

**ALMA MATER STUDIORUM - UNIVERSITÀ DI BOLOGNA**

---

**SCUOLA DI INGEGNERIA E ARCHITETTURA**

*Dipartimento di Informatica*

*Corso di Laurea in Ingegneria Informatica*

**TESI DI LAUREA**

in

**Tecnologie Web**

**Monitoraggio e confronto delle performance di rete dei maggiori Cloud Providers**

**CANDIDATO:**

Daniele Piscaglia

**RELATORE:**

Chiar.mo Prof. Paolo Bellavista

**CORRELATORE:**

Chiar.mo Prof. Luca Foschini

---

Anno Accademico 2017-2018

Sessione II

# Indice

Capitolo 1 Introduzione.....	5
Capitolo 2 Il Cloud.....	6
2.1    Definizione di Cloud .....	6
2.2    Come funziona il cloud .....	7
2.3    Tipologie di servizi.....	9
2.3.1    SaaS.....	9
2.3.2    PaaS.....	10
2.3.3    IaaS.....	10
2.4    Vantaggi .....	10
2.5    Problemi .....	12
Capitolo 3 Monitoraggio delle performance .....	14
3.1    Sistemi già esistenti.....	14
3.1.1    awsspeedtest.xvf.dk.....	14
3.1.2    azurespeed.com .....	15
3.1.3    Problematiche delle soluzioni già esistenti .....	16
3.2    Lo scopo del progetto .....	17
Capitolo 4 Strumenti utilizzati .....	18
4.1    MERN .....	18
4.2    MongoDB.....	18
4.2.1    Aggregazione .....	19
4.2.2    Scalabilità orizzontale e replicazione .....	20
4.2.3    Vantaggi e svantaggi .....	21
4.3    Node.js .....	22
4.3.1    Moduli e NPM.....	22
4.4    Express .....	23
4.5    React.....	23
4.6    Hping.....	24

4.7	Iperf3 .....	25
Capitolo 5 Presentazione progetto.....		26
5.1	Modeling4Cloud.....	26
5.2	Parti e responsabilità .....	27
5.3	Raccolta dati.....	28
5.3.1	Macchine Virtuali.....	28
5.3.2	Gestione dei test .....	28
5.3.3	Risultati dei test.....	31
5.4	Salvataggio dei dati .....	32
5.4.1	Cluster .....	33
5.4.2	Ottimizzazione aggregazione e indicizzazione .....	33
5.4.3	Precomputing .....	34
5.5	Consultazione dei dati .....	36
5.6	Budget e costi .....	37
5.6.1	Tipologia, costo e numero istanze.....	37
5.6.2	Utilizzo di rete.....	39
5.6.3	La scalabilità su Amazon Web Services .....	41
Capitolo 6 Implementazione soluzione .....		42
6.1	Script per la raccolta dati.....	42
6.1.1	File di configurazione.....	43
6.1.2	Installazione dei test .....	44
6.1.3	Esecuzione dei test .....	48
6.1.4	Invio risultati .....	49
6.2	Persistenza su MongoDB .....	50
6.2.1	Analisi delle performance .....	51
6.3	Backend.....	56
6.3.1	Consultazione dati .....	56
6.3.2	Caricamento risultati e precomputing .....	58
6.3.3	Analisi delle prestazioni .....	61
6.4	Interfaccia Web .....	64
Capitolo 7 Analisi dei risultati raccolti .....		66

7.1	Confronto tra provider.....	66
7.2	Monitoraggio completo su Amazon Web Services .....	70
Capitolo 8 Conclusioni.....		73
Riferimenti .....		75

# Capitolo 1 Introduzione

Lo sviluppo e la crescente richiesta di prodotti software altamente scalabili con bacini di utenza sempre più sparsi su tutto il pianeta hanno favorito una esponenziale crescita di utilizzo del cloud computing. Questo scenario ha portato alla nascita di numerosi cloud provider in un mercato di continua competizione, che possiedono server dislocati sulle varie parti del pianeta.

