo vedremo come è stato possibile lavorare su rSYBL, quali problematiche sono state riscontrate per poterlo avviare e come queste sono state risolte. Dopodiché vedremo quali criteri stavano dietro alla realizzazione del modulo bridge-rsybl-respect già presente in uno dei branch di sviluppo parallelo di SYBL e come questo è stato esteso.

1 Avvio del sistema

Si è dedido di sfruttare una possibilità di rSYBL, ovvero quello di poter essere eseguito in modalità dry-run in grado di simulare la presenza di un server cloud, verificando che le funzioni e le modifiche apportate al codice di SYBL vadano a chiamare le API del servizio cloud sottostante. SYBL infatti fornisce un supporto per la gestione dell'elasticità, ma la riuscita o meno dell'operazione è delegata alla API del servizio cloud. Dal momento che questa modalità era compatibile con l'ambiente Unix si è scelto di lavorare su macchina virtuale in ambiente Linux. Come ambiente di sviluppo si è scelto

Eclipse-EE per poter utilizzare le librerie di supporto ottenibili tramite l'integrazione Maven. Il codice di rSYBL è possibile ottenerlo da un repository remoto git pubblico [6]. Una volta scaricato tramite download diretto o tramite il plugin git di Eclipse si è proceduto a importare come progetto Maven tutto il progetto SYBL riscontrando due problematiche principali legate a Maven stesso:

- il non corretto download di alcune librerie Maven per il quale si è risolto cancellando le cartelle nascoste .m2 contenenti gli artefatti Maven già scaricati e probabilmente corrotti e forzando l'update del progetto tramite click col tastro destro del mouse sul progetto rSYBL selezionando Maven -> Update Project e nel pop-up spuntando l'opzione "Force updates"

- l'errore Maven "Plugin execution not covered by lifecycle configuration" risolto includendo i tag <plugins> <plugins> di alcuni file pom.xml all'interno dei tag <pluginManagement></pluginManagement>

Per poter inizializzare rSYBL occorrono:

- alcuni file .xml contenenti la descrizione di alcuni servizi che saranno poi simulati dal cloud da inserire all'interno del modulo "rSYBL/starting rSYBL/rSYBL Python clients/singleEnforcementMechanism/" sotituendo gli originali

- uno script python chiamato "lifecycle.py" che inizializza i servizi rsybl una volta avviato il sistema su un cloud.

- un file chiamato "config.properties" da inserire in "rSYBL/rSYBL-control-service-pom/rSYBL-analysis-engine/src/main/resources/" con la configurazione del sistema. In particolare si sono modificate le voci riguardandi le API da andare a chiamare imponendo "EnforcementPlugin = at.ac.tuwien.dsg.rSybl.cloudInteractionUnit.api.EnforcementAPI"

