



UNIVERSITÀ DEGLI STUDI DI SALERNO
DIPARTIMENTO DI INFORMATICA

Tesi di Laurea di I livello
in
Informatica

DMASON on the cloud:
Amazon EC2

Relatore

Ch.mo Prof. Vittorio Scarano

Co-relatore

Dott. Carmine Spagnuolo

Candidato

Simone Bisogno

Anno Accademico 2016-2017

*Al Dott. Felice Russo
luminare, medico, uomo*

Sommario

La simulazione è una branca dell'informatica che permette lo studio di sistemi reali formulando dei modelli che rispecchino meglio la realtà rispetto a modelli probabilistici.

I modelli di simulazione ad agenti sono classi di modelli di calcolo per la simulazione di azioni e interazioni tra agenti autonomi che rispondono a regole prefissate.

Tra gli strumenti disponibili per l'esecuzione di modelli di simulazione ad agenti vi è MASON, toolkit *single-process* per le simulazioni multi-agente scritto in Java, sviluppato dalla George Mason University, il quale gode di portabilità e generalità dei modelli con esso sviluppati.

I modelli di simulazione sviluppati con MASON possono essere eseguiti su una sola macchina: ciò rende i tempi di esecuzione molto lunghi in caso di simulazioni particolarmente complesse; questo problema può essere risolto prevedendo la distribuzione della simulazione su più macchine.

I principî di progettazione di MASON l'hanno reso leggero e soprattutto espandibile: in questo modo, è possibile estenderlo per ottenere simulazioni distribuite; una delle sue estensioni è **DMASON**, che aggiunge uno strato di distribuzione della simulazione tra più processori logici (worker), i quali sono coordinati da una macchina principale (master) secondo il paradigma *master-worker*. Il controllo dello stato da parte del master avviene tramite una console di gestione chiamata **System Management**, dalla quale è possibile gestire e monitorare i worker inattivi e le simulazioni in esecuzione su un sistema distribuito eterogeneo.

Avere un framework per la distribuzione di modelli di simulazione non è sufficiente: serve allestire manualmente un cluster di computer per poterle eseguire; oltretutto, una soluzione *on premise* manca di scalabilità, rischiando di avere risorse di calcolo sature o inutilizzate.

In questo contesto si inserisce il *cloud computing*. Esso è un modello di erogazione dei servizi di calcolo distribuito più avanzato: i provider di tali servizi mettono a disposizione dell'utente finale le proprie risorse con diverse modalità architetturali: *Infrastructure as a Service*, *Platform as a Service*, *Software as a Service*.

L'azienda leader nel settore cloud è Amazon, che con Amazon AWS offre un ecosistema di numerosi servizi di cloud computing; in particolare, **Amazon EC2** permette la creazione di istanze di calcolo di caratteristiche variabili a seconda della necessità dell'utente e possono essere scalate, se necessario.

In questa tesi è presentata la realizzazione di un modulo che consenta l'esecuzione automatica di DMASON su cloud tramite il servizio Amazon EC2; tale architettura permette la gestione dinamica di istanze di calcolo sia

“on demand” (dove il prezzo è determinato in base al tempo di utilizzo delle risorse di calcolo) sia “spot” (dove il prezzo è determinato in base all’offerta per risorse di calcolo inutilizzate di EC2 messe all’asta).

Ciò è stato possibile aggiungendo una classe di servizio principale per l’interazione con Amazon EC2 grazie all’AWS SDK per Java e alcune classi per l’esecuzione di comandi remoti su connessione sicura e la gestione delle installazioni di DMASON sulle istanze create.

Il risultato è la semplificazione dell’esperienza utente: piuttosto che dover allestire a mano un cluster di computer, con pochi click si può far partire una simulazione su istanze remote di Amazon EC2 su cui sia automaticamente installato e avviato DMASON.

Ringraziamenti

Voglio innanzitutto ringraziare il mio relatore, il Prof. Vittorio Scarano per aver accettato la mia richiesta di tesi e il tutor da lui assegnatomi, il Dott. Carmine Spagnuolo.

Ringrazio la Dott.ssa Pina Palmieri e i ragazzi dell'ISISLab per l'accoglienza in un ambiente allegro, caloroso (non solo in senso letterale), gradevole; in particolare, ringrazio Flavio Serrapica, Michele Carillo e Matteo D'Auria per l'aiuto prestatomi e la grande pazienza avuta nei miei confronti (Flavio per l'aiuto con Maven e Git, Matteo per la prima revisione alla presente tesi e Michele per non aver inveito contro di me tutte le volte in cui ho fatto fallire il processo di costruzione con Maven).

Ringrazio i manutentori del sito `stackoverflow.com` – “*my only hope*” – per aver donato agli sviluppatori di tutto il mondo un posto in cui dare e ricevere supporto per i problemi di programmazione più ardui da risolvere.


Ringrazio tutti i colleghi universitari con cui ho avuto modo sia di studiare sia anche solo di parlare: avete dato colore alla mia carriera universitaria dopo anni di grigiore; in particolare, ringrazio Antonio Caiazzo (detto “TONCS”) per le sapienti indicazioni ma soprattutto per i passaggi in macchina da e per Fisciano.

Ringrazio i miei genitori per il tempo, il denaro, le energie spese per garantirmi un'istruzione dignitosa.

Ringrazio Carmela – conforto del mio passato, gioia del mio presente, speranza del mio futuro – per essere sempre rimasta al mio fianco “nella buona e nella cattiva sorte”.

Infine, ringrazio chi, dopo aver dedicato la propria vita al prossimo, con molta discrezione ha iniziato l'*ultimo sogno*, tornando a far parte dell'imponderabile.

Simone

Questa tesi è stata sviluppata in  **ISISLab**

Indice

I DMASON su cloud: Amazon EC2

1	Introduzione	1
1.1	Simulazione	1
1.1.1	Ambienti di simulazione multi-agente	2
1.1.2	MASON	2
1.1.3	DMASON	4
1.2	Cloud computing	5
1.2.1	Sviluppo e commercializzazione	7
1.2.2	Caratteristiche e modalità di servizio	7
1.2.3	Problemi e critiche	9
1.2.4	<i>Amazon Web Services</i> ed <i>Elastic Computing Cloud</i>	10
1.3	Obiettivi	12
2	DMASON in azione	13
2.1	Introduzione	13
2.2	Interfaccia grafica	13
2.3	Funzionamento	15
2.3.1	Costruzione	15
2.3.2	Esecuzione	18
2.4	Limitazioni	19
3	Sviluppo frontend	20
3.1	Tecnologie utilizzate	20
3.1.1	Polymer	21
3.1.2	Bower	21
3.1.3	jQuery	21
3.1.4	Masonry	22
3.1.5	jsRender	22
3.1.6	JavaServer Pages e JSP Standard Tag Library	22
3.2	Attività preliminare	23
3.3	Modifiche introdotte	25
3.3.1	Struttura attuale del progetto	25
3.3.2	Riuso	25

3.3.3	Template per i worker	26
3.3.4	Informazioni worker	27
3.3.5	Intervallo aggiornamento worker	27
3.3.6	Gestione impostazioni DMASON	27
4	Sviluppo backend	29
4.1	Tecnologie utilizzate	29
4.1.1	Apache Maven	30
4.1.2	Servlet	30
4.1.3	Apache Commons Configuration	30
4.1.4	JSch	30
4.2	Sviluppo	31
4.2.1	Struttura del progetto	31
4.2.2	Gestione impostazioni	31
4.2.3	Connessioni remote	33
4.2.4	Caricamento e scaricamento di file remoti	33
4.2.5	Gestione remota di DMASON	34
4.2.6	VersionChooser	35
5	Amazon EC2 in DMASON	37
5.1	Schema generale	37
5.2	Strumenti di sviluppo	37
5.2.1	<i>AWS SDK for Java</i>	38
5.2.2	<i>AWS Toolkit for Eclipse</i>	38
5.3	Amazon EC2	39
5.3.1	Tipi di istanze	39
5.3.2	<i>Amazon Machine Images</i>	40
5.3.3	Console di gestione	40
5.4	Integrazione EC2 in DMASON	43
5.4.1	Classe di servizio EC2Service	43
5.4.2	Modelli	48
5.4.3	Integrazione front-end	48
6	Conclusioni	49
6.1	Lavoro svolto	49
6.2	Sviluppi futuri	
II	Appendici	
A	Listati	i
A.1	Front-end	i
A.1.1	Esempio Frammento JSP: <code>header.jsp</code>	i
A.1.2	Aspetto dei worker	ii

A.1.3	Aggiornamento informazioni worker	iv
A.1.4	Scheda impostazioni <i>General</i>	v
A.1.5	Gestione impostazioni	vi
A.2	Back-end	viii
A.2.1	Comandi remoti	viii
A.2.2	Gestione richieste istanze EC2 da System Management	viii
A.2.3	Estrazione versione DMASON da modello di progetto di Maven	x
A.2.4	Classe di servizio <code>EC2Service</code>	xi

Bibliografia

xvi

Elenco delle figure

1.1	Paradigma <i>master-workers</i> in DMASON.	6
1.2	Diagramma CSNET	8
1.3	<i>Quadrante magico</i> di Gartner	11
2.1	Schema d'uso di DMASON	14
2.2	DMASON, vista <i>Monitoring</i>	16
2.3	DMASON, vista <i>Simulations</i>	17
2.4	DMASON, viste <i>History</i> e <i>Settings</i>	17
3.1	Struttura del progetto frontend	26
3.2	DMASON, nuova vista <i>Monitoring</i>	28
3.3	DMASON, nuova vista <i>Settings</i>	28
4.1	Struttura del progetto backend	31
5.1	Schema d'uso corrente di DMASON	38
5.2	Console principale Amazon EC2	41
5.3	Gruppi di sicurezza e coppie di chiavi nella console EC2	42

Elenco dei listati

2.1	DMASON, download con Git	15
2.2	DMASON, costruzione con Maven	15
2.3	DMASON, comando base per l'esecuzione	18
2.4	DMASON, esecuzione simultanea di più worker	18
3.1	Vecchie dipendenze System Management	23
3.2	Attuali dipendenze System Management	24
3.3	Installazione dipendenze (dal file <code>bower.json</code>)	24
3.4	Esempio d'uso dei frammenti nelle pagine JSP	25
3.5	Aggiornamento impostazioni	27
4.1	File di configurazione precedente	32
4.2	File di configurazione corrente	32
4.3	Creazione builder per le configurazioni	32
4.4	Le prime dieci righe del POM di DMASON	35
5.1	Corpo del metodo <code>buildEC2Client(String)</code>	44
5.2	Richiesta coppia di chiavi in <code>createKeyPair()</code>	45
5.3	Salvataggio su file della coppia di chiavi	45
5.4	Richiesta istanza EC2	46
A.1	Frammento dell' <i>header</i> di DMASON	i
A.2	Creazione dei worker senza template (da <code>script.js</code>)	ii
A.3	Creazione dei worker con template (da <code>script.js</code>)	iii
A.4	Template per i worker	iii
A.5	Funzione aggiornamento variabile dati worker	iv
A.6	Esempio di scheda di impostazioni	v
A.7	Funzione di caricamento impostazioni correnti	vi
A.8	Esempio di aggiornamento impostazioni (scheda <i>General</i>)	vii
A.9	Metodo per l'esecuzione di comandi remoti	viii
A.10	Metodo istanziazione istanze EC2 da servlet	viii
A.11	Classe <code>VersionChooser</code>	x
A.12	Metodo di primo avvio della classe <code>EC2Service</code>	xi
A.13	Metodo per la creazione di un'istanza EC2 on demand	xii
A.14	Metodo per l'avvio di un'istanza EC2 on demand	xiv

Prefazione

La scelta del mio argomento di tesi, il *cloud computing*, nasce grazie a incontri organizzati dal Prof. Vittorio Scarano tra le aziende che operano nel settore informatico e gli studenti del Dipartimento di Informatica dell'Università degli studi di Salerno, per portare ai primi forza lavoro altamente qualificata e ai secondi preziose esperienze nella ricerca e gestione del posto di lavoro.

In uno di questi seminarî – che vanno sotto il nome di “*Hello Jobs!*” – l’azienda ospite è stata *Amazon* insieme a *Reply S.p.A.*, suo partner in Italia; in quel momento fu mostrato dal presentatore di Amazon il rapporto “*Quadrante magico per un’infrastruttura cloud come servizio, Versione internazionale*” di Gartner, citato anche nella presente tesi: la lotta tra le grandi aziende informatiche per la conquista di fette di mercato nel settore che questo rapporto esibisce – congiuntamente all’esperienza di Reply in ambito di migrazione di sistemi *on premise* su infrastrutture di cloud computing, mi hanno spinto a considerare seriamente la possibilità d’inserirmi in questo settore specifico rispetto al vasto panorama informatico, senza limitarmi alla semplice attività di programmazione.

Dopo aver esposto le mie intenzioni al Prof. Scarano, una volta giunto alla fase conclusiva del mio percorso di laurea di I livello, ho svolto l’attività di tirocinio curricolare – come previsto dal Corso di Laurea in Informatica – presso il laboratorio *ISISLab*, lavorando ad un’espansione del progetto DMASON del Dott. Carmine Spagnuolo; l’attività di tirocinio è culminata nella presentazione del mio lavoro all’intero laboratorio, col seminario *DMASON on the cloud: Amazon AWS* del 1 Dicembre 2017.

Lo sviluppo del modulo è terminato nel mese di Gennaio 2018; i miei contributi sono visibili all’indirizzo <https://github.com/isislab-unisa/dmason/commits?author=bissim>.

La presente tesi è stata ultimata nel mese di Febbraio 2018.

Strumenti

Sviluppo Nel corso dello sviluppo, sono stati usati i seguenti prodotti:

- IDE

Eclipse Oxygen 4.7;
AWS Toolkit for Eclipse;
Visual Studio Code 1.18;

- **Back-end**

Java Development Kit 8;
Apache Maven;
Amazon SDK for Java;
JSch;
Apache Commons Configuration;

- **Front-end**

Bower;
Google Polymer;
jQuery;
Masonry;
jsRender.

Eccettuati gli ambienti di sviluppo integrati, tali strumenti sono ampiamente esposti nei Capitoli 3, 4, 5.

Composizione tipografica La presente tesi è stata redatta in \LaTeX , facendo uso dell'IDE TexStudio 2.12.6 e del gestore di pacchetti MiKTeX 2.9; per la struttura, si è preso spunto dal modello di tesi del Prof. Scarano e disponibile sul sito dell'ISISLab, con l'aggiunta dei seguenti pacchetti:

- `listings` per i listati;
- `minitoc` per gli indici di capitoli;
- `biblatex` per la bibliografia;
- `subcaption` per le didascalie delle immagini composte;
- `forest` per la creazione di alberi di directory.

Parte I

**DMASON su cloud: Amazon
EC2**

Capitolo 1

Introduzione

Indice

1.1	Simulazione	1
1.1.1	Ambienti di simulazione multi-agente	2
1.1.2	MASON	2
1.1.3	DMASON	4
1.2	Cloud computing	5
1.2.1	Sviluppo e commercializzazione	7
1.2.2	Caratteristiche e modalità di servizio	7
1.2.3	Problemi e critiche	9
1.2.4	<i>Amazon Web Services</i> ed <i>Elastic Computing Cloud</i>	10
1.3	Obiettivi	12

1.1 Simulazione

Nell'antichità, lo studio dei fenomeni naturali era un processo empirico: si osservava il fenomeno, si formulava una teoria al riguardo ed *empiricamente* se ne dimostrava la fondatezza; spesso, questo processo si ripeteva innumerevoli volte prima di giungere a una tesi rigorosa e difficilmente confutabile.

Se fenomeni che ricadono nella branca dell'elettrostatica (piuttosto che della meccanica o altre aree della fisica classica) prevedono esperimenti elementari (facilmente riproducibili per la loro elementarità) ve ne sono altri più impegnativi in quanto richiedono ingenti risorse (energia, tempo, denaro). Queste sperimentazioni non possono basarsi sulla sola esperienza ma è necessario formulare dei modelli matematici che riproducano idealmente il fenomeno nel suo stato iniziale e la sua evoluzione nel tempo: in altre parole, va progettato un *modello di simulazione* che descriva il fenomeno.

La **simulazione** è una disciplina dell'area informatica di non recente sviluppo: già nel 1946, John von Neumann e Nicholas Metropolis elaboravano una tecnica detta "*metodo del centro del quadrato*" [1] per generare

sequenze di numeri casuali che simulassero esperimenti come lanci di monete o estrazioni dall'urna ma permettendo di ampliarne le possibilità, ad esempio simulando il lancio della moneta diecimila volte o l'estrazione di una biglia da un'urna che ne contenga un milione. Oggigiorno, a seguito della rapida evoluzione del processo di sviluppo dei microprocessori, risulta molto più economico sostenere le spese di gestione di un elaboratore su cui sia in esecuzione una simulazione piuttosto che tentare di riprodurre nella realtà il fenomeno da studiare.

I *sistemi multi-agente* risultano molto interessanti grazie anche al fatto che l'evoluzione tecnologica rende più economico simulare tali sistemi; i campi di ricerca in cui si rivelano utili sono svariati: si va dalla biologia alle relazioni sociali, passando per economia, politica, gestione del territorio. Ambiti particolarmente prolifici per lo sviluppo di simulazioni multi-agente sono la robotica e lo studio di gruppi di entità (come sciame di insetti o stormi di uccelli).

1.1.1 Ambienti di simulazione multi-agente

I simulatori multi-agente più diffusi sono realizzati principalmente per la robotica e per le relazioni sociali, per i quali esistono soluzioni come TeamBots [2] o SWARM [3]. Per chi lavora in un ambito specifico, tali ambienti possono risultare ideali; tuttavia, chi si occupa dello studio di problemi di natura eterogena tra loro troverà seccante dover cambiare di volta in volta ambiente di simulazione, a seconda del problema studiato.

C'è inoltre da considerare che l'attendibilità degli esperimenti svolti, a parità dei dati iniziali, è data dalla loro riproducibilità, la quale deve poter avvenire su sistemi con diverse caratteristiche: TeamBots, per esempio, è scritto in Java, il che gli permette di essere portabile su diversi tipi di sistemi.