Al giorno d'oggi l'infrastruttura cloud può fornire svariati livelli di personalizzazione e prestazioni dei servizi offerti all'utente, che connessi all'ampia lista di fornitori sul mercato può rendere la scelta della soluzione più adatta alle proprie necessità molto più complicata di quanto ci si aspetti, allontanando il cloud dallo scopo per il quale è nato, cioè facilitare lo sviluppatore nel rilascio di un nuovo prodotto software. Infatti, a causa dell'elevato grado di astrazione inserito dal modello del Cloud, per l'utilizzatore risulta impossibile stabilire precisamente a priori le prestazioni che un determinato sistema, distribuito sul cloud potrà offrire.

Per fornire una risposta a queste domande il presente progetto nasce con l'obiettivo di fornire un sistema capace di fornire un metodo di confronto e analisi delle prestazioni di rete dei principali cloud provider sul mercato:

- Amazon Web Services
- Google Cloud Platform
- Microsoft Azure
- IBM Cloud

Lo scopo sarà creare un modello dettagliato delle performance di rete dei provider, tramite la misurazione diretta dei parametri più significativi, al fine di permettere un confronto diretto, tramite una metrica comune, tra i differenti fornitori cloud e le aree geografiche da loro servite.

# Capitolo 2 Il Cloud

## 2.1 Definizione di Cloud

Per cloud si intende un nuovo metodo di gestione delle risorse che permette all'utente di delegare la gestione di una determinata infrastruttura informatica o servizio software a un ente terzo completamente scollegato chiamato cloud provider. Raggruppando tutte le risorse, con le relative necessità di installazione, configurazione e manutenzione, in un unico grande compito gestito dal provider.

Le caratteristiche chiave che distinguono un servizio cloud sono:

- L'utilizzo delle risorse offerte deve essere condiviso tra i vari utilizzatori, in maniera che la stessa risorsa venga assegnata a vari clienti in momenti diversi favorendo il riutilizzo
- Flessibilità di utilizzo garantendo sempre una immediata (in alcuni casi perfino automatica) scalabilità, che sia verso l'alto o verso il basso
- La completa automazione nel fornire il servizio all'utente che deve potervi accedere senza che sia necessario alcun intervento umano da parte del fornitore
- I servizi offerti devono essere accessibili ovunque, contemporaneamente da più luoghi e in qualsiasi momento
- I servizi devono essere poter utilizzati indipendentemente dal tipo (e soprattutto dalla potenza computazionale) dell'utilizzatore. Si pensi ad un utilizzo da smartphone piuttosto che PC o server.
- Basando l'intero sistema su un principio di costo on-demand<sup>1</sup> deve anche essere fornito un dettagliato meccanismo per l'analisi e il monitoraggio delle risorse utilizzate nel tempo

---

<sup>1</sup> Il pagamento on-demand prevede che un utente paghi in base all'effettivo utilizzo di una risorsa

## 2.2 Come funziona il cloud

Principalmente si possono distinguere due diversi modelli di cloud:

- **Pubblico:** fornito dai provider sparsi in tutto il mondo, utilizzato sia da aziende che privati, a seconda dei casi utilizzabile gratuitamente o a pagamento in base all'utilizzo
- **Privato:** principalmente utilizzato da imprese che lo gestiscono internamente nella sede o all'esterno, permette di fornire servizi più specifici e con un maggiore controllo, soprattutto mantenendo i dati gestiti all'interno della propria rete. Nel caso il cloud sia condiviso tra più aziende che condividono lo stesso obiettivo si parla di Community Cloud.

In realtà, per avvalersi dei vantaggi di entrambe le soluzioni spesso viene utilizzata una soluzione ibrida, ad esempio per mantenere i dati più riservati sul cloud privato e comunque avvalersi di tutti i vantaggi del cloud pubblico.

Particolarmente interessante è l'infrastruttura del cloud pubblico, nella quale i cloud provider, per garantire delle performance il più adatte e equilibrate a tutte le zone del pianeta mettono a disposizione uno svariato numero di datacenter nel quale raggruppano tutte le risorse utilizzabili da quella determinata zona, si pensi che AWS attualmente conta più di 55 datacenter in continua espansione [1]. Ad ogni modo grazie ad un elevato livello di astrazione i servizi vengono forniti all'utilizzatore senza che lui si renda conto di dove le effettive risorse fisiche risiedano (caratteristica principale che lo differenzia il cloud computing dal grid computing, dove il calcolo è sempre distribuito ma in maniera direttamente correlata al sistema stesso).