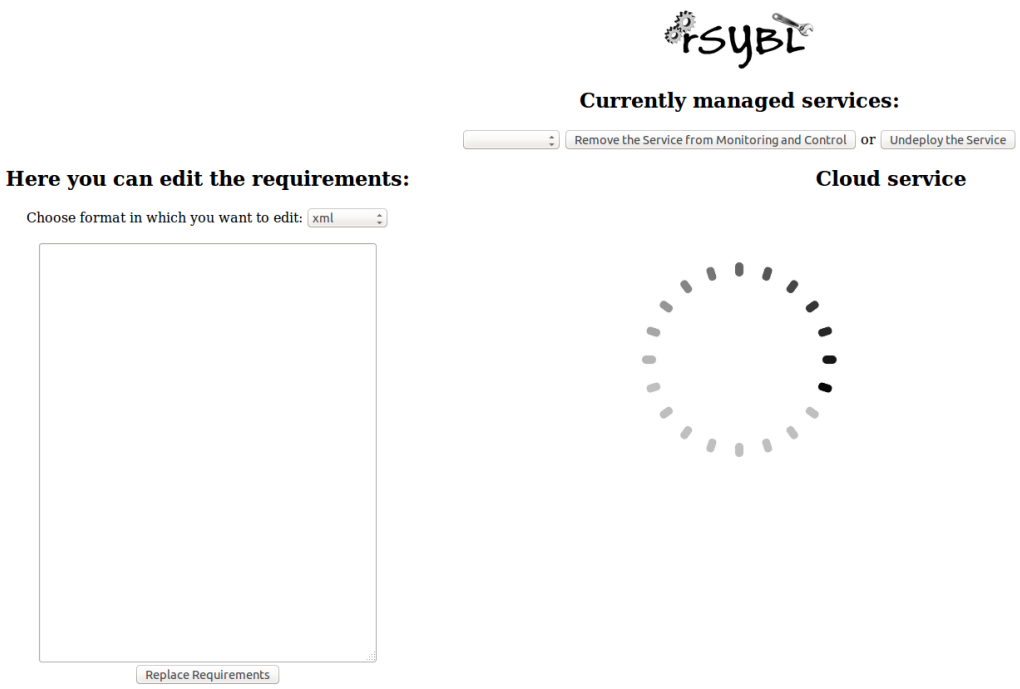

Figura 3.1: Schermata di rSybl vuota

In questo modo rSYBL sa che le API da andare a chiamare si trovano nella classe EnforcementAPI

A questo punto per lanciare rSybl è bastato installare un server Tomcat e avviare su quest'ultimo il modulo "rSYBL-analysis-engine". Eseguendolo si apre, a seconda delle impostazioni date ad Eclipse, una pagina browser (interna o esterna) come quella che si vede nella figura 3.1. A questo punto per abilitare il servizio basta eseguire lo script lifecycle.py. Nella schermata di rSybl è presente un menù a tendina in alto a sinistra con i quali è possibile selezionare uno tra i servizi attivi (nel nostro caso solo CloudService) e andare a modificare nel box a sinistra in linguaggio xml, o SYBL, le specifiche dei singoli elementi che compongono il servizio, andando a specificare nostre direttive, applicabili con il pulsante, presente sotto il box "Replace

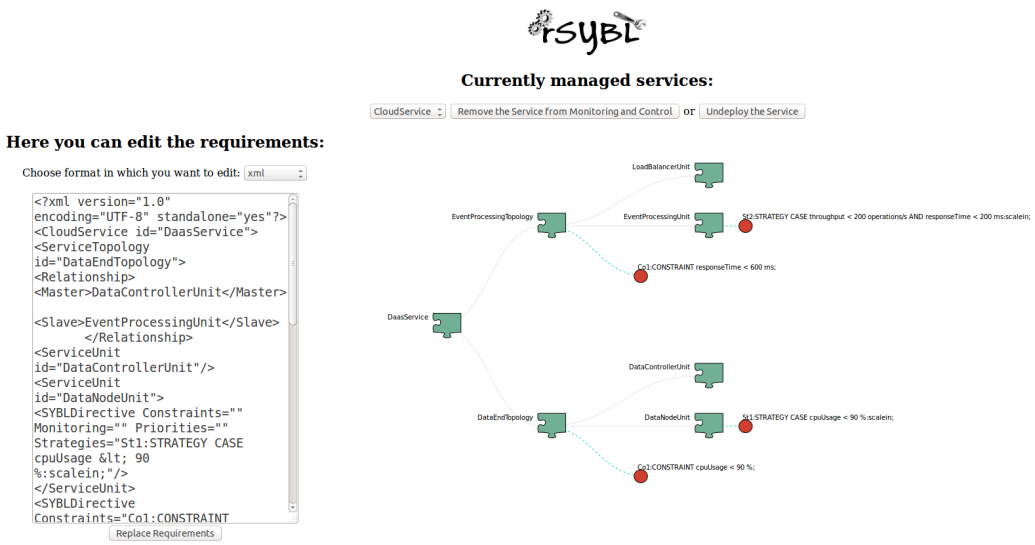

Figura 3.2: Schermata di rSybl una volta avviato un servizio tramite lifecycle.py

Requirements" come si vede nella figura 3.2.