1.1.2 MASON

Un simulatore che risponde ai requisiti di *portabilità* e *genericità* è **MASON** [4], toolkit sviluppato in linguaggio Java da una collaborazione tra il Dipartimento di Informatica della *George Mason University* e il Centro per lo studio della complessità sociale della medesima istituzione.

MASON è un simulatore *single-process* a eventi discreti utilizzabile per un'ampia gamma di modelli di simulazione, in particolare multi-agente con la possibilità di avere milioni di agenti. La filosofia di progettazione con la quale è stato sviluppato prevedeva una libreria di modellazione veloce, minimale e facilmente espandibile da un programmatore Java esperto; MASON nasce anche dall'esigenza dei suoi creatori di avere un sistema che, per l'appunto, potesse essere usato con modelli di simulazione provenienti da diversi domini, eseguiti efficientemente (anche in *parallelo*) su cluster di computer. Piuttosto che fare uso di un sistema di simulazione fortemente legato a un

dominio applicativo, rischiando di introdurre dei bug nel funzionamento nel processo di generalizzazione, oppure di un sistema in cui il modello e la sua visualizzazione grafica fossero troppo *accoppiati* oppure di un sistema scritto in linguaggi interpretati, i progettisti di MASON preferirono sviluppare una soluzione *ex novo*.

I principi di progettazione di MASON sono stati:

Leggerezza	un nucleo piccolo, semplice da capire ed estendere;
GUI separata	visualizzazione in 2D e 3D svincolata dal modello;
Portabilità	riproducibilità del risultato, indipendentemente dalla piattaforma utilizzata;
Suspendibilità	un modello può essere salvato sul disco di una piattaforma per essere ripreso su una piattaforma diversa;
Numerosità agenti	fino a un milione di agenti senza visualizzazione del modello;
Integrabilità	la possibilità di inclusione in altre librerie.

Architettura di MASON

L'uso di Java quale linguaggio di sviluppo ha permesso a MASON di usufruire della *tipizzazione forte* del linguaggio (a garanzia della riproducibilità dei risultati) e la *serializzazione* degli oggetti (per il salvataggio dello stato della simulazione); un altro punto di forza (a detta dei progettisti di MASON) è la *velocità* del linguaggio: la loro esperienza di sviluppo in Java li ha convinti del fatto che la cattiva reputazione di Java riguardo alla presunta lentezza di esecuzione sia immotivata e che piuttosto il tempo di esecuzione sia determinato dalla qualità del programma. C'è anche da dire che una simulazione non è un processo *time critical*: cali di prestazioni dovuti all'attivazione del *garbage collector* di Java sono tollerabili, non influiscono negativamente sull'esito della simulazione.

MASON si compone di tre strati software: lo strato di utilità (*utility layer*), lo strato di modello (*model layer*) e lo strato di visualizzazione.

Utilità Il primo strato di utilità fornisce classi utilizzabili per qualsiasi scopo, come un generatore di numeri casuali basato sull'algoritmo di *Mersenne-Twister*, strutture dati più efficienti, elementi grafici e strumenti per la generazione di video e fermi immagine.

Modello Il secondo strato di modello è una collezione di classi che comprendono uno *scheduler a eventi discreti*, utilità per la schedulazione, diversi *campi* contenenti oggetti ai quali vengono associate delle posizioni. Questo strato è necessario e sufficiente a scrivere simulazioni eseguibili da riga di comando.

Visualizzazione Il terzo strato di visualizzazione permette, tramite interfaccia grafica, la visione e la manipolazione del modello; l'uso di questo strato comporta un calo delle prestazioni dovuto alla necessaria sincronizzazione tra l'esecuzione del modello e quella della grafica che lo riproduce.

Limiti di MASON

Ci sono tre principî di progettazione volutamente trascurati dai progettisti di MASON:

- MASON è inteso per l'esecuzione su singola macchina e non prevede la distribuzione di una singola esecuzione su più macchine.
- MASON ha un nucleo semplice e compatto, non prevede caratteristiche specifiche per un singolo dominio.
- La gestione efficiente della memoria, pur grossomodo perseguita, non è un aspetto prioritario.

Il limite più grande è forse quello dell'impossibilità di *distribuire* l'esecuzione: se gli altri limiti rispondono pienamente ai principî perseguiti dai progettisti, questa sembra più una loro mancanza; tuttavia, la compattezza della libreria, la sua facilità d'uso e la sua conclamata espandibilità permettono di aggiungere eventuali caratteristiche desiderate: il sito ufficiale [4] offre estensioni per MASON e framework basati su di esso.

La possibilità di eseguire **simulazioni distribuite** su più macchine, tuttavia, ricopre un ruolo cruciale nell'esecuzione di modelli basati su agenti che, seppur semplici, possono avere istanze di taglia elevata e richiedere ingenti risorse computazionali che metterebbero a dura prova una singola macchina e dilaterrebbero i tempi di esecuzione; invece, a parità di modello di simulazione, partizionando l'istanza del problema iniziale e sottoponendo ogni parte a un computer distinto, si avrebbero dei tempi di esecuzione decisamente più accettabili.

1.1.3 DMASON

Tra i framework basati su MASON troviamo **DMASON** [5], dove la "D" sta per *Distributed* (in italiano *distribuito*), a denotare la natura del framework che prevede la parallelizzazione su diverse macchine di simulazioni.

Esso si basa sul paradigma *master-worker* che prevede che una macchina partizioni il carico di lavoro suddividendolo in regioni, le quali vengono assegnate, assieme agli agenti che esse contengono, tra le varie macchine deputate all'esecuzione della simulazione, le quali si occupano di:

- simulare gli agenti appartenenti alle regioni assegnate;
- gestire la migrazione di agenti da una regione a un'altra a essa adiacente;
- gestire la sincronizzazione tra regioni adiacenti, in modo tale che la simulazione venga eseguita in maniera consistente.

La suddivisione in regioni può essere anche effettuata dalle macchine stesse; in tal caso, il compito del master è quello di monitorare l'andamento della simulazione. Le macchine che eseguono simulazioni di DMASON comunicano tra di loro tramite il paradigma *produttore-consumatore* (anche noto in letteratura come *Publisher-Subscriber*): a ogni regione è associato un canale a cui le macchine si iscrivono se la regione appartiene alla loro *area d'interesse*, in modo da ricevere i relativi messaggi di aggiornamento.

D-simulation DMASON introduce, rispetto a MASON, un ulteriore livello di parallelizzazione di cui l'utente del framework non deve pienamente essere a conoscenza (se non, per esempio, nel decidere in quante parti suddividere il carico di lavoro); questo strato permette la distribuzione della simulazione su più macchine, anche diverse tra loro. L'introduzione di questo nuovo strato non compromette le funzionalità degli strati preesistenti: è stato infatti progettato in modo tale che applicazioni preesistenti basate su MASON possano facilmente essere convertite per l'uso con DMASON.

Una volta rese distribuibili le simulazioni, è sufficiente allestire un cluster di computer per l'esecuzione di DMASON; il processo non è immediato e può richiedere più tempo di quello necessario all'esecuzione della simulazione; inoltre, anche avendo a disposizione un cluster, non si avrebbe *scalabilità* in caso di simulazioni particolarmente complesse.

1.2 Cloud computing

Predisporre un gruppo di macchine per l'esecuzione di programmi distribuiti non è semplice: i costi di esercizio (l'energia elettrica necessaria sia per il raffreddamento sia per le macchine stesse) e di gestione (la manutenzione da parte di operatori umani) non la rendono un'operazione appetibile a chiunque se non a grandi aziende che abbiano necessità di gestire ingenti moli di dati; c'è anche da aggiungere che tali sistemi *on premise* mal si adattano ai picchi di carico: le aziende non possono permettersi l'acquisto

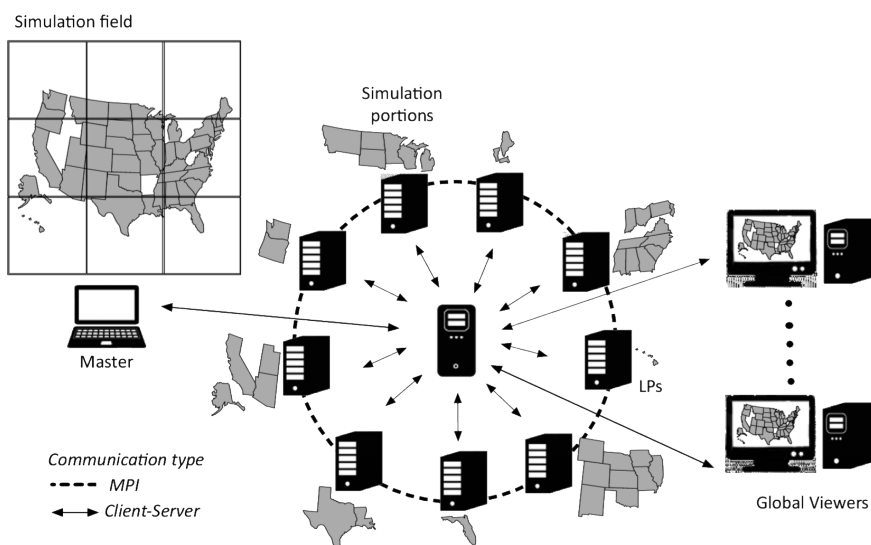


Figura 1.1: Paradigma *master-workers* in DMASON.

di attrezzature da usare saltuariamente quando ciò si verifici, tenendole accantonate per la maggior parte del tempo.

Come fare se però si ha necessità di potenza di calcolo per pochi mesi o addirittura poche ore? La soluzione più ovvia è accordarsi con altre grandi aziende per l'uso comune di risorse già nella disponibilità di una di esse oppure si può optare per il noleggio di potenza di calcolo da aziende che il cui business consiste proprio nell'erogare servizi specifici per il calcolo remoto; questa modalità di consumo di risorse di calcolo presso Internet prende il nome di **cloud computing**: il termine *cloud* (in italiano *nuvola*) è dovuto al fatto che Internet, fin dalla sua nascita negli anni '70, viene schematizzata nei diagrammi come una nuvola; questa notazione è poi lentamente entrata a far parte del linguaggio comune. Nel 1993, AT&T, nota società di telecomunicazione americana, in un suo video promozionale in cui presentava alle aziende Telescript¹ [6], già accennava al concetto di *cloud* per denotare la fruizione di servizi in remoto:

*“ You can think of our electronic meeting place as the **Cloud**. PersonaLink was built from the ground up to give handheld communicators and other devices easy access to a variety of services. [...] Telescript is the revolutionary software technology that makes intelligent assistance possible. Invented by General Magic, AT&T is the first company to harness Telescript, and bring its benefits to people everywhere. [...] Very shortly, any-*

¹Linguaggio di programmazione a oggetti distribuiti, sviluppato da General Magic, azienda spin-off di Apple. Sintassi ispirata a C, modello simile a Java.

one with a computer, a personal communicator, or a television will be able to use intelligent assistance in the Cloud. And our new meeting place is open, so that anyone, whether individual, entrepreneur, or multinational company, will be able to offer information, goods, and services. ”

1.2.1 Sviluppo e commercializzazione

Negli anni 2000 si entra nel pieno della commercializzazione dei servizi di cloud computing, principalmente pensando alle aziende come clienti; bisognerà attendere il 2004 perché un servizio di cloud computing sia offerto al grande pubblico, con Amazon che rilascia una funzionalità della propria piattaforma AWS: *Simple Queue Service* [7]. Due anni dopo, sempre Amazon avrebbe rilasciato il primo servizio di cloud computing inteso in senso stretto: *Elastic Compute Cloud* [8]. Le altre grandi aziende informatiche non restano a guardare: nel 2008, anche Google rilascia al pubblico il servizio *Google App Engine* [9] per l'esecuzione di applicazioni web; nel 2010, Microsoft rilascia *Microsoft Azure* [10], annunciato due anni prima; nel 2011 anche IBM entra in gioco, annunciando il framework *IBM SmartCloud* a supporto di *Smarter Planet* [11]; nel 2012, Oracle annuncia *Oracle Cloud* [12]; nello stesso anno, Google annuncia il rilascio in anteprima di *Google Compute Engine*, poi reso accessibile al pubblico nel 2013 [13].

Vi sono anche approcci per una standardizzazione del cloud computing, sia a livello software che a livello hardware: nel 2008, la NASA rilascia *OpenNebula* [14], il primo software *open source* per l'allestimento di sistemi cloud privati o ibridi e la realizzazione di sistemi federati di cloud; due anni più tardi, Rackspace Hosting e NASA lanciano una nuova iniziativa *open source* orientata al cloud chiamata *OpenStack* [15], finalizzata ad aiutare le aziende che erogano servizi di cloud ad utilizzare hardware standard.

Il cloud computing può essere inteso come un tipo specifico di *grid computing* orientato alla qualità del servizio: offre gli strumenti per costruire applicazioni parallele che richiedono una notevole potenza di calcolo ma a prezzi competitivi rispetto alle soluzioni classiche di calcolo parallelo.

1.2.2 Caratteristiche e modalità di servizio

Un sistema di cloud computing risponde tipicamente alle seguenti caratteristiche:

- maggiore agilità per le organizzazioni, in quanto le risorse (hardware e software) possono essere facilmente allocate e deallocate;
- riduzione e migrazione dei costi dalla gestione al servizio;
- indipendenza dalla posizione e dal dispositivo per l'accesso al servizio;

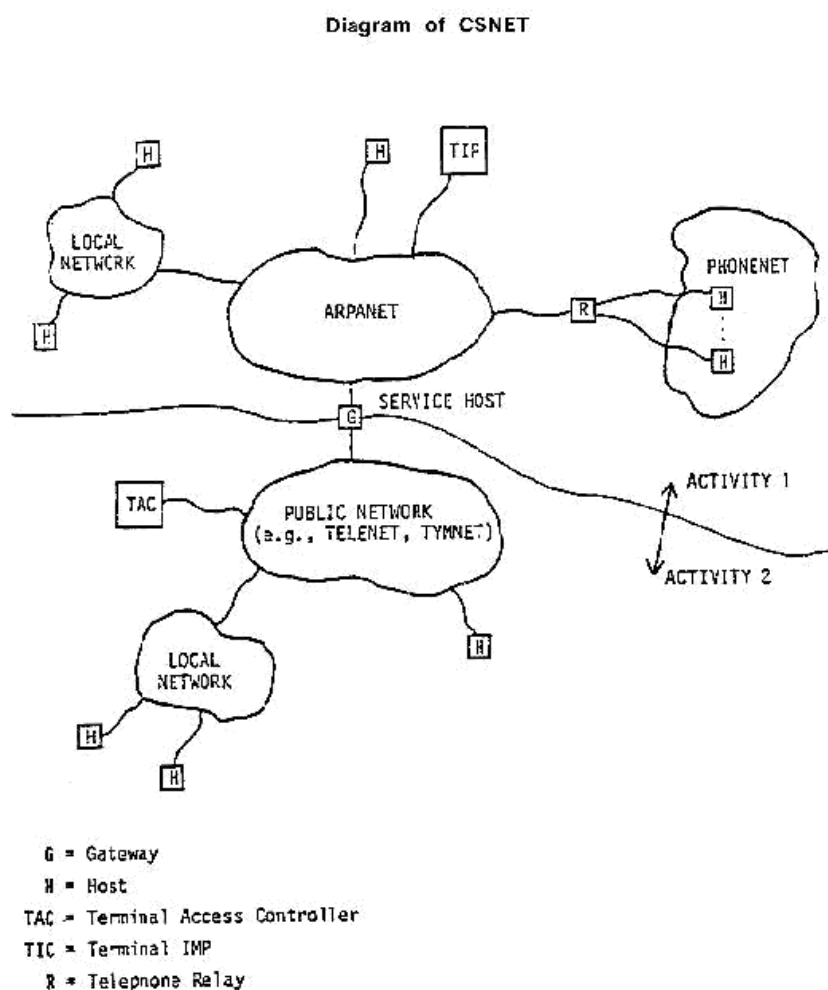


Figura 1.2: L'uso della nuvola come schematizzazione di una rete di calcolatori nel diagramma di CSNET [16] è chiaro in quanto esso rappresenta la congiunzione di reti amministrativamente distinte piuttosto che di singoli nodi.

- alta manutenibilità: le applicazioni cloud non vanno installate su più macchine;
- condivisione delle risorse tra una grande quantità di utenti;
- monitoraggio delle prestazioni;
- affidabilità dovuta alla struttura ridondante dell'architettura cloud;
- scalabilità del servizio erogato;

Se però le caratteristiche possono variare, nel numero e nella sostanza, da provider a provider, il NIST ha definito delle linee guida [17] che tutti i sistemi di cloud computing dovrebbero rispettare:

- **self service su richiesta:** l'utente può richiedere risorse di calcolo senza interagire con operatori umani del fornitore del servizio;
- **ampia possibilità di accesso:** le risorse sono disponibili in rete per l'accesso da diversi dispositivi di varia natura;
- **pooling delle risorse:** il fornitore mette a disposizione le proprie risorse in un *pool*, assegnate su richiesta, dal quale attingono i clienti;
- **elasticità:** le risorse sono fornite e rilasciate automaticamente, in modo da scalare in base alla richiesta: questo aspetto è trasparente al cliente il quale ha la percezione di una disponibilità illimitata di risorse;
- **monitoraggio:** i sistemi cloud controllano e ottimizzano in modo automatico le risorse regolando il monitoraggio in base al servizio fornito;

Le applicazioni che fanno uso di servizi di cloud computing possono avere diversi livelli di complessità: alcune richiedono l'allestimento di uno specifico sistema operativo mentre per altre è sufficiente avere un ambiente di esecuzione già allestito. A seconda di queste necessità, il fornitore di servizi di cloud computing offre generalmente le seguenti **modalità di servizio**:

- **Infrastructure as a service (IaaS):** oltre alle risorse virtuali in remoto, vengono messe a disposizione anche risorse hardware quali server, capacità di rete, sistemi di memoria, archivio e backup; la caratteristica dello IaaS è che le risorse vengono istanziate su richiesta al momento in cui una piattaforma ne ha bisogno.
- **Platform as a service (PaaS):** invece che uno o più programmi singoli, viene eseguita in remoto una piattaforma software che può essere costituita da diversi servizi, programmi, librerie, ecc.;
- **Software as a service (SaaS):** consiste nell'utilizzo di programmi installati su un server remoto, spesso attraverso un server web.