Figura 1 Mappa dei datacenter di Azure, Amazon e Google [2]

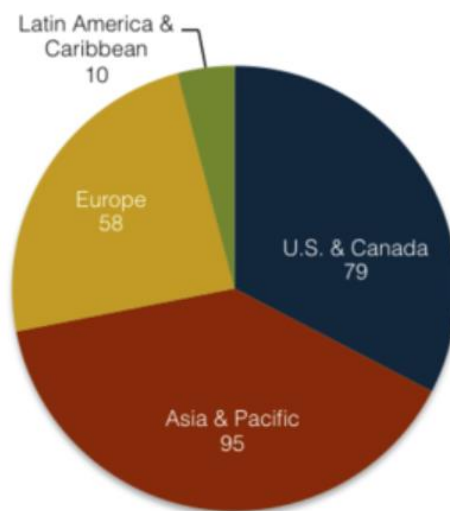


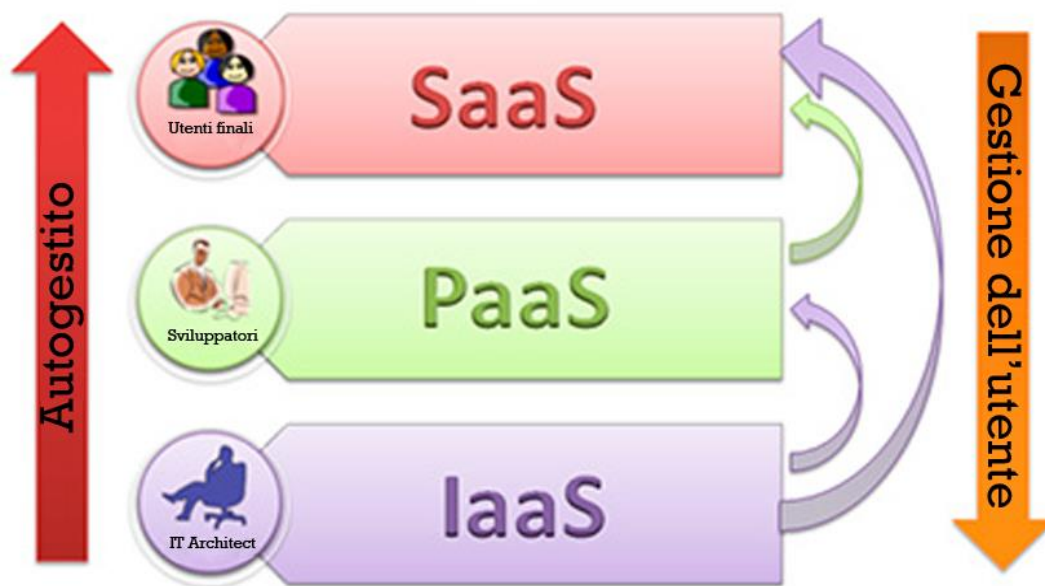
Figura 2 Distribuzione mondiale dei datacenter dei maggiori provider (Alibaba, AWS, GCP, Azure, IBM, Oracle)



## 2.3 Tipologie di servizi

Le risorse offerte dal cloud possono essere di diversi tipi, di basso livello come potenza di calcolo o spazio di archiviazione, oppure delle macchine già configurate con il sistema operativo preferito funzionanti pronte all'uso da remoto; oppure di più alto livello, come delle vere e proprie applicazioni.