Per espandere il modulo respect-sybl-respect precedentemente creato si è scaricato quel modulo dal branch di sviluppo di rSybl "coordination". A questo punto si è creato un nuovo modulo Maven (si poteva eventualmente importarlo) settando come parametri del id del progetto padre "at.ac.tuwien.rSYBL.control-service". Dato che il modulo era stato creato con una versione precedente di rSybl si sono dovuti aggiornare alcuni artefatti Maven modificando il file "pom.xml" contenuto nel modulo fornendogli la nuova versione del codice. Tra gli artefatti Maven era presente anche una vecchia versione di TuCSoN che si è aggiornato inserendo l'ultima versione.

Risolti i problemi Maven si è passato a risolvere i problemi legati alla compilazione Java. La maggior parte era dovuto all'importazione di classi o rinominate o in una posizione diversa rispetto alla versione precedente di SYBL o di TuCSon con cui il modulo era stato scritto. In particolare l'impor-

tazione delle classi Configuration e RunTimeLogger contenute in precedenza in cloudApiLowLevel → EnforcementPlugin → opestack spostate nel modulo dataProcessingUni → utils. Diversi altri errori invece erano dovuti alla generazione di possibili eccezioni non gestite. Semplicemente dove veniva segnalato che si poteva generare un'eccezione questa la si è catturata con un try/catch.

2 Integrazione API

Nell'articolo "Coordination-aware Elasticity" [5] si suggeriva l'ipotesi di fare l'integrazione a livello di API in questo modo:

- *monitoring API* possibile in due modi. Un'attivià spawned che aspetta di ricevere richieste di monitoraggio in forma di tupla ReSpecT che va ad abilitare il processo di monitoraggio chiamando le API di rSYBL o una specifica ReSpecT che permetete la configurazione runtime del processo di monirotaggio su specfici istanze delle tuple creatore dall'attività spawned.

- *constraint API* Un componente ponte che abilita il controllore di elasticità rSYBL a chiamare le primitive di coordinazione ReSpecT e i servizi TuCSoN per avviare o terminare servizi di coordinazione.

- *strategy API* Si può fare un discorso analogo alle API di Monitoraggio.

3 Estensione

3.1 Visione generale

Il modulo ponte contiene, oltre a delle specifiche ReSpecT e alcuni agenti (uno per effettuare la scalein e uno per la scaleout) , la classe RespectEn-

forcementAPI responsabile del collegamento tra i due mondi. Questa classe contiene infatti come unico metodo pubblico il metodo delegate che permette al controllore di elasticità rSYBL di chiamare una primitiva di coordinazione. Questo metodo verrà utilizzato nella classe EnforcementAPI.

Si è deciso di estendere il modulo in modo tale che oltre all'operazione di scalout, già presente, fosse in grado di compiere anche l'operazione inversa, una scalein. Per fare ciò si è:

- realizzato una specifica ReSpect per l'adattamento a tempo di esecuzione chiamata *"scalein.rsp"*

- si è realizzata la classe *"SyblScaleInAgent"* responsabile dell'interfacciamento tra ReSpecT e rSYBL nel caso in cui avvenga una scalein. Prima di delegare a rSYBL il compito di eseguire la scalein dal centro di tuple l'agente scopre quanti tentativi deve fare prima e l'intervallo tra i tentativi prima di dichiarare il fallimento dell'operazione e una volta eseguito il metodo scalein l'agente informa l'utente della riuscita o meno dell'operazione

- si è modificta la classe *EnforcementAPI*, presente nel modulo "rSYBL-cloud-interaction-unit" dove è possibile sviluppare nuove funzionalità per rSYBL. Si è modificato in particolare enforceAction() in modo tale che, qualora venga invocata una scalein, al suo posto venga invocato il metodo privato realizzato per l'occasione "doCoordinatedScaleIn". In questo modo si permette ad una operazione di elasticità consapevole della coordinazione di agire al posto di una direttiva SYBL. EnforcementAPI riesce ad invocare le specifiche Tucson tramite la classe RespectEnforcementAPI.