Esistono ulteriori modalità di erogazione del servizio che rispondo al paradigma "*as a service*", come il *Back-end as a service* dedicato alle applicazioni mobile; tutte possono però essere ricondotte alle tre già illustrate.

1.2.3 Problemi e critiche

Usufruire dei servizi di cloud computing è certamente una situazione da cui sia il fornitore sia il cliente traggono molto vantaggio; tuttavia, entrambi possono incorrere in alcuni problemi intrinseci di questa soluzione, principalmente legati alla sicurezza e riservatezza dei dati:

- **sicurezza informatica e privacy degli utenti:** le informazioni sensibili sono memorizzati in *server farm* che spesso non risiedono nello stesso stato di appartenenza dell'utente; oltretutto, il cloud provider potrebbe accedere alle informazioni se non crittografate;
- **problemi internazionali economico-politici:** se i dati appartengono a enti pubblici c'è il rischio che tali dati, sempre residenti in paesi esteri, non risultino più accessibili

Una problematica non intrinseca al servizio è la **continuità del servizio**: se i dati sono gestiti da servizi esterni, in caso di assenza di servizio essi risultano inaccessibili; le politiche di compensazione del sistema distribuito su cui si fonda la struttura cloud potrebbero non essere sufficienti in quanto l'accessibilità potrebbe essere compromessa lato client (si pensi a un'interruzione del servizio di accesso a Internet presso il cliente).

C'è anche chi critica questa modalità di erogazione di servizi: Richard Stallman, ideatore del *Progetto GNU* e fondatore della *Free Software Foundation*, critica aspramente il cloud computing, sostenendo che esso non sia altro che una trappola per vendere ai clienti i propri servizi proprietari sempre più costosi, arrivando a definirlo “stupido” [18]; altrettanto critico (ma con una posizione più moderata) è il CEO di Oracle Larry Ellison, riducendo il cloud computing a “un'espressione accattivante” per definire la fruizione di servizi su Internet.

1.2.4 *Amazon Web Services ed Elastic Computing Cloud*

Gartner, società di ricerca, nel rapporto “Quadrante magico per un'infrastruttura cloud come servizio” [19] ha rilevato che l'azienda leader nel settore del cloud computing è Amazon, con la sua piattaforma di servizi **Amazon Web Services**; la sua principale rivale è Microsoft con la piattaforma **Microsoft Azure**. Se è vero che, di fatto, il cloud computing si sviluppa parallelamente all'evoluzione di Internet – e nonostante nel 2017 Microsoft abbia fatturato più di Amazon nel settore [20] – bisogna riconoscere che un impulso notevole è stato dato in questo ambito da Amazon.

Nata nel 2002, la piattaforma di servizi *Amazon Web Services* (d'ora in poi AWS) offriva dapprima pochi servizi e strumenti per gli sviluppatori ma non l'accesso a macchine virtuali; bisognerà attendere il 2004 per avere il primo servizio offerto al pubblico, il già citato *Simple Queue Service*. In una conferenza stampa del 2006, AWS si ripropone [21] come combinazione di tre servizi: *Simple Storage Service (S3)* per il cloud storage, *Simple Queue Service (SQS)* per la messaggistica e *Elastic Compute Cloud (EC2)* per il calcolo su cloud.

Proprio Amazon EC2 costituisce il servizio di AWS più rappresentativo del concetto di cloud computing: Amazon lo pubblicizza come “[...] un

1.2. CLOUD COMPUTING

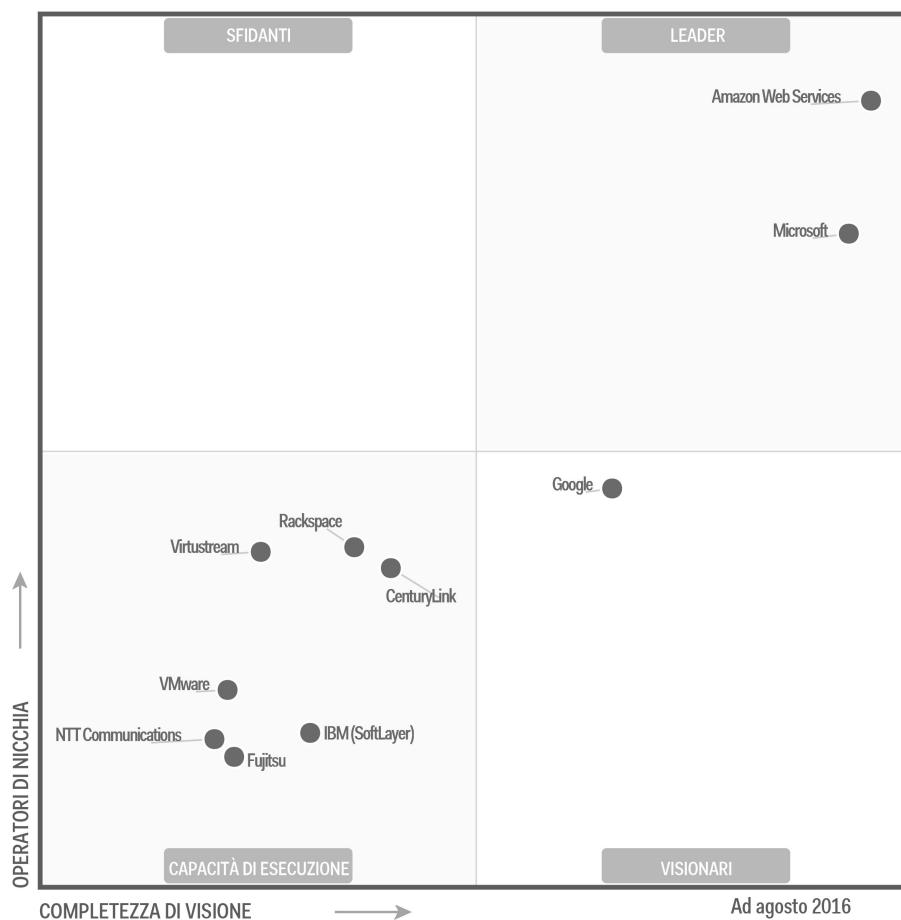


Figura 1.3: *Quadrante magico per un'infrastruttura cloud come servizio* di Gartner. Si nota la posizione di leadership di Amazon AWS, seguito da Microsoft.

servizio Web che fornisce **capacità di elaborazione sicura e scalabile** nel cloud. [...]”. EC2 viene offerto con quattro modalità di tariffazione:

- **istanze *on demand*:** si paga solo la capacità di elaborazione impiegata, all'ora o al secondo, in base al tipo di istanze eseguite; le istanze avviate possono scalare in base alla necessità della propria applicazione, pagando al termine della computazione solo per le istanze effettivamente impiegate in base alla tariffa oraria scelta;
- **istanze *riservate*:** si prenota capacità di calcolo in una specifica zona di disponibilità, risparmiando fino al 75% rispetto al prezzo della stessa tipologia di istanza richiesta on demand;
- **istanze *spot*:** si richiedono risorse EC2 inutilizzate e messe “all’asta”, risparmiando fino al 90% rispetto al prezzo della stessa tipologia di istanza richiesta on demand;

- **host dedicati:** server fisico che consente la riduzione dei costi impiegando licenze già in possesso dell'utente, risparmiando fino al 70% sui prezzi on demand.

Le tariffazioni più interessanti sono senz'altro la “on demand”, in quanto definisce il costo base del servizio (confrontabile con altri fornitori del servizio) e la “spot” [22] per l'alta percentuale di risparmio che offre all'utente rispetto alla tariffazione on demand. Le istanze spot, nello specifico, hanno lo svantaggio di essere interrompibili da Amazon in qualsiasi momento, con un preavviso di soli due minuti, quando la base d'asta per un'istanza EC2 superi l'offerta corrente da parte dell'utente; ciò non costituisce un grave problema per l'utente in quanto le istanze spot sono dedicate a coloro che necessitano di scalare rapidamente verso l'alto le prestazioni per la propria applicazione web senza però acquistare un'istanza on demand di base già più performante.

1.3 Obiettivi

Il servizio EC2 risulta ideale per applicazioni come DMASON, studiate per il calcolo distribuito. In precedenza, per poter effettuare una simulazione in DMASON, era necessario dover predisporre manualmente delle macchine, magari proprietarie, per l'esecuzione (Capitolo 2); con questo lavoro di tesi, si vuole mostrare come è stato possibile – dopo aver ripristinato l'usabilità del System Management (Capitoli 3 e 4) e grazie all'*SDK AWS* [23] (Capitolo 5) – permettere a DMASON di interfacciarsi automaticamente con il servizio EC2 per la creazione dei *worker*, richiedendo un intervento di configurazione minimo all'utente finale e il possesso di un account AWS [24].

Capitolo 2

DMASON in azione

Indice

2.1	Introduzione	13
2.2	Interfaccia grafica	13
2.3	Funzionamento	15
2.3.1	Costruzione	15
2.3.2	Esecuzione	18
2.4	Limitazioni	19

2.1 Introduzione

In questo capitolo si vogliono presentare aspetto e funzionamento di DMASON, senza riferimenti specifici alle tecnologie che ne realizzano le singole funzioni; le informazioni qui riportate sono tratte dalla *wiki* ufficiale di DMASON [25].

2.2 Interfaccia grafica

Tramite un'interfaccia grafica, l'utente interagisce con DMASON sia controllando lo stato dei worker liberi, delle simulazioni in esecuzione e di quelle concluse sia richiedendo nuove simulazioni.

DMASON dispone di una console di gestione chiamata *System Management* dalla cui pagina principale (la vista *Monitoring*) è possibile, a prima occhiata, osservare lo stato dei worker disponibili non occupati in attività di simulazione.

Per l'utilizzo del System Management, è necessario usare browser web come *Firefox* di Mozilla oppure *Google Chrome* di Google; le ragioni saranno spiegate nel Capitolo 3.

Dalla pagina principale è possibile:

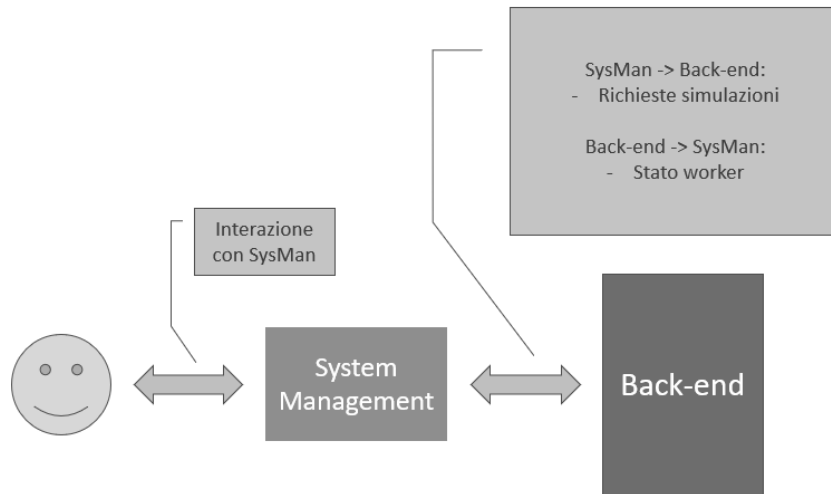


Figura 2.1: Schema d'uso di DMASON

- selezionare uno o più worker;
- arrestare i worker;
- selezionati alcuni worker, pianificare una simulazione.

Per ogni worker sono mostrate le seguenti caratteristiche: percentuale di CPU utilizzata, uso di memoria (massima, usata, disponibile), indirizzo IP e numero di *slot*.

Dal pannello di creazione simulazione – dove sono mostrati il numero di worker selezionati e il numero di loro slot totali – è richiesto il modello da utilizzare, selezionabile da una lista preesistente di esempi o di modelli caricati in precedenza) oppure si può caricare un nuovo modello; è possibile scegliere il tipo di partizionamento del problema: *uniforme* se viene suddiviso in parti uguali tra tutti i worker, *non uniforme* se viene determinata la dimensione delle singole parti in modo indipendente l'una dall'altra, ad esempio in base alle caratteristiche di ogni worker; si possono specificare, inoltre, il nome della simulazione, il numero di passi di simulazione, la suddivisione del problema (righe e colonne nel caso uniforme, celle nel caso non uniforme), l'area d'interesse, la dimensione del campo e il numero di agenti che esso contiene. Il numero di celle (oppure di righe per colonne) non può superare il numero di slot disponibile.

Nella vista *Monitoring* (Figura 2.2) è presente anche il pannello di controllo, un menù da cui è possibile raggiungere le viste ***Simulations***, ***History*** e ***Settings***.

Nella vista *Simulations* (Figura 2.3) è possibile avviare le simulazioni pianificate e gestire quelle in esecuzione, le quali possono essere messe in

pausa o arrestate; ogni simulazione è rappresentata da una scheda che riporta i seguenti dati: ID, nome, modalità di partizionamento, numero di celle, numero di worker, numero di agenti, data di inizio, passo di simulazione. Cliccando sulla scheda di una simulazione in corso, è possibile consultare altre informazioni ulteriori, come la percentuale media di CPU utilizzata per la simulazione e i file di *log* che possono essere scaricati.

Nella vista *History* (Figura 2.4a) è possibile consultare i dettagli delle simulazioni terminate, che consistono negli stessi dati riportati mentre erano in corso corredati di tempo di fine, durata in ore, durata in millisecondi e lo stato di terminazione; anche da questa vista, possono essere scaricati i log della simulazione.

Nella vista *Settings* (Figura 2.4b) è prevista solo l'impostazione dei dati del master (indirizzo IP e numero di porta); tuttavia, questi dati non vengono realmente impostati.

2.3 Funzionamento

Per poter essere usato, DMASON va prima “costruito” a partire dal *repository* su cui è ospitato.

2.3.1 Costruzione

Il processo di *build* (costruzione) inizia con lo scaricare una copia del repository remoto in una cartella locale del proprio sistema; una volta aperta una finestra di terminale e posizionatisi sulla cartella che conterrà il repository locale, si possono scaricare i file di DMASON col seguente comando Git:

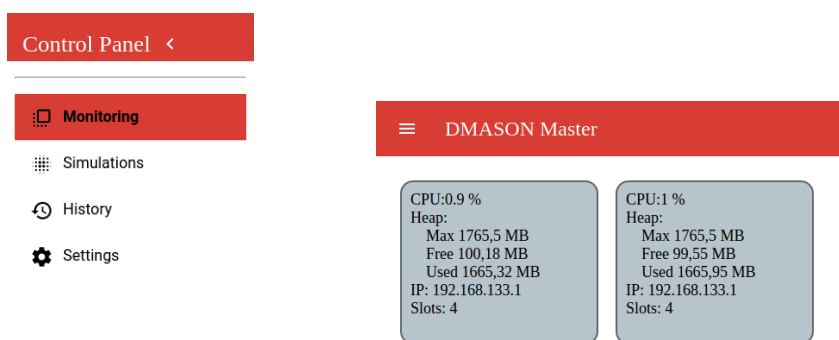
```
$ git clone https://github.com/isislab-unisa/dmason.git
```

Codice 2.1: DMASON, download con Git

Da qui, il processo di costruzione vero e proprio può differire se si preferisce usare un'interfaccia di sviluppo o meno; in questa sede, sarà illustrato solamente quello che prevede l'uso del programma *Apache Maven* [26] per la gestione delle dipendenze e costruzione del progetto. Il comando è il seguente:

```
$ mvn -Dmaven.test.skip=true clean package
```

Codice 2.2: DMASON, costruzione con Maven



(a) Pannello di controllo

(b) Vista worker

Simulation Settings

Worker(s) selected: 2 Available slot(s): 8

Select an external simulation

UPLOAD

Select an example simulation

Select

PARTITIONING

Uniform

Non-Uniform

EXTRA

enable MPI boost

PARAMETERS

Simulation name	Number of step	Cells
Rows	Columns	Area of interest
Width	Height	Number of Agents

RESET **SUBMIT**

(c) Pannello nuova simulazione

Figura 2.2: Vista *Monitoring*.

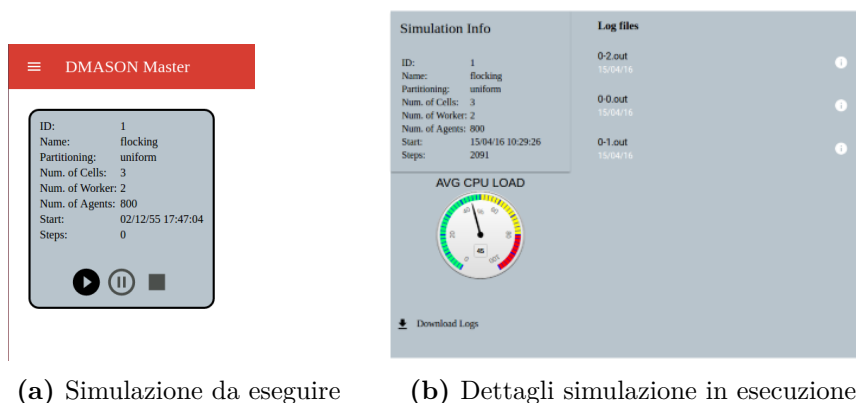


Figura 2.3: Vista *Simulations*

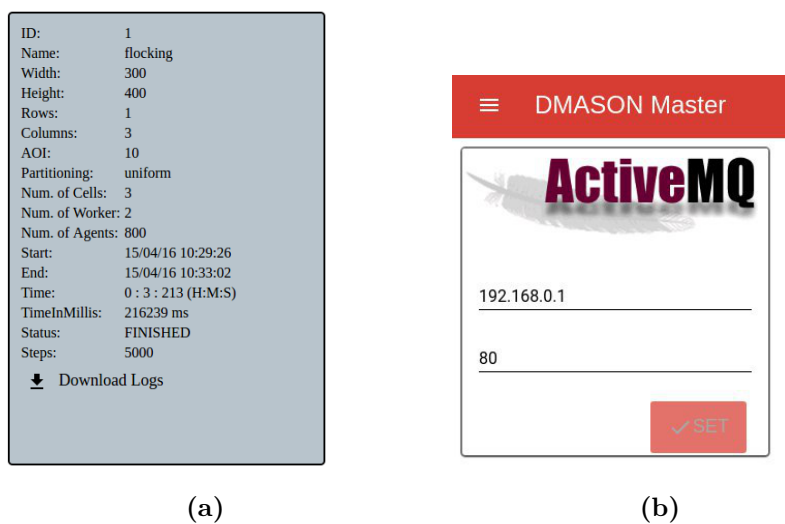


Figura 2.4: A sinistra, i dettagli di una simulazione terminata; a destra, le impostazioni del server di messaggistica.