I servizi erogati si distinguono in 3 macro categorie: SaaS, PaaS e IaaS



*Figura 3 Categorie dei servizi offerti dal cloud [3]*

### 2.3.1 SaaS

Software as a Service è il modello che prevede la fornitura di un prodotto software che anziché essere eseguito sulla macchina cliente come si è abituati, viene eseguito su un datacenter del cloud provider continuando ad offrire una normalissima interfaccia per l'utente. Esempi di applicazioni di questa categoria: Office 365, Gmail e Google Drive

### 2.3.2 PaaS

Platform as a Service, in questo caso viene fornita una piattaforma che è già pronta per essere utilizzata dagli sviluppatori per sviluppare e distribuire un'applicazione. Fornendo i servizi necessari per eseguire l'applicazione e salvare i dati. Un esempio è la piattaforma Azure di Microsoft utilizzabile per distribuire applicazioni del framework .Net.

### 2.3.3 IaaS

Infrastructure as a Service, è il modello di più basso livello, che fornisce l'intera infrastruttura IT partendo dal basso. Permette di acquistare potenza di calcolo, storage, servizi di rete per poi poterli personalizzare ed utilizzare per distribuire i propri servizi. Solitamente questi servizi vengono erogati come una macchina virtuale utilizzabile in remoto, sulla quale può essere installato il sistema operativo che si desidera.

## 2.4 Vantaggi

Il cloud permette all'utilizzatore di scaricarsi di tutti i costi di installazione, configurazione e manutenzione delle infrastrutture collegate, garantendo una flessibilità e semplicità nella espansione (o riduzione) delle risorse infinitamente più conveniente rispetto alla soluzione autonoma, che richiederebbe un investimento iniziale direttamente proporzionale alle esigenze e legherebbe l'utente ad una strumentazione che potrebbe velocemente rivelarsi obsoleta. Oltre alle esigenze economiche va sottolineata anche l'esigenza di personale adeguato alla gestione che queste apparecchiature richiederebbero.

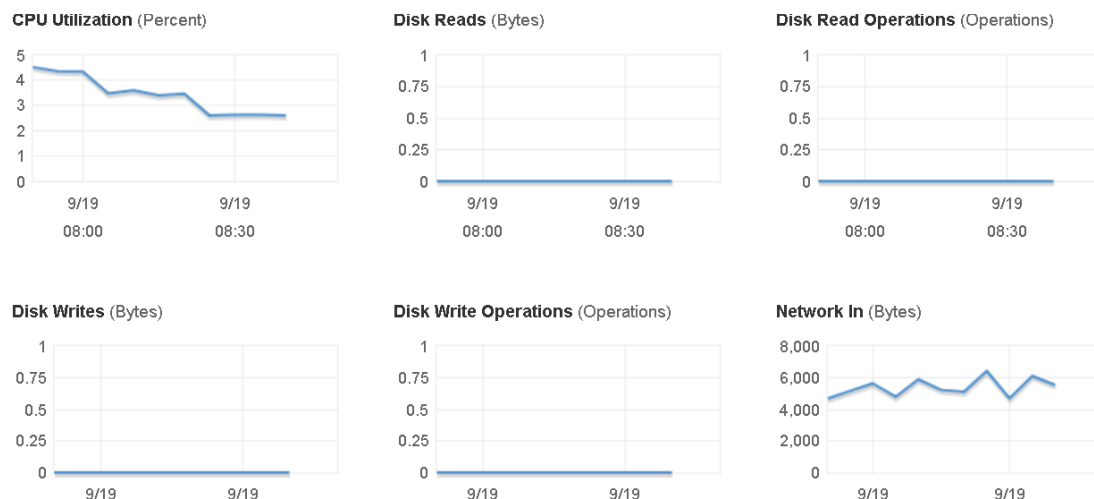
Questo modello di erogazione dei servizi è evidentemente molto conveniente soprattutto per quel tipo di utenti che hanno necessità di una quantità di risorse molto variabile del tempo o addirittura che non può essere stabilita a priori in modo chiaro. Si pensi ad esempio ad un sito di vendite online che ha pianificato un periodo di saldi, durante il quale sarà auspicabile ricevere una mole di utenti superiore a quella media, nel caso il sito fosse gestito con delle apparecchiature proprie bisognerebbe aumentare la potenza disponibile con un conseguente spreco in eccesso durante i periodi normali, mentre nel

caso si faccia uso di una piattaforma cloud non sarebbe necessario alcun intervento in quanto quest'ultima potrebbe automaticamente bilanciare le risorse in base al carico richiesto.