Il codice verrà analizzato nell'Appendice.

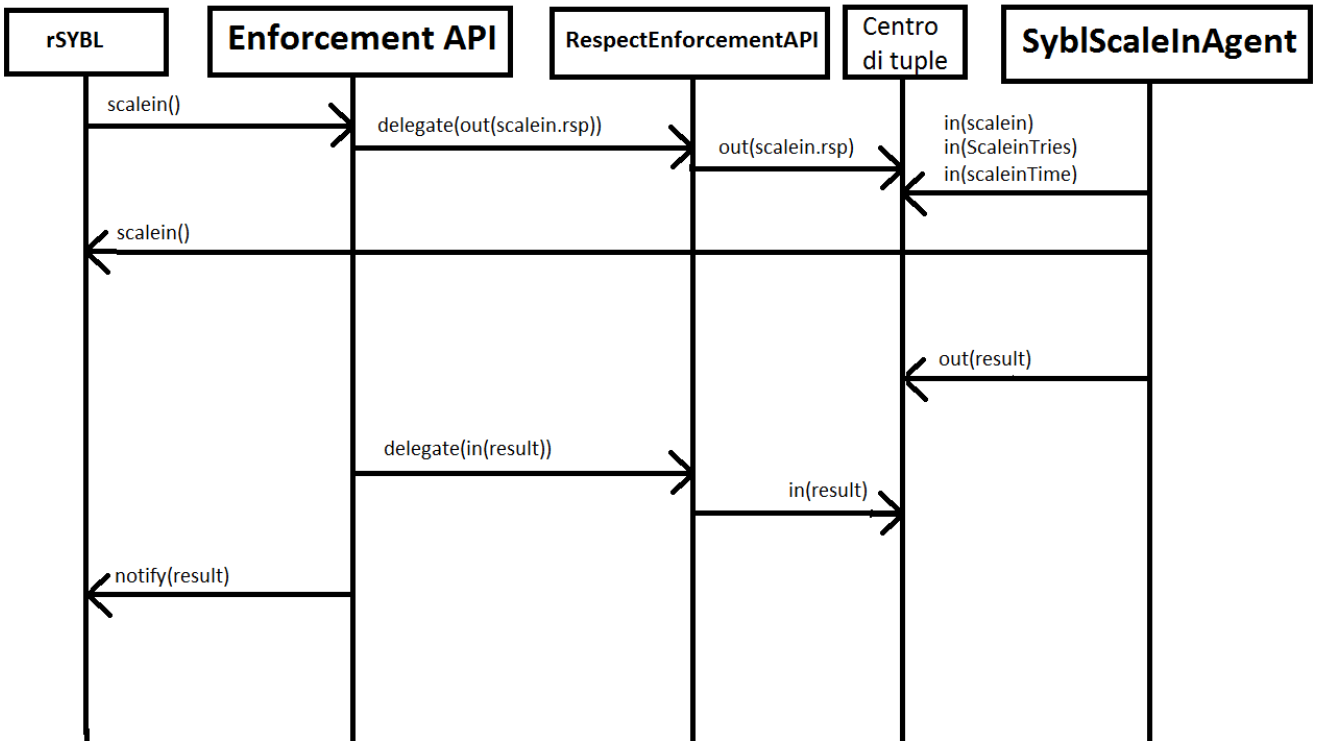

Figura 3.3: Diagramma di flusso

3.2 Funzionamento di scalein

Ogni volta che viene invocata una scalein (da un utente in maniera forzata, o da rSYBL in risposta ad uno dei vincoli monitorati)

1. enforcementiAPI intercetta l'invocazione di scalein ed invoca il metodo "doCcordinatedScaleIn"

2. enforcementiAPI tramite la funzione "delegate" della classe RespectEnforcementAPI inserisce la specifica ReSpecT "scalein.rsp" nel centro di tuple

3. scalein.rsp fa in modo che quando viene invocata una scalein inserisce nel centro di tuple due nuove tuple con il numero di tentativi che devono

essere fatti per eseguire l'operazione e l'intervallo tra questi.