Questo comando creerà una cartella `target` contenente l'eseguibile di DMASON; inoltre, il parametro `-Dmaven.test.skip=true` comporta la non esecuzione dei test previsti.

2.3.2 Esecuzione

DMASON si compone di tre componenti principali:

- server di messaggistica;
- master;
- uno o più worker.

Si presenta come un archivio Java (un file `.jar`) eseguibile da riga di comando:

```
$ java -jar DMASON-3.2.jar -m <mode>
```

Codice 2.3: DMASON, comando base per l'esecuzione

L'argomento per il parametro `-m` permette di specificare se DMASON vada eseguito in modalità *master* o in quella *worker*; nel secondo caso, è necessario specificare il parametro `-ip` per indicare l'indirizzo del master a cui connettersi, il parametro `-p` per specificare a quale porta connettersi e il parametro `-ns` per indicare il numero di slot disponibile al worker (è consigliabile che il numero di slot corrisponda al numero di processori fisici sulla macchina in cui venga eseguito il worker).

Per impostazione predefinita, DMASON fa uso di un server per la messaggistica integrato; all'occorrenza, può essere specificato l'uso di un server esterno con una piccola modifica al codice sorgente.

In modalità *worker*, possono essere specificati con un solo comando gli indirizzi di tutte le macchine su cui vada eseguito DMASON in tale modalità, specificando il parametro `-h` seguito dagli indirizzi IP delle succitate macchine:

```
$ java -jar DMASON-3.2.jar -m worker -h <ip1> ... <ipN>
```

Codice 2.4: DMASON, esecuzione simultanea di più worker

Nel codice in 2.4 sono stati omessi per brevità i parametri `-ip`, `-p` e `-ns`, necessari per l'esecuzione di un worker.

2.4 Limitazioni

Come rilevato nel paragrafo 2.3.2, la pagina per le impostazioni del server di messaggistica per la comunicazione master-server non è funzionante; la limitazione maggiore consiste tuttavia nel dover allestire singolarmente le singole macchine per predisporle all'esecuzione di DMASON (installando gli strumenti di sviluppo di Java, Apache Maven eccetera).

Al capitolo 4, sarà illustrato come si è predisposto DMASON all'installazione automatica su macchine virtuali ospitate da Amazon AWS, le cui impostazioni saranno effettivamente configurabili (insieme a quelle del server di messaggistica) nella vista *Settings*.

Capitolo 3

Sviluppo frontend

Indice

3.1	Tecnologie utilizzate	20
3.1.1	Polymer	21
3.1.2	Bower	21
3.1.3	jQuery	21
3.1.4	Masonry	22
3.1.5	jsRender	22
3.1.6	JavaServer Pages e JSP Standard Tag Library . . .	22
3.2	Attività preliminare	23
3.3	Modifiche introdotte	25
3.3.1	Struttura attuale del progetto	25
3.3.2	Riuso	25
3.3.3	Template per i worker	26
3.3.4	Informazioni worker	27
3.3.5	Intervallo aggiornamento worker	27
3.3.6	Gestione impostazioni DMASON	27

3.1 Tecnologie utilizzate

L'aspetto della console di *System Management* mostrato al Capitolo 2 si rifà al paradigma del *Material Design* [27], sviluppato da Google, le cui linee guida sono state rilasciate nel 2014. Il Material Design viene impiegato principalmente per le applicazioni per smartphone, tuttavia esse possono essere applicate anche in ambito di web design: la stessa Google – lavorando al *progetto Polymer* – ha sviluppato una libreria sperimentale per costruire facilmente siti web che implementino questo paradigma.

3.1.1 Polymer

Polymer è una libreria JavaScript che permette di “creare elementi HTML riutilizzabili e usarli per costruire app performanti e manutenibili” [28], espandendo in questo modo il concetto introdotto nella specifica 5 di HTML di “tag semantico” [29, 30]; come quando si parla spesso di riuso di codice, è stato sviluppato Web Components [31], un sito web per la distribuzione di componenti preassemblate.

Polymer permette anche agli sviluppatori di utilizzare tecniche previste da specifiche di recente rilascio (quali, per esempio, le API di *Web Animations* [32]) grazie all’integrazione nelle pagine HTML di librerie apposite.

3.1.2 Bower

Il rilascio di nuovi componenti per Polymer è consigliato tramite la su citata piattaforma Web Components; tuttavia – specie nel caso di componenti sperimentali – può capitare che essi siano disponibili su altro tipo di piattaforma, rendendone complicata la ricerca e la diffusione. Per rimediare a tale problema, esistono gestori di pacchetti come *Bower* [33] che si occupano di effettuare la ricerca sui principali siti di repository e di rilascio di librerie quali *npm*, *GitHub* eccetera. Il sito principale di Bower fornisce un motore di ricerca per i pacchetti, in modo da specificare poi le *dipendenze* per Polymer in un file apposito chiamato `bower.json`.

3.1.3 jQuery

Le librerie sono molto utili per compiere operazioni in modo chiaro e conciso, senza la necessità di “reinventare la ruota”; non sempre, però, esiste una libreria per una funzionalità specifica da sviluppare: bisogna rimboccarsi le maniche e svilupparla. Scrivere programmi in JavaScript per il web risulta noioso a causa della diversa offerta di funzionalità dei diversi browser sul mercato, ognuno con un proprio interprete che traduce una diversa idea dello standard ECMAScript: se una funzione esiste su Firefox non è detto che esista su Chrome, costringendo lo sviluppatore web a fare test sui browser più diffusi.

Sè vero che non esistono librerie per ogni esigenza di progettazione, si può facilmente risolvere il problema della portabilità del codice tra i diversi browser sul mercato: *jQuery* [34] è una libreria che permette allo sviluppatore di risolvere questo problema, concentrandosi sulla logica di business piuttosto che sulla gestione dell’intercompatibilità. jQuery non fa solo questo: rende più semplici molte operazioni come le chiamate asincrone che, anche grazie al concatenamento di funzioni, permette di scrivere in poche righe una chiamata GET o POST specificando la destinazione, i dati da inviare e le funzioni che gestiscano l’esito della chiamata.

3.1.4 Masonry

DMASON fa uso di molti elementi ripetuti più volte all'interno di una pagina (worker, simulazioni, impostazioni); organizzarli in una griglia in maniera dinamica risulta tedioso da fare anche con l'aiuto di jQuery. Grazie a *Masonry* [35] è possibile, sempre a fronte di poche direttive, organizzare gli elementi ripetuti in modo che si dispongano ordinatamente nella pagina, "scorrendo" all'interno della stessa al variare del loro numero o delle dimensioni della pagina; questo permette agli sviluppatori web di implementare più facilmente le direttive del *Responsive Design* per la fruizione da dispositivi con qualsiasi fattore di forma (computer, tablet, phablet, smartphone).

3.1.5 jsRender

Alcuni elementi – come i già citati dettagli del worker – necessitano in maniera *asincrona* di essere replicati nella sola struttura, permettendo distintamente l'aggiornamento di ogni singolo oggetto: i worker di DMASON devono essere rappresentati tutti allo stesso modo ma ognuno con i propri dati. È necessario creare e aggiornare dinamicamente tramite JavaScript le schede contenenti i dettagli dei worker; farlo dichiarando gli elementi HTML come oggetti stringa rende il codice poco manutenibile. Con la libreria **jsRender** [36] è possibile creare *template* da riutilizzare a proprio piacimento, mantenendo distinta la parte di presentazione (il codice HTML) dalla logica di business: ciò è possibile con poche righe di codice, importando il file esterno contenente il template.

3.1.6 JavaServer Pages e JSP Standard Tag Library

Ci sono sezioni di codice che, anche in HTML, possono ripetersi tra più pagine; costruire la pagina con JavaScript nel momento in cui viene richiesta può risultare in un rallentamento sensibile nei tempi di caricamento.

In DMASON, le pagine web del System Management sono in formato **JSP** (che sta per *JavaServer Pages* [37]), le quali permettono l'esecuzione di codice Java; tuttavia, lo *scripting* di codice Java puro in pagine HTML è una pratica fortemente sconsigliata – in generale, quando non indispensabile, è sempre consigliato avere diversi linguaggi frammisti in un solo file – come anche visto nel caso di codice HTML inserito in file JavaScript.

Come in quel caso, esiste una soluzione che consiste nell'utilizzo di **JSP Standard Tag Library** (**JSTL** [38]) che permette di programmare senza dover ricorrere allo scripting in JSP ma utilizzando una notazione simile ad HTML.

Con JSP e JSTL è possibile, per esempio, riciclare parti di codice comuni come menù, *header*, *footer* eccetera.

3.2 Attività preliminare

L'aspetto del System Management è stato lievemente cambiato rispetto a quanto mostrato nel Capitolo 2, per meglio rispondere alle direttive del Material Design e per rendere più intuitivi alcuni elementi, migliorando l'esperienza utente; prima di ciò, è stato necessario un lavoro di riorganizzazione e aggiornamento delle librerie usate nel front-end: poiché il progetto Polymer è di tipo sperimentale, gli elementi deprecati divengono inutilizzabili (per questo, può capitare che una libreria non sia più distribuita né reperibile).

```
20 "polymer": "Polymer/polymer#~1.2.0",
   "iron-elements": "PolymerElements/iron-elements#~1.0.7",
   "paper-elements": "PolymerElements/paper-elements
     #~1.0.7",
   "gold-elements": "PolymerElements/gold-elements#~1.0.1",
   "neon-elements": "PolymerElements/neon-elements#~1.0.0",
25 "google-map": "GoogleWebComponents/google-map#~1.1.8",
   "elements": "*",
   "masonry": "~4.0.0",
   "neon-animation": "PolymerElements/neon-animation#~1.1.0"
```

Codice 3.1: Le dipendenze del System Management precedentemente dichiarate in `bower.json`

DMASON implementava il Material Design per mezzo del componente `paper-layout` – un elemento attualmente deprecato in favore di `app-layout` – che aveva smesso di funzionare correttamente rendendo l'interfaccia inerte e non interagibile sulla maggior parte dei browser; di conseguenza, elementi come il *drawer* (il menù laterale a scomparsa) sono stati riscritti con i nuovi elementi che fanno parte di `app-layout`.

Anche le dipendenze dichiarate nel file `bower.json` del System Management sono state aggiornate: in particolare, Polymer è stato portato alla versione 2 per poter far uso di componenti aggiornati. Sono state, inoltre, introdotte nuove dipendenze tra cui le più rilevanti sono le seguenti:

- **jQuery**, precedentemente importato manualmente nella cartella `js` del System Management, in modo da facilitarne l'aggiornamento;
- **jsRender** per creare e usare template;
- **web-animations-js**, dipendenza per l'uso delle Web Animations API precedentemente citate, necessaria per alcuni componenti;
- **webcomponentsjs** per la riproduzione di funzionalità previste da altri standard, non ancora disponibili su tutti i browser.

```
"jquery": "^3.2.1",
"jsrender": "^0.9.89",
"polymer": "*",
"web-animations-js": "*",
25 "webcomponentsjs": "^0.7.24",
"iron-elements": "*",
"iron-flex-layout": "*",
"paper-elements": "*",
"gold-elements": "*",
30 "neon-elements": "*",
"google-map": "*",
"masonry": "*",
"neon-animation": "*",
"app-layout": "*",
35 "highcharts": "^5.0.14"
```

Codice 3.2: Le dipendenze attuali del System Management

Nell'attuale versione del file delle dipendenze per il System Management, è stato impostato per alcune di essi il carattere speciale *, in modo che sia Bower a determinare quale versione installare; in alcuni casi, però, questo può comportare una richiesta all'utente su quale versione installare: dichiarando la proprietà `resolutions` in `bower.json`, può essere dichiarata una lista di dipendenze per cui si vuole specificare quale versione vada installata, anche nel caso in cui ci siano conflitti di versione causati da dipendenze del pacchetto di cui ognuna ne richiede una versione diversa¹.

Le dipendenze del System Management sono già caricate nel repository di DMASON nella cartella `bower_components` in modo da non costringere l'utente finale all'installazione di Bower, il che comporterebbe dover installare il gestore di pacchetti `npm` [39] che richiede a sua volta l'installazione di `Node.js` [40]; volendo installare le dipendenze con l'uso di Bower, è possibile farlo tramite riga di comando.

```
$ bower install
```

Codice 3.3: Installazione dipendenze (dal file `bower.json`)

Il comando nel Codice 3.3 installerà le dipendenze specificate nel file `bower.json`; è anche possibile installare i singoli pacchetti specificando dopo il parametro `install` il nome del pacchetto desiderato.

¹Se **A** richiede la versione *x* di **B** ma **C** ne richiede la versione *y*, `"resolutions": {"B": "x"}` dirimerà automaticamente il conflitto di versioni installando la versione *x*.

Alcuni elementi definiti specificamente per DMASON sono stati spostati dalla cartella `bower_components` a `custom_components`, per tenerli separati in modo da essere più facilmente recuperabili e anche per evitarne la cancellazione accidentale.

Infine, anche le pagine web del System Management sono state aggiornate di conseguenza, facendo uso dei nuovi componenti provenienti dal pacchetto `app-layout` e importando correttamente la libreria `webcomponents.js`.

3.3 Modifiche introdotte

Dopo essere stata resa nuovamente funzionante, l'interfaccia del System Management è stata modificata per riutilizzare codice – rendendo importabili gli elementi d'uso comune nelle pagine JSP – ma soprattutto per introdurre nuove funzionalità come i template per i worker e la modifica delle impostazioni.

3.3.1 Struttura attuale del progetto

Il front-end è organizzato secondo la struttura mostrata in Figura 3.1; essa non differisce di molto dalla precedente, in quanto le uniche modifiche sono state lo spostamento della cartella `custom_components` da `bower_components` a `master` e la creazione della cartella `fragments`.

Nella cartella `conf` è presente il file `config.properties` (di cui si parlerà nel paragrafo 4.2.2), in `fragments` sono presenti i frammenti JSP e il template del worker, in `js` è presente lo script `script.js` e in `WEB-INF` sono presenti il *deployer descriptor* `web.xml` e la cartella `lib` contenente la libreria JSTL.

3.3.2 Riuso

Molti elementi comuni nelle pagine JSP (drawer, parti di `<head>`) sono stati tramutati in frammenti e resi importabili con il tag `<jsp:include>`; qualora necessario, le eventuali variabili all'interno dei frammenti sono state impostate con il tag `<jsp:param>` posto all'interno di `<jsp:include>`.

```
316 <!-- Sliding drawer menu -->
317 <jsp:include page="fragments/header.jsp">
318   <jsp:param name="page" value="index"></jsp:param>
319 </jsp:include>
```

Codice 3.4: Esempio d'uso dei frammenti nelle pagine JSP

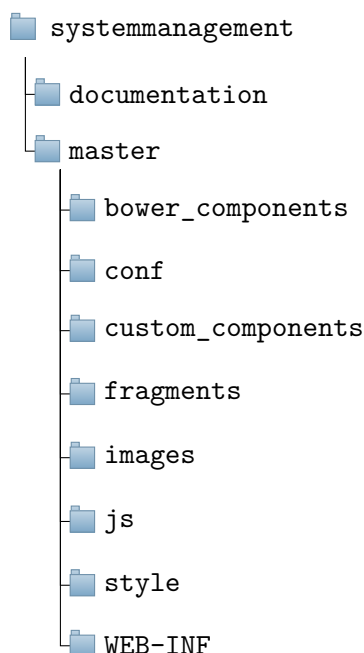


Figura 3.1: Struttura del progetto frontend

L'esempio nel Codice 3.4 è tratto dalla pagina `index.jsp`: il file `header.jsp` presente nella cartella `fragments` viene caricato e gli viene passato il parametro `index` per scegliere quali pulsanti mostrare nella barra in alto (cfr. appendice “Listati”, Codice A.1).

3.3.3 Template per i worker

I worker venivano generati e aggiornati nel System Management tramite una funzione che generava l'HTML a partire da stringhe (cfr. appendice “Listati”, Codice A.2); come detto in precedenza, ciò rende difficilmente manutenibile il codice.

Con l'uso dei *template* grazie alla libreria `jsRender`, è stato possibile separare il codice HTML per l'aspetto del worker dalla logica per la gestione delle schede nella vista *Monitoring*: dopo aver recuperato i dati del worker con una chiamata asincrona, questi vengono estratti e posizionati in oggetti JavaScript per l'iniezione all'interno del template, recuperato con una chiamata sincrona (cfr. appendice “Listati”, Codice A.3). Se i dati si riferiscono a un worker già visualizzato, non viene recuperato il template.

Grazie al *templating* è stato possibile aggiornare l'aspetto dei worker, rendendoli più conformi alle direttive del Material Design per gli elementi di tipo *card*.

3.3.4 Informazioni worker

Nella vista *Monitoring* (Figura 3.2) è stata inserita una scheda che permetta, a colpo d’occhio, di sapere quanti worker sono liberi, quanti sono selezionati e di quanti slot questi ultimi dispongono; è anche possibile variare la velocità di aggiornamento dei dati da 1 a 3 secondi, con un passo di 0,5 secondi.

3.3.5 Intervallo aggiornamento worker

Le informazioni dei worker nella vista *Monitoring* si aggiornavano ogni secondo; tuttavia, con l’introduzione dei worker remoti su EC2, potrebbe essere necessario più tempo per avere le informazioni per via delle invocazioni remote.

La funzione di aggiornamento dei worker è stata così inglobata in un’altra funzione che si occupa di eseguirla a intervalli variabili in base al valore specificato dal cursore scorrevole presente nella vista *Monitoring*. A intervalli regolari, la funzione `loadWorkersDynamicInterval()` (cfr. appendice “Listati”, Codice A.5) su citata si occupa di:

- recuperare il valore corrente dallo *slider*;
- recuperare i dati dei worker;
- chiudere la finestra di dialogo di caricamento dei worker,
- caricare nella griglia dei worker le impostazioni di Masonry;
- impostare l’intervallo dinamico di aggiornamento dei dati;
- chiamare ricorsivamente sé stessa in modo da poter caricare i worker con l’intervallo appena letto dallo slider.

3.3.6 Gestione impostazioni DMASON

La vista *Settings* (Figura 3.3) è divenuta funzionante, estendendosi per includere due ulteriori sezioni per le impostazioni generali di DMASON e per quelle relative ad Amazon EC2; come per i worker, anche l’aspetto delle sezioni per le impostazioni è stato reso conforme alle specifiche per le *card* in Material Design (cfr. appendice “Listati”, Codice A.6).

```

907 // attach settings update logic to settings.jsp paper
    cards
908 $.ready(function () {
909 // $(loadSettings); // done in a previous function
910 $("#setgeneral").click(updateGeneralSettings);
911 $("#setactivemq").click(updateActiveMQSettings);
912 $("#setamazonaws").click(updateAmazonEC2Settings);

```

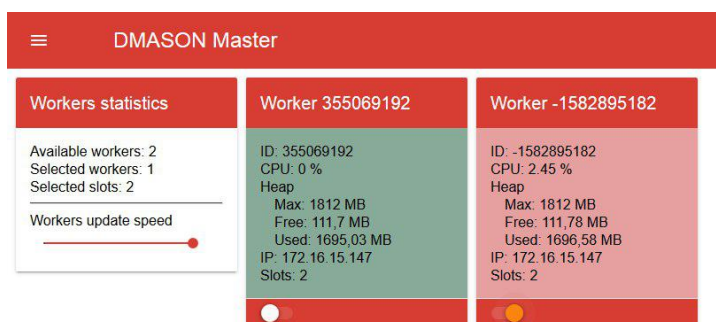



Figura 3.2: DMASON, vista *Monitoring*: nuova grafica worker e informazioni worker selezionati

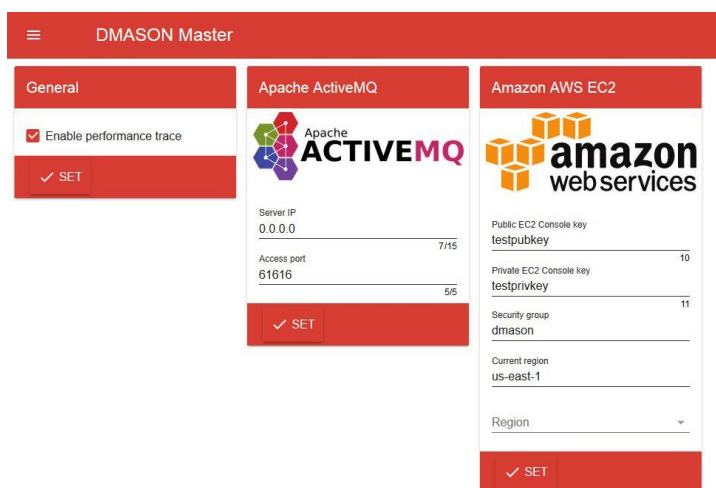


Figura 3.3: DMASON, vista *Settings*: nuova grafica e nuove impostazioni

913 }) ;

Codice 3.5: Assegnazione delle funzioni di modifica impostazioni ai bottoni delle impostazioni

Sono state definite nello script JavaScript le funzioni per la gestione delle impostazioni: in particolare, quella per mostrare le impostazioni correnti (cfr. appendice “Listati”, Codice A.7) viene eseguita al caricamento della pagina mentre quelle per la modifica (cfr. appendice “Listati”, Codice A.8) vengono associate all’evento *onclick* dei bottoni “SET” presenti in ogni sezione (Codice 3.5).

Capitolo 4

Sviluppo backend

Indice

4.1	Tecnologie utilizzate	29
4.1.1	Apache Maven	30
4.1.2	Servlet	30
4.1.3	Apache Commons Configuration	30
4.1.4	JSch	30
4.2	Sviluppo	31
4.2.1	Struttura del progetto	31
4.2.2	Gestione impostazioni	31
4.2.3	Connessioni remote	33
4.2.4	Caricamento e scaricamento di file remoti	33
4.2.5	Gestione remota di DMASON	34
4.2.6	VersionChooser	35

La parte back-end di DMASON ha subito un'attività di sviluppo consistente sia nella scrittura del modulo per la creazione e gestione di istanze EC2 su Amazon AWS sia per la gestione di particolari richieste provenienti dal lato front-end; quest'ultima attività è esposta nel presente capitolo mentre lo sviluppo del modulo di servizio per EC2 viene trattato nel Capitolo 5.

4.1 Tecnologie utilizzate

Le tecnologie qui descritte erano già in uso a DMASON; l'unica introdotta per la realizzazione del modulo di servizio è JSch.

4.1.1 Apache Maven

Come già accennato nel Capitolo 2, le dipendenze di DMASON non sono integrate manualmente nel progetto ma recuperate e gestite automaticamente da *Maven* [26], della Apache Foundation. Esso si occupa anche di eseguire degli script tramite *plugin* addizionali: si può, per esempio, prevedere l'esecuzione di un programma da linea di comando oppure generare automaticamente la documentazione; per impostazione predefinita, esegue anche eventuali test definiti all'interno del progetto.

4.1.2 Servlet

Pur essendo integrato in DMASON, il System Management può essere considerato a tutti gli effetti un progetto web a sé: nella cartella `resources\systemmanagement` di DMASON sono presenti tutti i file necessari al suo funzionamento. In quest'ottica, è stato previsto che il System Management non possa entrare direttamente in contatto con i file del back-end di DMASON ma debba usare le **Servlet** Java [41] come intermediarie; il contenitore che ne gestisce il ciclo di vita è **Jetty**, in versione *embedded*, della Eclipse Foundation.

4.1.3 Apache Commons Configuration

Alcune impostazioni di DMASON sono salvate in un file ASCII e sono necessaria sia al back-end sia al front-end che vi accede tramite richieste a servlet; la classe `Properties` del pacchetto `java.util` permette di accedere in lettura e scrittura a file di configurazione; tuttavia, essendo una classe sviluppata agli albori di Java, la scrittura dei parametri nel file non prevede il mantenimento dell'ordine in cui si trovavano in precedenza, rendendo difficile a un operatore umano la lettura del file di configurazione. A tal scopo, esiste la libreria *Apache Commons Configuration* [42] che permette una gestione più *elegante* dei file *properties*.

4.1.4 JSch

Per la connessione alla console di gestione di Amazon EC2 e alle singole istanze, sono richieste connessioni sul protocollo *Secure Shell* (`ssh`); la libreria Java più diffusa per l'implementazione di connessioni `ssh` è **JSch** [43] di JCraft, una libreria che crea sessioni sicure sia per l'esecuzione di comandi sia per il trasferimento file.

Problemi

Ogni sessione creata da JSch, alla prima connessione, viene dotata di un numero casuale (chiamato *packet*); se la sessione viene disconnessa e riconnessa continuamente oppure se si verifica una disconnessione per inattività,

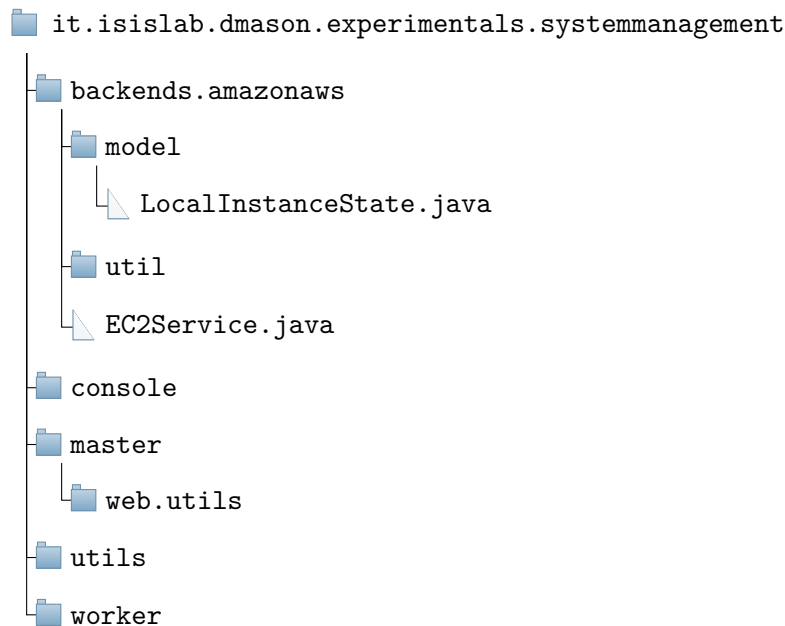


Figura 4.1: Struttura del progetto backend relativa a servlet e classi di servizio per EC2

l'oggetto `packet` viene perduto e al tentativo di riconnessione alla sessione viene lanciato l'errore `Packet corrupt`.

Questo comportamento si verifica puntualmente in caso di connessioni SFTP ma può verificarsi anche in altri casi; di conseguenza, non è possibile conservare gli oggetti sessione per una riconnessione rapida: essi vanno scartati e ricreati ad ogni connessione.

4.2 Sviluppo

4.2.1 Struttura del progetto

La struttura di classi in Figura 4.1 mostra la sezione di back-end relativa alla parte servlet e alle classi di gestione di EC2; in particolare, nel pacchetto `backends.amazonaws.util` sono presenti classi per la gestione remota dei comandi, dei file e di DMASON mentre nel pacchetto `master.web.utils` sono presenti tutte le servlet per permettere l'interazione con front-end, come quelle per la gestione di impostazioni, worker e simulazioni, per l'istanziamento di nuove istanze EC2 e così via.

4.2.2 Gestione impostazioni

Alcuni parametri di DMASON sono conservati nel file `config.properties`: inizialmente erano presenti solo indirizzo e porta per il server di messaggi-

stica **Apache ActiveMQ**, poi il file si è esteso aggiungendo un gruppo di impostazioni generali ed uno per Amazon EC2.

```
#Configurations parameters connection
ipmaster=0.0.0.0
portmaster=61616
```

Codice 4.1: File di configurazione precedente

I codici 4.1 e 4.2 mostrano, rispettivamente, le organizzazioni del precedente file di configurazione e di quello attuale.

```
# ActiveMQ connection parameters
activemq.ipmaster = 0.0.0.0
activemq.portmaster = 61616

# Amazon EC2 parameters
ec2.ami = ami-cd0f5cb6
ec2.amiuser = ubuntu
ec2.consolepubkey = testpubkey
ec2.consoleprikey = testprivkey
ec2.region = us-east-1
ec2.securitygroup = dmason
ec2.size = 1
ec2.type = t2.micro

# General settings
general.enableperftrace = true
```

Codice 4.2: File di configurazione corrente

Tramite `FileBasedConfigurationBuilder`, una classe di oggetti *builder* per le configurazioni (della libreria Apache Commons Configuration), il file viene caricato in memoria: da un oggetto builder, dandogli in input il percorso del file da leggere, viene restituito un oggetto istanza della classe `Configuration` che permette la manipolazione delle proprietà contenute nel file.

```
Parameters params = new Parameters();
FileBasedConfigurationBuilder<FileBasedConfiguration>
    builder =
    new FileBasedConfigurationBuilder<
        FileBasedConfiguration>(PropertiesConfiguration.
        class)
    .configure(
```

```
        params.properties().setFileName(  
            PROPERTIES_FILE_PATH  
        );  
  
Configuration config = builder.getConfiguration();
```

Codice 4.3: Creazione builder per le configurazioni

Le proprietà possono essere recuperate dall'oggetto `Configuration` con semplici metodi accessori come, per esempio, `getString(String)` il cui parametro esplicito è il nome della proprietà desiderata. In maniera altrettanto semplice, le proprietà possono essere modificate per mezzo del metodo `setProperty(String, String)` di `Configuration`, dove il primo parametro è il nome della proprietà da modificare mentre il secondo è il valore; una volta modificata la classe di configurazione, essa va resa persistente con l'istruzione `builder.save()`.

4.2.3 Connessioni remote

Per le connessioni remote effettuate da DMASON, è stata appositamente scritta la classe `RemoteCommand`, i cui metodi erano inizialmente parte della classe di servizio per la connessione ad Amazon EC2 e successivamente sono stati esportati per definire più nettamente le responsabilità. `RemoteCommand` prevede due metodi pubblici (uno per l'esecuzione remota dei comandi e l'altro per il recupero di una sessione esistente per l'istanza remota desiderata) e uno privato (per la lettura della console remota).

Il metodo `executeCommand(Session, String, boolean)` prende come parametri espliciti la sessione da utilizzare per l'esecuzione del comando, la stringa contenente il comando da eseguire e un valore di verità per determinare se l'esecuzione del comando debba stampare l'output remoto sulla console locale. Esso non fa altro che stabilire un canale di tipo `exec` con la sessione data, impostare sul canale il comando da eseguire e stabilire la connessione, leggendo il valore di ritorno dell'esecuzione remota.

Il metodo `retrieveSession(String)`, dato l'identificativo di una specifica istanza come parametro esplicito, si preoccupa di recuperare dalla classe di servizio per la connessione ad Amazon EC2 una nuova sessione per l'istanza data e testare la connessione prima di restituire la sessione.

4.2.4 Caricamento e scaricamento di file remoti

Le simulazioni eseguite su istanze remote restituiscono in output risultati e log; inoltre, può essere necessario mettere a loro disposizione alcuni file. La gestione remota dei file viene effettuata per mezzo della classe `DMasonRemoteFileManager`, la quale prevede due metodi pubblici per il

recupero e il caricamento da e verso istanze remote ospitate da Amazon EC2.

Il metodo `putFile(String, String, String, String)` prende come argomenti l'identificativo dell'istanza su cui caricare il file, il percorso locale di origine, il percorso remoto di destinazione e il nome del file. Dopo aver recuperato una sessione per mezzo del metodo `retrieveSession(String)` di `RemoteCommand`, viene stabilita una connessione di tipo `sftp`, impostata la directory remota specificata nel canale e inviato il file con nome e percorso specificati. Il metodo `retrieveFile(String, String, String, String)` funziona in maniera complementare a `putFile(String, String, String, String)`: l'unica differenza consta nel metodo chiamato sul canale stabilito nel momento in cui il file va inviato (`put(FileInputStream, String)`) o ricevuto (`String, String`).

4.2.5 Gestione remota di DMASON

Sempre nell'ottica della separazione di responsabilità distinte tra classi distinte, altri metodi sono stati esportati in una nuova classe diversa da quella di servizio per la connessione ad Amazon EC2: `DMasonRemoteManager` si occupa di gestire l'esecuzione di DMASON sulle istanze remote; essa prevede i seguenti metodi:

- `installDMason(String)` per l'installazione di DMASON sull'istanza remota;
- `startDMason(String, boolean)` per l'esecuzione di DMASON (come master o come worker) sull'istanza remota;
- `stopDMason(String)` per l'arresto di DMASON sull'istanza remota;
- metodi accessóri e modificatori per indirizzo e porta del server di messaggistica.

Questa classe dipende da `RemoteCommand` per l'esecuzione dei comandi remoti e da `DMasonRemoteFileManager` per la gestione dei file remoti.

Installazione remota

`installDMason(String)` prevede una prima fase di controllo dell'istanza remota in cui si verifica se essa sia in esecuzione e se non vi sia già installato DMASON; una volta superate queste verifiche (ossia, se l'istanza è in esecuzione e non ha già DMASON installato) si procede alla creazione di una sessione per le operazioni remote:

1. aggiornamento repository del gestore pacchetti (le istanze montano sistemi *GNU/Linux*, in particolare la distribuzione **Ubuntu**);

2. installazione del Java Development Kit;
3. installazione di Apache Maven;
4. download di DMASON dal repository GitHub;
5. costruzione di DMASON con Maven.

Esecuzione remota

Come nel caso del metodo per l'installazione, `startDMason(String, boolean)` prevede alcuni controlli sull'attività dell'istanza, la presenza di DMASON e il suo stato (se DMASON è già in esecuzione, il metodo restituisce il controllo al chiamante). La parte fondamentale del metodo consiste nell'esecuzione del comando per l'avvio di DMASON, il quale prevede sia la variante per l'avvio come master sia quella come worker; la discriminazione viene effettuata col parametro esplicito booleano: se è vero, DMASON viene avviato come master.

Arresto remoto

Il metodo `stopDMason(String)` prevede semplicemente l'arresto del processo della Java Virtual Machine su cui è in esecuzione DMASON, dopo aver controllato che sia in esecuzione (nel qual caso, restituisce il controllo al chiamante senza proseguire).

4.2.6 VersionChooser

Per eseguire DMASON, è necessario sapere esattamente quale versione sia installata: Maven usa la versione dichiarata nel *modello a oggetti del progetto* (**POM**, da cui il nome del file `pom.xml` che lo contiene) per la costruzione di DMASON e la generazione degli archivi eseguibili finali, secondo il formato `DMASON-X.X.jar`, dove `X.X` è la versione dichiarata nel POM. Non è possibile eseguire in remoto DMASON senza sapere quale versione sia stata compilata in remoto.

`VersionChooser` (cfr. Appendice "Listati", Codice A.11) permette di estrarre dal POM la versione dichiarata, navigando l'albero del suo DOM; il metodo `extract()` provvede all'estrazione della versione, restituendola come stringa, navigando l'albero radicato nel nodo `project`.