4. enforcementiAPI inizializza un SyblScaleInAgent (che andrà ad eseguire una scalein in maniera coordinata)

5. enforcementiAPI cede il compito di monitorare all'agente tramite il metodo "doCoordinatedMonitoring" e rimane in attesa di leggere il risultato dell'operazione

6. syblScaleInAgent legge dal centro di tuple se è stata invocata una scalein, e in caso affermativo cerca le tuple con gli altri dati necessari per l'operazione (numero di tentativi, e intervallo tra questi)

7. syblScaleInAgent invoca una scalein su rSYBL e lo farà per un certo numero di volte a intervalli regolari finché l'operazione non avrà successo o finché non saranno esauriti i tentativi

8. syblScaleInAgent inserisce nel centro di tuple una nuova tupla contenente il risultato dell'operazione

9. enforcementAPI una volta ottenuto il risultato dell'operazione informerà rSYBL dell'esito

3.3 Testing

Tramite tre figure vediamo cosa succede quando viene chiamata prima una scaleout e, a seguito di questa, una scalein. Per fare questo abbiamo bisogno, oltre a quanto già visto per inizializzare rSYBL, anche di una cartella "load" contenente diversi file .csv, uno per ogni elemento che costituisce il nostro sistema, che descrivono l'andamento in termini di costi, cpusage,e altre fattori dei singoli servizi nel tempo. rSYBL conosce la posizione della cartella poichè nel file "config.properties" abbiamo imposto che "MonitoringPlugin =

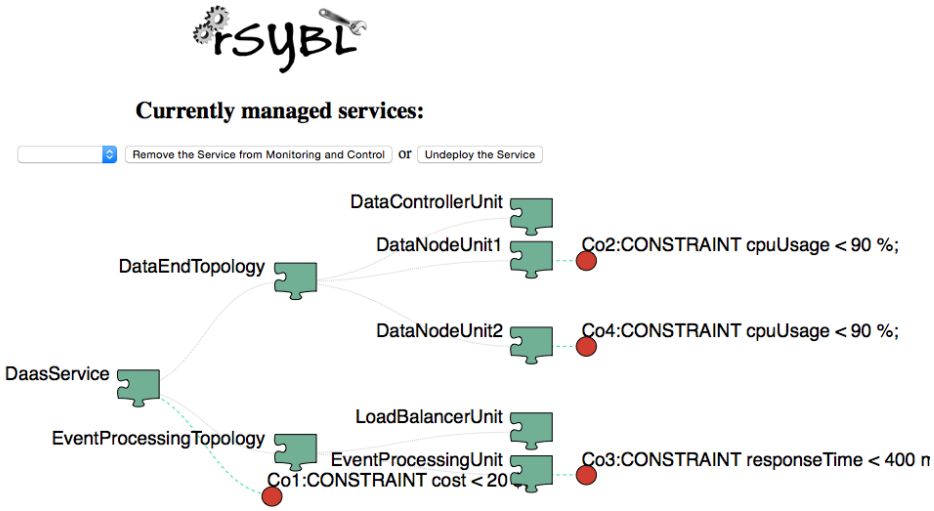

Figura 3.4: A seguito della prima scaleout

at.ac.tuwien.dsg.rSybl.dataProcessingUnit.monitoringPlugins.replay.MonitoringAPILoadData" e in MonitoringAPILoadData è segnata la posizione che di base è "rSYBL-analysis-engine/DeployedResources/webapp/", ma è possibile inserire un percorso assoluto.

Tra questi file .csv è presente DataNodeUnit.csv, modificato in modo tale che l'uso della cpu (quindi lungo tutta la colonna cpuUsage) in un primo momento abbia un valore superiore al 90% e successivamente inferiore.

A questo punto si è avviato il sistema(visibile nella figura 3.2), e imposta all'elemento DateNodeUnit le direttive SYBL <SYBLDirective Constraints = "Co1 : CONSTRAINT cpuUsage < 90 %" Strategies = "St1: scaleout;"/>. Dato che inizialmente l'uso della cpu è superiore al 90% il vincolo viene violato e per risposta si attiva la scaleout. Nelle figure 3.4 e 3.5 vediamo infatti che aumentano i DataNodeUnit, passando da 1 a 3. A questo punto si è modificata, a tempo di esecuzione, la direttiva per i DataNodeUnit

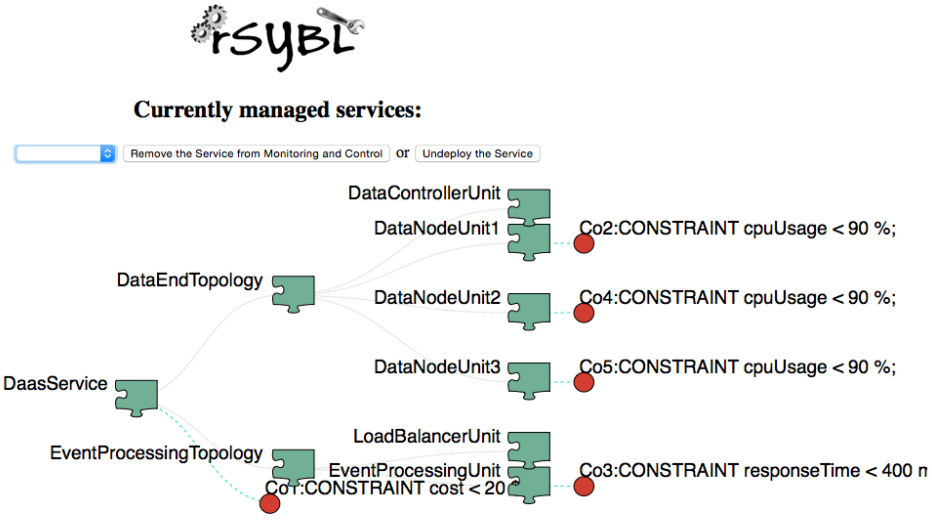

Figura 3.5: A seguito della seconda scaleout

in <SYBLDirective Constraints="Co1 : CONSTRAINT cpuUsage > 90 %" Strategies=" St1 : scalein;"/>. In questo modo quando l'uso della cpu torna inferiore al 90% si attuano una serie di scalein che portano al risultato che vediamo in figura 3.6, con un solo DataNodeUnit.