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
    http://maven.apache.org/xsd/maven-4.0.0.xsd">
5  <modelVersion>4.0.0</modelVersion>
```



```
<groupId>it.isislab</groupId>  
<artifactId>DMASON</artifactId>  
<version>3.2</version>  
10 <name>DMASON</name>
```

Codice 4.4: Le prime dieci righe del POM di DMASON

Capitolo 5

Amazon EC2 in DMASON

Indice

5.1	Schema generale	37
5.2	Strumenti di sviluppo	37
5.2.1	<i>AWS SDK for Java</i>	38
5.2.2	<i>AWS Toolkit for Eclipse</i>	38
5.3	Amazon EC2	39
5.3.1	Tipi di istanze	39
5.3.2	<i>Amazon Machine Images</i>	40
5.3.3	Console di gestione	40
5.4	Integrazione EC2 in DMASON	43
5.4.1	Classe di servizio EC2Service	43
5.4.2	Modelli	48
5.4.3	Integrazione front-end	48

5.1 Schema generale

Lo schema d'uso di DMASON resta il medesimo di quello mostrato al Capitolo 2 in Figura 2.1, con l'aggiunta dell'interazione tra il back-end e il servizio Amazon EC2, quasi trasparente all'utente a cui sono richiesti pochi passi per l'uso: l'inserimento delle chiavi d'accesso a EC2 e la creazione delle istanze remote da interfaccia grafica.

5.2 Strumenti di sviluppo

L'attività fondante del lavoro descritto in questa tesi è la costruzione del modulo per l'interazione di DMASON col servizio **Amazon EC2** di AWS. Le istanze su EC2 possono essere create sia dalla console grafica, accessibile

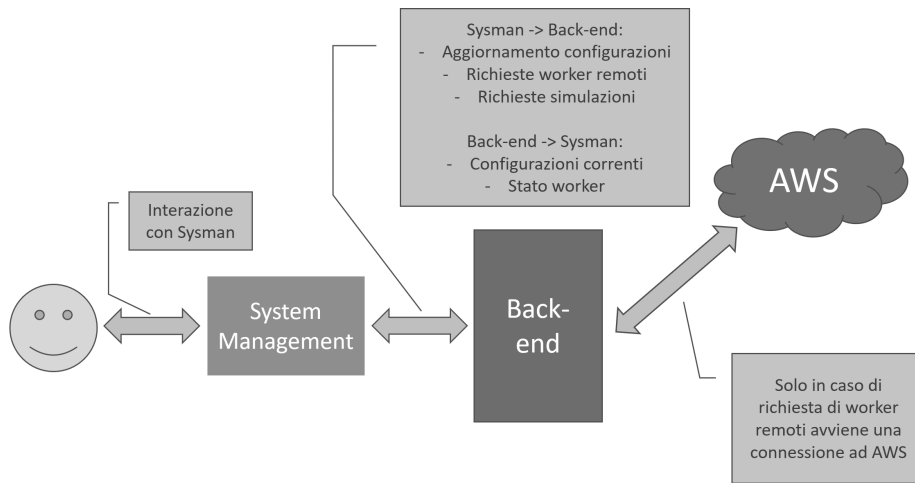


Figura 5.1: Schema d'uso di DMASON, aggiornato con l'interazione con Amazon EC2

con un browser web, sia programmaticamente tramite una API specifica che varia a seconda del linguaggio utilizzato: Java (con una API specifica anche per Android), JavaScript (sia lato server per Node.js sia lato client), .NET, PHP, Python, Ruby.

5.2.1 *AWS SDK for Java*

Come detto al Capitolo 1, DMASON è scritto in Java; per poter scrivere il modulo di richiesta di istanze su EC2, è necessario far uso di **AWS SDK for Java** [44], una collezione di API per i servizi AWS. La versione utilizzata per DMASON è la 1.11 ma nella pagina ufficiale è annunciata la versione 2, in via di sviluppo, basata su Java 8; quest'ultima prevede una profonda riscrittura della precedente versione, con un occhio di riguardo rispetto a “consistenza, immutabilità e facilità d'uso”.

5.2.2 *AWS Toolkit for Eclipse*

Durante l'attività di sviluppo, è possibile consultare la console online di EC2 per la verifica delle classi e dei metodi utilizzati; tuttavia, se come ambiente di sviluppo integrato si usa **Eclipse**, Amazon mette a disposizione un plugin, l'**AWS Toolkit for Eclipse** [45], che permette di controllare tutti gli aspetti di tutti i servizi AWS: in particolare per EC2, all'interno dello stesso ambiente di sviluppo, si possono controllare le istanze in esecuzione, i loro volumi, i gruppi di sicurezza definiti e le coppie di chiavi.

5.3 Amazon EC2

Amazon EC2 è già stato introdotto nel paragrafo 1.2.4: è il servizio di Amazon AWS che fornisce capacità di calcolo sul cloud; è progettato per essere sicuro e rendere più semplice agli sviluppatori creare applicazioni in grado di scalare su cloud a seconda del carico di lavoro. Di seguito, saranno illustrati alcuni punti chiave del servizio [46].

Innanzitutto, tra i vantaggi offerti da EC2 vi sono:

- scalabilità** è possibile variare la capacità di calcolo rapidamente nel giro di pochi minuti, potendo richiedere in simultanea molte istanze;
- pieno controllo** le istanze forniscono accesso come utente *root*, fornendo quindi all'utente il controllo assoluto; le si può anche arrestare (conservando i dati) e riavviarle tramite le API;
- ampia offerta** sono disponibili numerosi tipi di istanze con diversi sistemi operativi e applicativi preinstallati; è anche possibile scegliere varie configurazioni fisiche (CPU, memoria, *storage*) con varie opzioni anche sulla dimensione della partizione d'avvio;
- integrazione** EC2 è integrato con altri servizi AWS come S3 per lo storage RDS per il database relazionale, VPC per la rete privata, per offrire un'ampia gamma di applicazioni;
- affidabilità** l'ambiente in cui vengono eseguite le istanze EC2 è garantito dall'affidabilità dell'infrastruttura di Amazon;
- sicurezza** la priorità massima per i servizi AWS è la sicurezza, che viene offerta a quelle aziende che abbiano necessità di comunicazioni sicure anche grazie alla già citata integrazione col servizio *Amazon VPC*;
- economia** la potenza di calcolo di Amazon viene offerta a prezzi molto contenuti, molto poco rispetto alla potenza di calcolo offerta;
- facilità d'uso** le istanze sono facilmente accessibili dalla console di gestione di AWS, tramite strumenti a riga di comando o grazie agli AWS SDK; viene persino offerto un periodo di prova gratuito di un anno.

5.3.1 Tipi di istanze

Amazon EC2 offre un'ampia scelta di tipi d'istanze, ognuna dedicata a un diverso caso d'uso; tali tipologie si differenziano nel numero di CPU, nella quantità di memoria primaria e secondaria, nell'ampiezza di banda, permettendo all'utente di scegliere le caratteristiche più adatte alla propria applicazione. Ogni tipo offre a sua volta delle taglie, in modo da poter scalare le caratteristiche in base al carico di lavoro dell'applicazione.

Le istanze **T2** sono ideali per applicazioni esposte a picchi di carico (*burst*), in quanto forniscono una configurazione base di risorse che si può ampliare in caso di necessità. Le prestazioni di base e la capacità di scalare sono determinate dai **crediti CPU**: le istanze T2 ricevono continuamente una quantità fissa di crediti, variabile in base alla taglia, accumulandoli quando l'istanza sia inattiva e spendendoli altrimenti. La tipologia T2 è ideale per l'allestimento di microservizi, gestione di database di taglia piccola o media, desktop virtuali, repository di codice eccetera.

Le istanze **M5** sono quelle di ultima generazione, adatta a usi generici; un buon compromesso di risorse, costituiscono una buona scelta per varî tipi di applicazioni. Prima di esse vi erano le istanze **M4**.

Le istanze **C5** sono ideali per applicazioni che richiedono grande capacità di calcolo, garantendo alte prestazioni a basso costo.

Le istanze **P3** sono ideali per applicazioni che fanno uso di GPU (come *machine learning*, calcolo ad alte prestazioni o calcolo sulla dinamica dei fluidi); le istanze **G3** sono ottimizzate per applicazioni che fanno uso intensivo di grafica (visualizzazione e *rendering* 3D, codifica video); le istanze **F1** forniscono accelerazione hardware per mezzo dei *field programmable gate arrays* (FPGA).

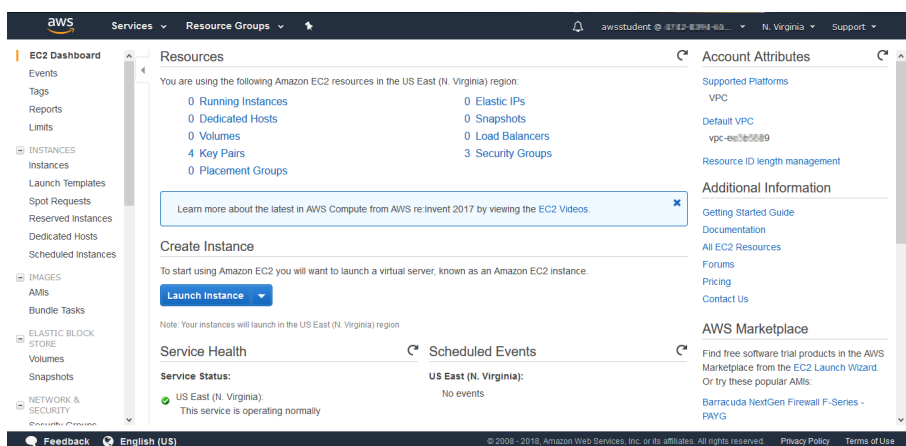
Vi sono molti altri tipi di istanze offerti da Amazon EC2 sia per scopi generali sia ottimizzati per l'uso di una specifica risorsa (memoria primaria, memoria secondaria, calcolo).

5.3.2 Amazon Machine Images

Le istanze EC2 alla creazione non subiscono un processo di installazione di sistema operativo e applicazioni ma viene loro caricata un'immagine di un sistema già pronto che prende il nome di **Amazon Machine Image** (AMI). Le AMI sono preconfigurate e si può scegliere tra una vasta gamma di sistemi (Windows, GNU/Linux e altri basati su Unix).

5.3.3 Console di gestione

La console di gestione di Amazon AWS (figure 5.2 e 5.3) permette l'accesso alle impostazioni dei varî servizi AWS disponibili col proprio account; in particolare, la console di gestione di EC2 permette a colpo d'occhio, selezionata la regione d'interesse, di sapere non solo quante istanze, quante coppie di chiavi e quanti gruppi sono definiti ma anche lo stato del servizio nella regione scelta, globale e specifico per ogni server presente. Come detto a inizio capitolo, dalla console è possibile lanciare una nuova istanza definendo AMI, tipo d'istanza, volumi di memoria secondaria e il gruppo di sicurezza.



(a) Console principale Amazon EC2

Service Health

Service Status:

- ✓ US East (N. Virginia):
This service is operating normally

Availability Zone Status:

- ✓ us-east-1a:
Availability zone is operating normally
- ✓ us-east-1b:
Availability zone is operating normally
- ✓ us-east-1c:
Availability zone is operating normally
- ✓ us-east-1d:
Availability zone is operating normally
- ✓ us-east-1e:
Availability zone is operating normally
- ✓ us-east-1f:
Availability zone is operating normally

[Service Health Dashboard](#)

Resources

You are using the following Amazon EC2 resources in the US East (N. Virginia) region:

0 Running Instances	0 Elastic IPs
0 Dedicated Hosts	0 Snapshots
0 Volumes	0 Load Balancers
4 Key Pairs	3 Security Groups
0 Placement Groups	

(b) Stato della regione (c) Risorse utilizzate dall'utente nella regione scelta

Figura 5.2: Pagina principale della console di gestione di Amazon EC2; nella sezione *Resources* (5.2c) sono mostrati i dettagli di istanze in esecuzione, coppie di chiavi e gruppi di sicurezza disponibili mentre nella sezione *Service Health* (5.2b) è indicato lo stato di salute globale della regione e dei singoli server di cui si compone

Security Group: sg-54772727

Name	Group ID	Group Name	VPC ID	Description
<input type="checkbox"/>	sg-30110442	dmason	vpc-ea700009	Security group dmason created on 2018-01-24T13:08:10...
<input checked="" type="checkbox"/>	sg-54772727	isislab-dmason	vpc-ea700009	Security group isislab-dmason created on 2017-09-29T13:...
<input type="checkbox"/>	sg-a19111e	default	vpc-ea700009	default VPC security group

Security Group: sg-54772727

Description | Inbound | Outbound | Tags

Group name: isislab-dmason
Group description: Security group isislab-dmason created on 2017-09-29T13:03:01.861.
Group ID: sg-54772727
VPC ID: vpc-ea700009

(a) Gruppi di sicurezza

Key Pair: dmason-key

Key pair name	Fingerprint	
<input type="checkbox"/>	cluster-red-key	44:e8:c2:c2:85:ab:55:46:08:11:c9:74:8e:8f:c9:a2:8d:82:c8:e9
<input checked="" type="checkbox"/>	dmason-key	92:e6:b8:b8:4c:70:de:ad:63:67:9e:76:8c:71:df:d9:45:9d:80:b8
<input type="checkbox"/>	dmason-key-2	d9:3b:55:55:66:e9:71:c3:ab:8a:87:87:88:64:66:3e:85:20:bf:58
<input type="checkbox"/>	qwikLABS-L265-1663259	38:e8:8e:2b:85:a3:58:b2:9c:8f:d1:29:7c:e4:88:75

Key Pair: dmason-key

Key pair name: dmason-key
Fingerprint: 92:e6:b8:b8:4c:70:de:ad:63:67:9e:76:8c:71:df:d9:45:9d:80:b8

(b) Coppie di chiavi

Figura 5.3: Impostazioni gruppi di sicurezza (5.3a) e coppie di chiavi (5.3b) nella console di EC2

Security Group

Un *security group* definisce un firewall per il controllo del traffico da e verso le istanze che vi appartengono. In un gruppo di sicurezza possono essere specificate regole di accesso a un'istanza su una determinata porta o per un determinato servizio; esse possono anche essere aggiunte dopo la definizione del gruppo ed entrare subito in vigore per le istanze appartenenti.

Coppia di chiavi

L'accesso alle istanze avviene attraverso la *crittografia a chiave pubblica* per crittare e decrittare le informazioni di accesso. Per effettuare l'accesso presso un'istanza – o anche per lanciarla – è necessario far uso di una coppia di chiavi; Ad ognuna di esse è associato un nome, necessario per poter accedere a un'istanza.

Le chiavi possono essere create in EC2 oppure con uno strumento esterno ed importate; se create tramite EC2, Amazon conserverà la sola chiave pubblica mentre quella privata sarà mandata all'utente all'atto della creazione. Le chiavi usate da EC2 sono chiavi **RSA a 2048 bit**; in ogni regione possono essere definite fino a cinquemila coppie di chiavi.

5.4 Integrazione EC2 in DMASON

5.4.1 Classe di servizio EC2Service

Il modulo di connessione ad EC2 è realizzato per mezzo di una classe di servizio, `EC2Service`, che implementa metodi statici per la creazione di istanze. Lo stato della classe si compone delle seguenti proprietà:

AMI il codice della AMI da caricare sulle istanze da creare;

utente AMI il nome dell'utente con accesso alla macchina (possibilmente non *root*);

controllo avvio una proprietà per verificare se la classe sia già stata inizializzata;

nome gruppo il nome del *security group* a cui afferiscono le macchine;

regione il codice della regione in cui vanno create le macchine;

taglia il numero di istanze da creare;

client EC2 il client locale che permette l'interazione con il servizio EC2;

dati locali istanze per evitare frequenti richieste remote, alcuni dati immutabili come il nome dell'istanza vengono salvati in memoria.

5.4. INTEGRAZIONE EC2 IN DMASON. AMAZON EC2 IN DMASON

Vi sono anche dei valori costanti preconfigurati come il prefisso per il nome del gruppo, il nome della chiave, il percorso del file di proprietà.

I comportamenti previsti dalla classe di servizio sono:

- **avvio**: le operazioni necessarie prima di poter interagire con EC2¹ quali
 - lettura proprietà dal file `config.properties`;
 - creazione client EC2;
 - creazione coppia di chiavi per istanza;
 - creazione *security group*;
- **creazione *security group***: viene creato il gruppo col nome specificato, se già non esiste;
- **creazione coppia di chiavi**: viene creata una nuova coppia di chiavi, se già non esiste; questa fase è l'unico momento in cui Amazon invia la chiave privata, salvandola nella cartella `.aws`;
- **gestione istanze**: operazioni per la creazione, avvio, arresto, terminazione di istanze²;
- **lettura e scrittura proprietà**: i metodi accessóri e modificatori per le proprietà della classe più altri metodi accessori per oggetti delle istanze come il DNS pubblico oppure l'oggetto sessione.

Di seguito, si analizzano alcune operazioni più rilevanti nella gestione di EC2.

Creazione client EC2

La creazione del client EC2 avviene tramite un oggetto *builder* su cui si possono configurare alcune proprietà generali del client da creare; il metodo `buildEC2Client(String aRegion)` prevede quanto segue:

```
136 EC2Service.ec2 = AmazonEC2ClientBuilder.standard()
137     .withRegion(aRegion)
138     .withCredentials(new ProfileCredentialsProvider())
139     .build();
```

Codice 5.1: Corpo del metodo `buildEC2Client(String)`

¹cfr. Appendice “Listati”, Codice A.12

²La differenza tra *arresto* e *terminazione* consta nel fatto che nel primo caso l'istanza è riavviabile e i dati vengono preservati, nel secondo caso (a meno dell'uso di servizi di *storage* esterni) no.

Il corpo del metodo `buildEC2Client()` di `EC2Service` è riportato nel Codice 5.1: il metodo statico `standard()` a riga 136 costruisce un builder con impostazioni predefinite; i metodi seguenti sono invocati a catena (si parla infatti di *chaining*) perché sono invocati su una classe e restituiscono l'oggetto stesso su cui sono invocati il cui stato è cambiato secondo quanto previsto dal metodo invocato: a riga 137, il metodo `withRegion(String)` imposta la regione di residenza delle istanze, a riga 138, `withCredentials(ProfileCredentialsProvider)` imposta il provider delle credenziali per l'accesso ad EC2 mentre a riga 139, il metodo `build()` restituisce l'oggetto `EC2Client` con le caratteristiche specificate dai metodi precedenti.

Creazione coppia di chiavi

Per l'accesso alle istanze create su EC2, è necessaria una coppia di chiavi; il metodo `createKeyPair()` si occupa di garantire che una coppia di chiavi per le istanze create dal client esista, ossia se già non ne esista una la crea con il nome impostato nel file di configurazione.

```
286 CreateKeyPairRequest keyRequest = new
    CreateKeyPairRequest().withKeyName(EC2Service.
        KEY_NAME);
287 CreateKeyPairResult responseToCreate = EC2Service.ec2.
    createKeyPair(keyRequest);
```

Codice 5.2: Richiesta coppia di chiavi in `createKeyPair()`

Nel Codice 5.2 vengono mostrate le due istruzioni chiave nella creazione della coppia di chiavi: con la prima si imposta il nome (salvato nella costante `KEY_NAME`), con la seconda si ottiene il risultato della richiesta.