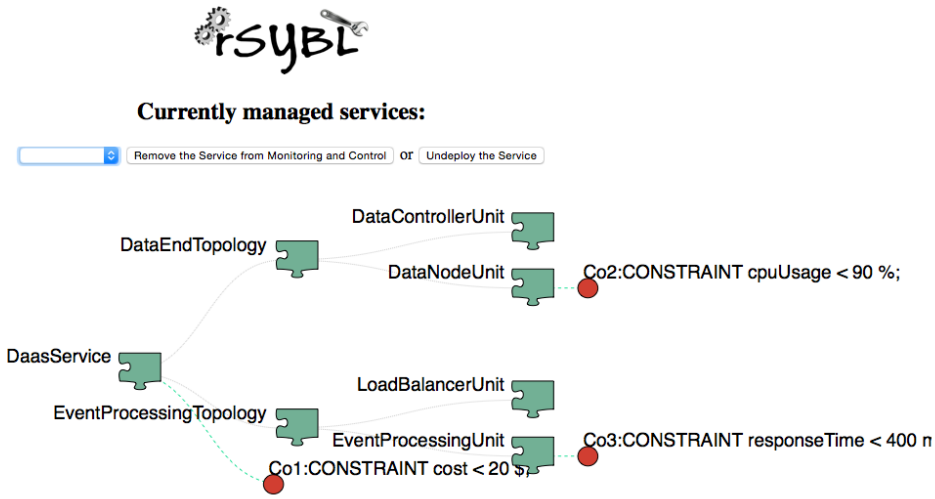

Figura 3.6: A seguito della doppia scalein

Conclusioni

Il cloud si sta sempre più espandendo e diversi fornitori già permettono di gestire le loro applicazioni in maniera elastica ma non in maniera così semplice cosa può esserlo tramite direttive di programmazione. Le direttive di programmazione di elasticità SYBL riescono ad interfacciarsi con l'API del cloud sottostante per permettere una gestione elastica dei sistemi con un linguaggio di più alto livello e più facilmente comprensibile.

In ambienti sempre più vasti e su più livelli l'elasticità rischia di scontrarsi sempre più con problematiche legate alla coordinazione e a dover risolvere questo tipo di problemi cosa che al momento non avviene.

Abbiamo visto che è possibile pensare ad un tipo di elasticità che sia nel contempo coordinato. Non solo, abbiamo anche visto che, grazie all'apporto di TuCSoN, è possibile realizzare un sistema elastico consapevole della coordinazione in grado di risolvere i problemi che SYBL da solo non riesce ad affrontare.
Abbiamo inoltre visto come è possibile estendere questo sistema realizzando una nuova funzione, la scalein, in modo tale che questa sia coordinata.

Tuttavia questa è solo una minima parte di quello che può essere realizzato. Sono tante le funzioni di SYBL che al momento non possono svolgersi

in maniera coordinata. Inoltre si è cercato di realizzare solo uno dei prossibili approcci all'elasticità consapevole della coordinazione, mentre altre strade percorribili potrebbero portare a risultati migliori.

Per il futuro si potrebbe pensare addirittura di partire da quanto è stato fatto per realizzare un'elasticità che sia intrinsecamente consapevole della coordinazione o una coordinazione che sia nel contempo elastica arrivando a vedere i due concetti strettamente legati tra loro.

Appendice

Scalein.rsp

```
reaction(
    out(doScaleIn(NodeId)),
    (operation, invocation),
    (
        out(scaleInTries(10)),
        out(scaleInTime(100))
    )
).
```

Questa reazione fa sì che ogni qualvolta venga invocata una operazione che inserisce nel centro di tuple una "doScaleIn" su un nodo vengano inserite nel centro di tuple il numero di tentativi che devono essere fatti e l'intervallo di tempo tra un tentativo e l'altro. In questo modo questi dati che dovrebbero essere hard-coded possono essere prelevati da apposite tuple di configurazione e quindi essere noti da chi le utilizza.

EnforcementAPI.java

```
public void enforceAction( String actionName,  Node e){
    else if (actionName.startsWith("scalein")) {
               if ("scalein".length() == actionName.length()) {
                   args = null;
               } else {
                   args = actionName.substring("scalein(".length(),
                           actionName.length() - 1);
               }
               this.doCoordinatedScaleIn(args, e);
```

EnforceAction() viene attivato ogniqualvolta viene chiamata una qualche operazione. in questo caso se viene riconosciuto come operazione una scalein, si va a chiamare l'operazione coordinata doCoordinatedScaleIn.

```
private void doCoordinatedScaleIn(String actionName, Node node){
       this.respect.delegate("set_s(" + Utils.fileToString
            RespectEnforcementAPI.MONITOR_SCALEIN_METRICS_PATH) + ")
            ",Long.MAX_VALUE, node};
```

In questa porzione di codice vediamo ennforceAction() delegare a RespectEnfocementAPI ad inserire la specifica scalein.rsp (che individua tramite la stringa RespectEnforcementAPI.MONITOR_SCALEIN_METRICS_PATH) nel centro di tuple.

```
    new SyblScaleInAgent(SyblScaleInAgent.AID,
    RespectEnforcementAPI.TUCSON_PORT, this.controlledService, node).go();
    this.respect.delegate("out(doScaleIn('"+ node.getId() + "'))",
```

```
            Long.MAX_VALUE, node);
    this.doCoordinatedMonitorMetrics(actionName, node);
```

In questa porzione di codice vediamo enforceAction() inizializzare l'agente che eseguirà la scalein e in seguito delega a ReSpecT il compito di monitorare.

```
    op = this.respect.delegate("in(scaleIn('" + node.getId()
            + "'), done(B))", RespectEnforcementAPI.OP_TIMEOUT, node);
            if (op.isResultSuccess()) {
RuntimeLogger.logger.info("Finished scaling in "
+ node.getId() + " : "
+ op.getLogicTupleResult().getArg(1).getArg(0));
} else {
RuntimeLogger.logger.info("Finished scaling in "
+ node.getId() + " failed!");
}
} else {
RuntimeLogger.logger.info(node);
}
```

Di nuovo delegando la richiesta a respectEnforcementAPI enforcercementAPI verifica se è presente una tupla nel centro di tuple (creata dall'SyblScaleInAgent) che lo informi della riuscita o meno dell'operazione di scalein. Sarà quindi sua premura una volta noto l'esito della scalein informare l'utente (in questo caso tramite un log).

SyblScaleInAgent.java

```
public class SyblScaleInAgent extends AbstractTucsonAgent{

     protected String scaleInAndWaitUntilNewServerBoots
     (final Node entity, final Node controller)
     throws InvalidVarNameException {

         // how many tries?
            ITucsonOperation op = this.acc.in(this.tid, new LogicTuple
            ("scaleInTries", new Var("Tries")), Long.MAX_VALUE);
            tries = op.getLogicTupleResult().getArg(0).intValue();
            // which time step between re-tries?
            op = this.acc.in(this.tid, new LogicTuple("scaleInTime",
             new Var("Time")), Long.MAX_VALUE);
            step = op.getLogicTupleResult().getArg(0).intValue();
```

SyblScaleInAgent è l'agente che realizza la scalein. Viene riportato solo il metodo che realizza la scalein. In questo segmento di codice vediamo che preleva dal centro di tuple il numero di tentativi e l'intervallo tra questi. Questi dati erano stati inseriti da scalein.rsp.

```
            this.acc.out(this.tid, new LogicTuple("scaleIn", new Value(
                    this.node.getId(), new Value("done", new Value(res)))),
                    Long.MAX_VALUE);
```

A seguito dell'operazione l'agente inserisce nel centro di tuple una tupla contenente il risultato dell'operazione che, come verrà letto dall'EnforcementAPI.

Bibliografia

[1] “Coordination as a Service” Mirko Viroli, Andrea Omicini

[2] “Principles of Elastic Processes” Schahram Dustdar, Yike Guo, Benjamin Satzger and Hong-Linh Truong.

[3] “The TuCSoN Coordination Model & Technology A Guide” Andrea Omicini, Stefano Mariani

[4] “Programming Directives for Elastic Computing”, Schahram Dustdar, Yike Guo, Rui Han, Benjamin Satzger and Hong-Linh Truong.

[5] “Coordination-aware Elasticity” Stefano Mariani, Hong-Linh Truong, Georgiana Copil, Andrea Omicini, Schahram Dustdar

[6] https://github.com/tuwiendsg/rSYBL

Ringraziamenti

Non tante, ma importanti sono le persone che voglio ringraziare.
In primo luogo i miei genitori e la mia famiglia, per i valori che mi hanno trasmesso e per il sostegno che mi han dato.
Il gruppo scout RN9, la Casa Giovani, e tutte le realtà che ho frequentato, perché mi hanno permesso di arrivare fin qui.
La mitica e unita III D che nonostante tutto riesce ancora a vedersi.
L'ufficio fundraising dell'APG23 che mi è sempre stato vicino in questi ultimi due anni.
E infine a te, lettore, che nonostante tutto sei arrivato a leggere fino a qui.