```
292 filePrinter.print(responseToCreate.getKeyPair().
    getKeyMaterial());
```

Codice 5.3: Salvataggio su file della coppia di chiavi

Nel Codice 5.3, la coppia di chiavi viene resa persistente su file per poter essere riutilizzata per le richieste remote su istanze.

Creazione istanza

Il metodo `createInstance(String, String, int)` si occupa della creazione di istanze remote, nel gruppo e nel numero specificati come parametri espliciti; nel Codice 5.4 (cfr Appendice “Listati”, Codice A.13) viene ripor-

5.4. INTEGRAZIONE EC2 IN DMASON

tata la prima sezione del metodo, quella in cui avviene la richiesta vera e propria. All'atto della creazione, le istanze risultano già avviate.

```
208 RunInstancesRequest instanceRequest = new
    RunInstancesRequest();
209
210 // define the requested virtual machine
211 String instanceType = EC2Service.type;
212 instanceRequest.withImageId(EC2Service.ami)
213     .withInstanceType(instanceType)
214     .withMinCount(1) // the minimum number of instances to
        launch
215     .withMaxCount(maxNumberInstances) // the maximum
        number of instances to launch
216     .withKeyName(EC2Service.KEY_NAME)
217     .withSecurityGroups(groupName);
218
219 // request the virtual machine
220 RunInstancesResult result = ec2.runInstances(
    instanceRequest);
```

Codice 5.4: Sezione di `createInstance(String, String, int)` in cui viene richiesta un'istanza EC2

Le righe 212-215 riportano la creazione della richiesta con tutti i parametri necessari: `withImageId(String)` imposta l'AMI di cui l'istanza dev'essere copia, `withInstanceType(String)` imposta il tipo di istanza, `withMinCount(int)` e `withMaxCount` impostano quante istanze vanno create, `withKeyName(String)` imposta il nome della coppia di chiavi da usare e `withSecurityGroups(String)` imposta a quale gruppo debbano appartenere le istanze richieste.

Gestione istanze

Le istanze remote possono essere avviate, arrestate, riavviate e terminate; in `EC2Service` sono previsti rispettivamente i metodi `startInstance(String)` (cfr. appendice "Listati", Codice A.14), `stopInstance(String)`, `rebootInstance(String)` e `terminateInstance(String)` dove l'argomento comune di tutte è l'ID dell'istanza su cui agire. Essi riportano tutti la stessa struttura:

- controllo stato corrente;
- richiesta variazione stato;
- attesa variazione di stato;

- eventuale persistenza dello stato (per le istanze terminate, in modo che non siano più considerate in locale).

La parte fondamentale di questi metodi è la richiesta che viene effettuata in tutti i casi con un metodo sull'oggetto `EC2Client`: si crea prima la richiesta di azione per un gruppo di istanze desiderate, invocando sull'oggetto richiesta il metodo `withInstanceIds(String)` – che può avere come argomento una o più stringhe – e poi si invoca sul client EC2 il metodo `xxxInstance(xxxInstanceRequest)`, dove `xxx` sta per `start`, `stop`, `reboot` oppure `terminate`.

Istanze spot

Le istanze spot vengono richieste in maniera simile a quelle on demand, con la differenza che piuttosto che essere eseguite vengono richieste con dei parametri specificati.

Stato La richiesta di istanze spot può trovarsi in uno dei seguenti stati:

`open` la richiesta è in vigore e attende di essere esaudita;

`active` la richiesta è stata esaudita e ha istanze spot associate;

`failed` la richiesta è difforme, ha alcuni parametri non validi;

`closed` la richiesta è stata conclusa o interrotta;

`cancelled` la richiesta è stata cancellata o è scaduta

Richiesta Il metodo di `AmazonEC2Client` per richiedere istanze spot è `RequestSpotInstances(RequestSpotInstancesRequest)`, il cui argomento oggetto della classe `RequestSpotInstancesRequest` viene sempre configurato con la concatenazione di metodi: in questo modo è possibile, col metodo `withSpotPrice(String)` impostare il prezzo massimo orario che si intende spendere, con `withValidFrom(Date)` si imposta il momento in cui si intende iniziare a partecipare all'asta spot e con `withValidUntil(Date)` il momento a partire dal quale non si ha più bisogno dell'istanza spot.

L'istanza spot viene lanciata nel momento in cui il prezzo massimo specificato è maggiore del prezzo spot e termina se interrotto da EC2 o dall'utente. Per sapere se una richiesta spot viene esaudita, si può ciclare sul metodo `describeSpotInstanceRequests(DescribeSpotInstanceRequestsRequest)`, nel cui oggetto passato come argomento sia stato impostato l'ID di una richiesta precedentemente effettuata finché lo stato restituito non sia `open`.

Durata Se nelle richieste di istanze spot è specificata una durata, Amazon EC2 non le termina: si può richiedere istanze di durata da 1 fino a sei ore e il tempo viene calcolato nel momento in cui alla richiesta viene associato l'ID di un'istanza e termina o quando viene interrotta dall'utente o allo scadere del tempo specificato (nel secondo caso, EC2 invia una notifica di terminazione con un preavviso di due minuti). Tali istanze sono chiamate **Spot block**.

5.4.2 Modelli

L'accesso rapido ai dati delle istanze è realizzato in locale per mezzo della classe `LocalInstanceState`, il cui stato prevede come proprietà ID, DNS pubblico, tipo e stato dell'istanza (se è in esecuzione o se è terminata), lo stato di DMASON sulla macchina (installato, in esecuzione, modalità di esecuzione ed eventuale DNS del master) e la data di ultima modifica, la quale non prevede un metodo modificatore ma l'aggiornamento tramite un metodo privato invocato da tutti gli altri metodi modificatori.

Gli oggetti `LocalInstanceState` di istanze esistenti vengono conservati da `EC2Service` in una mappa in cui la chiave è l'ID dell'istanza.

5.4.3 Integrazione front-end

Le richieste di nuove istanze da parte del front-end avvengono con richieste HTTP alla servlet `InstantiateEC2WorkersServlet` (cfr. Appendice "Listati", Codice A.10), elaborate secondo le seguenti fasi:

1. controllo parametri richiesta;
2. configurazione iniziale classe `EC2Service` (nel caso non fosse già inizializzata);
3. installazione DMASON col metodo `installDMason(String)` della classe `DMasonRemoteManager` (descritta al paragrafo 4.2.5);
4. avvio di DMASON col metodo `StartDMason(String, boolean)` della medesima classe.

Capitolo 6

Conclusioni

6.1 Lavoro svolto

In questa tesi, è stato presentato il toolkit *single-process* MASON per le simulazioni multi-agente a eventi discreti; il suo limite è il non prevedere la distribuzione delle simulazioni su più macchine.

Quest'importante lacuna viene colmata da DMASON, che aggiunge uno strato ai tre già previsti di MASON – Modello, Vista e Utilità – per il partizionamento dell'istanza al modello di simulazione in parti (che possono essere a due a due uguali o meno) tra più processori logici, i quali sono in comunicazione con una macchina centrale secondo il paradigma *master-worker*; il master monitora l'andamento della simulazione e, per impostazione predefinita, si occupa lui di effettuare il partizionamento in regioni. DMASON può essere gestito sia tramite riga di comando sia attraverso uno strato front-end, il *System Management*, che permette di arrestare i processori logici collegati, avviare, interrompere e monitorare le simulazioni, consultare i risultati di quelle già terminate.

DMASON si occupa solo del partizionamento dell'istanza della simulazione; l'allestimento dei vari processori logici va effettuato manualmente dall'operatore; ciò costituisce un'ulteriore limitazione, in quanto bisogna disporre di un cluster di computer; ciò non permette a chi non abbia la disponibilità strutturale (di laboratori informatici, per esempio) o economica per allestirne uno di poter usufruire dei vantaggi in termini di tempi di esecuzione offerti da DMASON.

Il *cloud computing* è la soluzione a questo problema: è un servizio in evoluzione fin dagli anni Novanta, giunto a un tale livello di sviluppo da essere, ai giorni nostri, accessibile da chiunque a prezzi estremamente contenuti rispetto a quanto offerto. Leader del campo è sicuramente Amazon con il suo ecosistema *Amazon Web Services*, il cui servizio *Amazon EC2* è proprio l'emblema di cloud computing. Il vantaggio principale di DMASON, congiuntamente alla possibilità di eseguire simulazioni in modo economico

grazie ai servizi cloud, permetterebbe potenzialmente a chiunque di poter svolgere simulazioni distribuite.

L'obiettivo principale del lavoro qui presentato è stato proprio rendere DMASON in grado di usufruire delle potenzialità del servizio Amazon EC2 grazie a un modulo che lo interfacci direttamente con esso. Per fare ciò, è stato necessario prima rendere il System Management nuovamente funzionante: le dipendenze del front-end – in primis la libreria *Polymer* – non erano al passo con le capacità delle attuali versioni di navigatori web; approfittando del lavoro sul front-end, è stata lievemente rivista anche l'esperienza utente, rendendo funzionante la consultazione e la modifica delle attuali impostazioni e aggiornando la grafica dei worker.

Grazie all'*AWS SDK for Java* – una collezione completa di API per tutti i servizi AWS – è stato possibile scrivere la classe di servizio `EC2Service` che si preoccupa di impostare i parametri fondamentali per la richiesta di istanze (regione, tipo, sistema operativo) ed espone dei metodi per la loro gestione; la creazione di istanze EC2 – siano esse di tipo *spot* oppure *on demand* – tramite il System Management passa tramite la servlet `InstantiateEC2WorkersServlet`.

6.2 Sviluppi futuri

La creazione del modulo per l'esecuzione di DMASON su EC2 è un processo con ampi margini di miglioramento, che dà anche nuovi spunti di sviluppo per ulteriori funzionalità future.

Al momento, per l'interazione con Amazon EC2 viene usato un client *sincrono*, il che comporta la necessità di doverlo interrogare per essere al corrente dello stato attuale delle istanze; l'API per EC2 offre una classe builder per client asincroni, il che renderebbe più efficiente l'interazione con EC2 in termini di connessioni remote e quindi di tempo di esecuzione.

Il lavoro svolto per l'installazione di DMASON può essere generalizzato prevedendo l'esecuzione di comandi remoti: la classe `RemoteCommand`, oltre ad astrarre da `EC2Service` la gestione dei comandi remoti, ha proprio lo scopo di permettere ciò.

Al momento, è solo possibile richiedere istanze spot ma non è prevista una vera e propria gestione di tale richieste: in caso di carichi di lavoro di durata stimata superiore alle sei ore (per cui non è possibile usare istanze *spot block*), è necessario prevedere una politica di variazione del prezzo massimo offerto per le istanze spot che sia il più efficiente ed economico possibile.

La gestione delle coppie di chiavi per le istanze EC2 può essere migliorata, offrendo all'utente la possibilità di utilizzarne di già esistenti e collegarsi a eventuali istanze già in esecuzione.

Parte II

Appendici

Appendice A

Listati

Indice

A.1 Front-end	i
A.1.1 Esempio Frammento JSP: <code>header.jsp</code>	i
A.1.2 Aspetto dei worker	ii
A.1.3 Aggiornamento informazioni worker	iv
A.1.4 Scheda impostazioni <i>General</i>	v
A.1.5 Gestione impostazioni	vi
A.2 Back-end	viii
A.2.1 Comandi remoti	viii
A.2.2 Gestione richieste istanze EC2 da System Management	viii
A.2.3 Estrazione versione DMASON da modello di progetto di Maven	x
A.2.4 Classe di servizio <code>EC2Service</code>	xi

A.1 Front-end

A.1.1 Esempio Frammento JSP: `header.jsp`

```
<!-- tag import -->
<%@ taglib
  prefix="c"
  uri="http://java.sun.com/jsp/jstl/core" %>
5 <app-header reveals fixed slot="header">
  <app-toolbar flex id="mainToolBar" class="horizontal">
    <paper-icon-button icon="menu" onclick="drawer.toggle
      ()" drawer-toggle></paper-icon-button>
```

```

    <div class="flex" spacer main-title><span>DMASON
      Master</span></div>
    <c:choose>
10    <c:when test="{param.page == 'index' }">
      <div onclick="selectAllWorkers()" class="
        selectAllWorker">
        <paper-icon-button icon="select-all"></paper-
          icon-button><span>Select all workers</span>
      </div>
    </c:when>
15    <c:when test="{param.page == 'history' }">
      <div onclick="cleanHistory()" class="
        cleanAllHistory">
        <paper-icon-button icon="select-all"></paper-
          icon-button><span>Clean all history</span>
      </div>
    </c:when>
20  </c:choose>
    </app-toolbar>
  </app-header>

```

Codice A.1: Frammento dell'*header* di DMASON

A.1.2 Aspetto dei worker

Senza template

```

node = $("<div id="+w.workerID+" class=\"grid-item-
  monitoring\" onclick=\"selectItem(this)\"></div>");
// node.append($("<div class=\"worker-system-info\"><span
  >Worker ID: "+ w.workerID+"</span></div>"));
node.append($("<div class=\"worker-system-info\"><span id
  =\"w-cpu-"+w.workerID+"\">CPU: "+w.cpuLoad+" %</span
  ></div>"));
185 node.append($("<div class=\"worker-system-info\"><span>
  Heap:</span></div>"));
node.append($("<div class=\"worker-system-info\"><span id
  =\"w-max-heap-"+w.workerID+"\" class=\"tab\">Max "+w.
  maxHeap+" MB</span></div>"));
node.append($("<div class=\"worker-system-info\"><span id
  =\"w-heap-avaialable-"+w.workerID+"\" class=\"tab\">
  Free "+w.availableheapmemory+" MB</span></div>"));
node.append($("<div class=\"worker-system-info\"><span id
  =\"w-heap-used-"+w.workerID+"\" class=\"tab\">Used "+
  w.busyheapmemory+" MB</span></div>"));

```

```

node.append($("#<div class=\"worker-system-info\"><span id
  =\"w-ip-\"+w.workerID+\">IP: \"+w.ip+\"</span></div>"))
;
190 node.append($("#<div class=\"worker-system-info\"><span id
  =\"w-slots-\"+w.workerID+\">Slots: \"+ w.slots+\"</span
  ></div>"));

$(grid).append(node);

```

Codice A.2: Creazione dei worker senza template (da script.js)

Con template

```

// retrieve and populate worker template
310 var html;
// this call must be synchronous or else callbacks
// containing the same node will cumulate and insert
// it several times
$.ajax({
315  url: "../fragments/worker.html",
  async: false, // DO NOT MAKE ASYNC
  success: function (value) {
    var workerTemplate = $.templates(value);
    html = workerTemplate.render(workerData);
320
    // inject populated worker template
    $(grid).append(html);
  }
});

```

Codice A.3: Creazione dei worker con template (da script.js)

```

<paper-card id="{:workerID}" class="grid-item-
  monitoring" heading="Worker {:workerID}" onclick="
  selectItem(this)">
<div class="card-content">
  <div class="worker-system-info">
    <span>ID: <span class="worker-data">{:workerID}</
    span></span>
5  </div>
  <div class="worker-system-info">
    <span id="w-cpu-{:workerID}">CPU: <span class="
    worker-data">{:workerCPU}</span> %</span>
  </div>

```

```

    <div class="worker-system-info">
10   <span>Heap</span>
    </div>
    <div class="worker-system-info">
      <span id="w-max-heap-{{:workerID}}" class="tab">Max:
        <span class="worker-data">{{:workerMaxMB}}</span>
          MB</span>
    </div>
15   <div class="worker-system-info">
      <span id="w-heap-available-{{:workerID}}" class="tab
        ">Free: <span class="worker-data">{{:workerFreeMB
          }}</span> MB</span>
    </div>
    <div class="worker-system-info">
      <span id="w-heap-used-{{:workerID}}" class="tab">Used
        : <span class="worker-data">{{:workerUsedMB}}</
          span> MB</span>
20   </div>
    <div class="worker-system-info">
      <span id="w-ip-{{:workerID}}">IP: <span class="worker
        -data">{{:workerIP}}</span></span>
    </div>
    <div class="worker-system-info">
25   <span id="w-slots-{{:workerID}}">Slots: <span class="
        worker-data">{{:workerSlots}}</span></span>
    </div>
  </div>

  <div class="card-actions">
30   <div>
      <paper-toggle-button class="toggle" noink></paper-
        toggle-button>
    </div>
  </div>
</paper-card>

```

Codice A.4: Template per i worker

A.1.3 Aggiornamento informazioni worker

```

90 var dynDelay = 750; // default delay 0.75s
   function loadWorkersDynamicInterval() {
     var slider = document.querySelector("#update-speed");
     setTimeout(
       function () {
95       // retrieve existing workers data

```

```

        loadWorkers();

        // close loader
        if ($('#load_workers_dialog').prop("opened")) {
100     close_dialog_by_ID("load_workers_dialog");
        }

        // use Masonry on workers grid
        load_tiles_monitoring();
105

        // retrieve update speed value
        dynDelay = slider.value;
        if (!dynDelay) {
110     // set 0.75s interval while slider object loads
        dynDelay = 750;
        }

        // this last call lets setTimeout() with variable
        // to act like a setInterval() with variable delay
115     loadWorkersDynamicInterval();
    },
    dynDelay
);
}

```

Codice A.5: Funzione aggiornamento variabile dati worker

A.1.4 Scheda impostazioni *General*

```

<!-- General settings --%>
40 <paper-card heading="General" class="grid-item-settings">
    <div class="card-image"></div>
    <div class="card-content">
        <paper-checkbox id="enableperftrace" noink>Enable
            performance trace</paper-checkbox>
        <paper-tooltip for="enableperftrace" position="bottom
            " animation-delay="0" offset="1">This option
            allows to trace running simulations performance
            in workers consoles</paper-tooltip>
45     </div>
    <div class="card-actions">
        <div class="horizontal justified">
            <paper-button id="setgeneral" raised><iron-icon icon
                ="check"></iron-icon>&nbsp;Set</paper-button>
        </div>
50 </div>

```

</paper-card>

Codice A.6: Esempio di scheda di impostazioni

A.1.5 Gestione impostazioni

Recupero

```
915 function loadSettings() {
    $.post(
        "getSettings",
        function (data) {
            if (data == null) {
920         console.warn("No setting has been retrieved!");
                return;
            //} else {
            //    console.log("Settings successfully retrieved
                !")
            }
925
            // extract data
            var performanceTrace = data.generalSettings.
                enablePerfTrace;
            //console.log("Performance trace: " +
                performanceTrace); // TODO comment after
                testing

930         var activeMQIp = data.activeMQSettings.ip;
            var activeMQPort = data.activeMQSettings.port;
            //console.log("ActiveMQ location: " + activeMQIp +
                ":" + activeMQPort);

            var amazonAWSSecurityGroup = data.amazonAWSSettings
                .securityGroup;
935         var ec2ConsolePriKey = data.amazonAWSSettings.
                ec2PriKey;
            var ec2ConsolePubKey = data.amazonAWSSettings.
                ec2PubKey;
            var amazonAWSRegion = data.amazonAWSSettings.region
                ;

            // show general settings
940         performanceTrace = performanceTrace.toLowerCase();
            var perfTraceCheckbox = document.querySelector("#
                enableperftrace");
            if (performanceTrace == "true") {
```

```

        perfTraceCheckbox.checked = true;
    } else if (performanceTrace == "false") {
945     perfTraceCheckbox.checked = false;
    } else {
        console.error("Illegal value for performance
            trace setting!");
    }

950     // show ActiveMQ settings
    $("#activemqip").val(activeMQIp);
    $("#activemqport").val(activeMQPort);

    // show Amazon AWS settings
955     $("#securitygroup").val(amazonAWSSecurityGroup);
    $("#curregion").val(amazonAWSRegion); // as today,
        paper-dropdown-menu is unsettable
    $("#ec2pubkey").val(ec2ConsolePubKey);
    $("#ec2prikey").val(ec2ConsolePriKey);
    }
960 )
    .fail(function () {
        console.error("Error while retrieving current
            settings!");
    });
}

```

Codice A.7: Funzione di caricamento impostazioni correnti

Aggiornamento

```

function updateGeneralSettings() {
    var perfTrace = document.querySelector("#
        enableperftrace").checked;
    console.log("Enable performance trace: " + perfTrace);

970     // send POST request to server
    $.post(
        "updateSettings",
        {
            "setting": "general",
975     "enableperftrace": perfTrace
        }
    )
    .fail(function () {
        console.error("Error while sending general data!")
980     });
}

```

}

Codice A.8: Esempio di aggiornamento impostazioni (scheda *General*)

A.2 Back-end

A.2.1 Comandi remoti

```

public static int executeCommand(Session session, String
    command, boolean printRemoteOutput)
    throws JSchException, IOException,
        InterruptedException
{
50   ChannelExec channel = (ChannelExec) session.openChannel
        ("exec");
        channel.setInputStream(null);
        InputStream in = channel.getInputStream();
        // ((Channel) channel).setOutputStream(System.out);
        // channel.setErrStream(System.err);
55   channel.setCommand(command);
        channel.connect(RemoteCommand.CONNECTION_TIMEOUT);
        int exitStatus = RemoteCommand.readRemoteInput(channel,
            in, printRemoteOutput);
        channel.disconnect();
60   in.close();

        return exitStatus;
}

```

Codice A.9: Metodo per l'esecuzione di comandi remoti dati nella sessione data

A.2.2 Gestione richieste istanze EC2 da System Management

```

// helper methods
private void newEC2Instance(HttpServletRequest request,
    HttpServletResponse response)
{
    // comment or properly edit following line to enable
    logging
80   LOGGER.setLevel(Level.ALL);

```



```
// extract data from request
String ec2Type = request.getParameter("instancetype");
if (ec2Type == null || ec2Type == "")
85  {
    LOGGER.severe("EC2 type is null or invalid!");
    return;
}

90  int numInstances = 0;
    try
    {
        numInstances = Integer.parseInt(request.getParameter
            ("numinstances"));
    }
95  catch (NumberFormatException e)
    {
        LOGGER.severe(e.getClass().getSimpleName() + ": " + e
            .getMessage() + ".");
    }

100  LOGGER.info("Received " + numInstances + " new " +
        ec2Type + " EC2 instance request");

// create instance on EC2
RunInstancesResult instancesResult = null;
EC2Service.setType(ec2Type);
105  EC2Service.boot();
    try
    {
        instancesResult = EC2Service.createInstance(
            numInstances);
    }
110  catch (IOException e)
    {
        LOGGER.severe(e.getClass().getSimpleName() + ": " + e
            .getMessage() + ".");
    }
    Iterator<Instance> instanceIterator = instancesResult.
        getReservation().getInstances().iterator();
115  List<String> instanceIds = new Vector<>();
    while (instanceIterator.hasNext())
    {
        String instanceId = instanceIterator.next().
            getInstanceId();
        LOGGER.info("Generated instance " + instanceId);
120  instanceIds.add(instanceId);
    }
}
```

```

    // run remote instance(s)
    for (String instanceId: instanceIds)
125  {
        EC2Service.startInstance(instanceId);
    }

    // install DMASON on remote instances
130  for (String instanceId: instanceIds)
    {
        DMasonRemoteManager.installDMason(instanceId);
    }

135  // run DMASON as worker on remote instance
    MasterServer server = (MasterServer) request.
        getServletContext().getAttribute("masterServer");
    DMasonRemoteManager.setActiveMQIP(server.getActiveMq
        ());
    DMasonRemoteManager.setActiveMQPort(server.
        getPortActivemq());
    LOGGER.info(
140  "Worker(s) will connect to ActiveMQ@" +
        DMasonRemoteManager.getActiveMQIP() +
        ":" + DMasonRemoteManager.getActiveMQPort() + "..."
    );
    for (String instanceId: instanceIds)
    {
145  DMasonRemoteManager.startDMason(instanceId, false);
    }

    LOGGER.info("Remote request for " + numInstances + " "
        + ec2Type + " instances has been evaded.");
}

```

Codice **A.10:** Metodo ausiliario
 richiamato dai metodi doGet(HttpServletRequest, HttpServletResponse) e
 doPost(HttpServletRequest, HttpServletResponse) della servlet
 InstantiateEC2WorkersServlet.

A.2.3 Estrazione versione DMASON da modello di progetto di Maven

```

20 public class VersionChooser
    {
        public static String extract()
            throws ParserConfigurationException, SAXException,
                IOException
    }

```

```

    {
25     String version = null;
        DocumentBuilderFactory dbf = DocumentBuilderFactory.
            newInstance();
        DocumentBuilder db = dbf.newDocumentBuilder();
        Document doc = db.parse(new File(POM_PATH));

30     Node projectNode = doc.getFirstChild();
        NodeList projectNodeList = projectNode.getChildNodes
            ();
        for (int i = 0; i < projectNodeList.getLength(); i++)
        {
            Node childNode = projectNodeList.item(i);
35     if (childNode.getNodeName().equals("version"))
            {
                version = childNode.getTextContent();
                break;
            }
40     }

        return version;
    }

45 // constants
    private static final String POM_PATH = "pom.xml";
}

```

Codice A.11: La classe `VersionChooser` estrae la versione del progetto di DMASON dal file `pom.xml`

A.2.4 Classe di servizio `EC2Service`

Primo avvio

```

public static void boot()
{
    if (!EC2Service.booted)
145  {
        // load properties
        EC2Service.loadProperties();
        LOGGER.info("Properties loaded from " +
            PROPERTIES_FILE_PATH + ".");

150     // create an EC2 client
        EC2Service.buildEC2Client(EC2Service.region);
        LOGGER.info("A new EC2 client has been created!");
    }
}

```

```

    // create security group and keypair
155  try
    {
        EC2Service.createKeyPair();
    }
    catch (IOException e)
160  {
        LOGGER.severe(e.getClass().getSimpleName() + ": " +
            e.getMessage() + ".");
    }
    EC2Service.createSecurityGroup();

165  // initialize instance local state map
    EC2Service.initializeLocalInstances();
    LOGGER.info("Local instances states map populated!");

    EC2Service.booted = true;
170  LOGGER.info("Amazon EC2 Service boot process
        completed!");
    }
    else
    {
        LOGGER.warning("Amazon EC2 Service is already booted
            !");
175  }
    }
}

```

Codice A.12: Metodo di primo avvio della classe EC2Service

Creazione istanza EC2 on demand

```

205 public static RunInstancesResult createInstance(String
    groupName, String clusterType, int maxNumberInstances
    )
    throws FileNotFoundException, IOException
    {
        RunInstancesRequest instanceRequest = new
            RunInstancesRequest();

210  // define the requested virtual machine
        String instanceType = EC2Service.type;
        instanceRequest.withImageId(EC2Service.ami)
            .withInstanceType(instanceType)
            .withMinCount(1) // the minimum number of instances
                to launch
    }

```

```
215     .withMaxCount(maxNumberInstances) // the maximum
        number of instances to launch
        .withKeyName(EC2Service.KEY_NAME)
        .withSecurityGroups(groupName);

    // request the virtual machine
220 RunInstancesResult result = ec2.runInstances(
    instanceRequest);

    // add all created local instances to local map
    Iterator<Instance> instanceIterator = result.
        getReservation().getInstances().iterator();
    while (instanceIterator.hasNext())
225 {
        // add created local instance state to local map
        Instance instance = instanceIterator.next();
        String instanceID = instance.getInstanceId();
        String instanceDns = instance.getPublicDnsName();
230 EC2Service.localInstances.put(instanceID, new
            LocalInstanceState(instanceID, instanceDns,
                instanceType)); // here it creates a new local
            instance state
        LOGGER.info(clusterType + " node reservation: " +
            instanceID);

        // tag created instance
        int numInstances = EC2Service.localInstances.values()
            .size(); // size already includes created
            instance
235 EC2Service.tagInstance(
            instanceID,
            "Name",
            numInstances + "-" + EC2Service.groupName + "-" +
            clusterType
        );
240 }

    // update local instances map to file
    LocalInstanceStateManager.saveLocalInstanceStates(
        localInstances);

245 return result;
}
```

Codice A.13: Metodo per la creazione di un'istanza EC2 on demand

Avvio istanza EC2 on demand

```
public static StartInstancesResult startInstance(String
    instanceId)
{
800 // logging level
    // LOGGER.setLevel(Level.SEVERE);

    LOGGER.info("Request for instance " + instanceId + " to
        start.");
    Instance instance = retrieveInstance(instanceId);
805
    // check if instance is already running
    if (instance.getState().getName().equals(
        InstanceStateName.Running.toString()))
    {
        LOGGER.warning("Instance " + instanceId + " is
            already running!");
810 return null;
    }

    LOGGER.info("Instance " + instanceId + " is not already
        running!");
    StartInstancesRequest startRequest = new
        StartInstancesRequest();
815

    // define the start request
    startRequest.withInstanceIds(instanceId);
    StartInstancesResult result = ec2.startInstances(
        startRequest);

820 // be sure the instance is running
    boolean started = false;
    do
    {
        instance = retrieveInstance(instanceId);
825 if (instance.getState().getName().equals(
            InstanceStateName.Running.toString()))
        {
            started = true;
        }

830 try
        {
            // wait 1 second before checking again
            Thread.sleep(1000);
        }
835 catch (InterruptedException ie)
        {
            LOGGER.warning("Thread not ready to sleep!");
        }
    }
```

```
    }  
840  while (!started);  
    LOGGER.info("Instance " + instanceId + " has been  
        started!");  
  
    return result;  
} // end startInstance(String) method
```

Codice A.14: Metodo per l'avvio di un'istanza EC2 on demand

Bibliografia

- [1] John von Neumann. “Various Techniques Used in Connection with Random Digits”. In: National Bureau of Standards Applied Math Series.12 (1951), pp. 36–38 (pag. 1).
- [2] Tucker Balch. “TeamBots simulation and real robot execution environment”. In: (1997). URL: <https://www.cs.cmu.edu/~trb/TeamBots/> (pag. 2).
- [3] Nelson Minar et al. “The swarm simulation system.” In: (1996). URL: <http://www.swarm.org> (pag. 2).
- [4] Sean Luke et al. “MASON: A Java Multi-Agent Simulation Environment”. In: *Proceedings of the Agent 2003 Conference* (2003). URL: <https://cs.gmu.edu/~eclab/projects/mason/> (pag. 2, 4).
- [5] Gennaro Cordasco et al. “A Framework for Distributing Agent-Based Simulations”. In: *Euro-Par Workshops (1)’11*. 2011, pp. 460–470 (pag. 4).
- [6] AT&T. *What Is The Cloud?* 1993. URL: https://youtu.be/_a7hK6kWttE (pag. 6).
- [7] Jeff Bar. *Amazon Simple Queue Service Released*. 2004. URL: http://aws.typepad.com/aws/2004/11/amazon_simple_q.html (pag. 7).
- [8] *Announcing Amazon Elastic Compute Cloud (Amazon EC2) - beta*. 2006. URL: <https://aws.amazon.com/about-aws/whats-new/2006/08/24/announcing-amazon-elastic-compute-cloud-amazon-ec2---beta/> (pag. 7).
- [9] Paul McDonald. *Introducing Google App Engine + our new blog*. 2008. URL: <http://googleappengine.blogspot.nl/2008/04/introducing-google-app-engine-our-new.html> (pag. 7).
- [10] *Windows Azure General Availability*. 2010. URL: http://blogs.technet.com/b/microsoft_blog/archive/2010/02/01/windows-azure-general-availability.aspx (pag. 7).
- [11] *Launch of IBM Smarter Computing*. 2011. URL: https://web.archive.org/web/20130420162543/https://www-304.ibm.com/connections/blogs/IBMSmarterSystems/date/201102?lang=en_us (pag. 7).

- [12] Kara Swisher. D: All Things Digital (with Larry Ellison as guest). 2012. URL: <http://allthingsd.com/20120530/oracle-ceo-larry-ellison-live-at-d10/?refcat=d10> (pag. 7).
- [13] Ari Balogh. *Google Compute Engine is now Generally Available with expanded OS support, transparent maintenance, and lower prices*. 2012. URL: <https://developers.googleblog.com/2013/12/google-compute-engine-is-now-generally.html> (pag. 7).
- [14] B. Rochwerger et al. “The Reservoir model and architecture for open federated cloud computing”. In: *IBM Journal of Research and Development* 53.4 (lug. 2009), 4:1–4:11. DOI: 10.1147/jrd.2009.5429058. URL: <https://opennebula.org/> (pag. 7).
- [15] Jim Curry. *Introducing OpenStack*. 2010. URL: <https://www.openstack.org/blog/2010/07/introducing-openstack/> (pag. 7).
- [16] National Science Foundation. *Diagram of CSNET*. 1981 (pag. 8).
- [17] P M Mell e T Grance. *The NIST definition of cloud computing*. Rapp. tecn. 2011. DOI: 10.6028/nist.sp.800-145 (pag. 8).
- [18] Bobbie Johnson. *Cloud computing is a trap, warns GNU founder Richard Stallman*. Article on theguardian.com. 2008. URL: <https://www.theguardian.com/technology/2008/sep/29/cloud.computing.richard.stallman> (pag. 10).
- [19] Lydia Leong et al. *Quadrante magico per un’infrastruttura cloud come servizio, Versione internazionale*. 2016. URL: <https://www.gartner.com/technology/media-products/reprints/amazon/1-3IFUTGE-ITA.html> (pag. 10).
- [20] Bob Evans. *Why Microsoft Is Ruling The Cloud, IBM Is Matching Amazon, And Google Is 15BillionBehind*. Feb. 2018. URL: <https://www.forbes.com/sites/bobevans1/2018/02/05/why-microsoft-is-ruling-the-cloud-ibm-is-matching-amazon-and-google-is-15-billion-behind/#66f999f81dc1> (pag. 10).
- [21] *Amazon Web Services Launches*. 2006. URL: <http://phx.corporate-ir.net/phoenix.zhtml?c=176060%5C&p=irol-newsArticle%5C&ID=830816> (pag. 10).
- [22] URL: <https://aws.amazon.com/ec2/spot/> (pag. 12).
- [23] *SDK AWS for Java*. URL: <https://aws.amazon.com/it/sdk-for-java/> (pag. 12).
- [24] *AWS account*. URL: https://portal.aws.amazon.com/billing/signup?redirect_url=https%5C%3A%5C%2F%5C%2Faws.amazon.com%5C%2Fregistration-confirmation%5C&language=it_it#/start (pag. 12).

-
- [25] ISISlab. URL: <https://github.com/isislab-unisa/dmason/wiki> (pag. 13).
- [26] Apache Foundation. *Maven*. URL: <https://maven.apache.org/> (pagg 15, 30).
- [27] *Material Design*. URL: <https://material.io/> (pag. 20).
- [28] *Polymer Project*. URL: <https://www.polymer-project.org> (pag. 21).
- [29] W3C. Latest HTML specification. URL: <https://www.w3.org/standards/techs/html> (pag. 21).
- [30] W3C. URL: <https://www.w3.org/TR/2017/REC-html52-20171214/dom.html#elements-antics> (pag. 21).
- [31] URL: <https://www.webcomponents.org> (pag. 21).
- [32] W3C. 2016. URL: <https://www.w3.org/TR/web-animations-1/> (pag. 21).
- [33] URL: <https://bower.io> (pag. 21).
- [34] URL: <https://jquery.com/> (pag. 21).
- [35] URL: <https://masonry.desandro.com/> (pag. 22).
- [36] URL: <https://www.jsviews.com> (pag. 22).
- [37] URL: <https://docs.oracle.com/javaee/5/tutorial/doc/bnagx.html> (pag. 22).
- [38] URL: <https://docs.oracle.com/javaee/5/tutorial/doc/bnakc.html> (pag. 22).
- [39] URL: <https://www.npmjs.com/> (pag. 24).
- [40] URL: <https://nodejs.org/en/> (pag. 24).
- [41] URL: <https://docs.oracle.com/javaee/7/tutorial/servlets.htm#BNAFD> (pag. 30).
- [42] URL: <https://commons.apache.org/proper/commons-configuration/> (pag. 30).
- [43] URL: <http://www.jcraft.com/jsch/> (pag. 30).
- [44] URL: <https://aws.amazon.com/sdk-for-java/> (pag. 38).
- [45] URL: <https://aws.amazon.com/eclipse/> (pag. 38).
- [46] URL: <https://aws.amazon.com/ec2/details/> (pag. 39).