Per tutte le tipologie di servizi che devono essere accessibili da vaste aree geografiche (o globalmente) l'utilizzo di un cloud provider che possiede vari datacenter dislocati nelle varie aree di interesse renderebbe estremamente semplice assicurare che il servizio richiesto sia accessibile ovunque e con prestazioni il più possibile equilibrate indipendentemente dalla regione dell'utente.

Grazie all'elevata disponibilità di risorse che un cloud provider può gestire, grazie una capacità di ridondanza, ribilanciamento e ridistribuzione viene anche garantito un elevatissimo standard di affidabilità.

Per ogni risorsa utilizzata il pagamento è previsto in base all'utilizzo in una prestabilita finestra di tempo (che può variare da oraria a mensile), questo permette all'utente di pianificare i costi anche grazie a tutti gli strumenti di monitoraggio forniti per controllare le risorse che vengono utilizzate e il relativo prezzo.



*Figura 4 Monitoraggio utilizzo risorse di una Macchina Virtuale su AWS*

## 2.5 Problemi

Mentre è evidente di quali vantaggi si può godere grazie al cloud, quest'ultimo in alcuni casi può anche portare a delle problematiche nascoste non trascurabili.

Una volta che un determinato servizio viene gestito tramite il cloud, anche i dati ad esso collegati vengono inevitabilmente gestiti ed inviati a datacenter dei quali non conosciamo la posizione fisica che spesso potrebbe anche essere in altre nazioni, aspetto che potrebbe portare a problematiche riguardanti la privacy e di tipo legislative.

Mentre l'adozione di alcune infrastrutture informatiche per una piccola azienda porterebbe a dei costi relativamente grossi, forse per aziende più grandi può risultare economicamente conveniente adottare un modello più tradizionale gestendo autonomamente i servizi necessari quando possibile. Inoltre nonostante il cloud permetta di assolvere di tutti i compiti di gestione, spesso è comunque richiesto del personale specializzato addetto alla gestione dei servizi tramite cloud.

I servizi offerti dal cloud hanno evidentemente meno esigenze di manutenzione e gestione, questo aspetto però ovviamente porta anche a una ridotta personalizzazione possibile per esigenze particolari rispetto a quello che si riuscirebbe a fare con qualcosa realizzato internamente. Inoltre nel caso sia necessaria una migrazione dei propri servizi da un provider ad un altro il trasferimento potrebbe rivelarsi abbastanza complicato.

Ogni volta che si utilizza il cloud per svolgere una determinata operazione, tutti i dati necessari per quella operazione devono essere trasferiti dall'utilizzatore al cloud provider tramite la rete, in base alla tipologia e soprattutto alla loro quantità diventa evidente quanto l'adozione del modello cloud possa diventare controproducente in quanto vengono introdotte tre tipologie di problematiche:

- **Disponibilità:** è sempre richiesta la connettività alla rete, quindi mentre si può considerare il cloud provider affidabile a tal punto da fornire una disponibilità sempre assicurata, lo stesso potrebbe non

valore per l'utilizzatore, per il quale il servizio diventerebbe completamente irraggiungibile se scollegato dalla rete.

- Latenza: ogni volta che si vuole effettuare un'operazione sul cloud, nonostante le elevatissime prestazioni che quest'ultimo offre, bisogna innanzitutto inviare la relativa richiesta e attendere che il risultato ci venga rispedito indietro. Questo trasferimento sulla rete potrebbe avvenire tra parti più o meno lontane geograficamente e quindi aggiungere una latenza non trascurabile, soprattutto nel caso di sistemi real-time.
- Banda: la rete inoltre è un sistema di trasmissione relativamente lento rispetto a una qualsiasi comunicazione locale, quindi per un utilizzo che richiede il trasferimento di grosse quantità di dati la velocità di banda va necessariamente tenuta in considerazione.

Grazie al costante miglioramento delle prestazioni delle reti globali l'impatto di queste problematiche si è ampiamente ridotto negli anni, ad ogni modo nelle parti successive si prenderà in considerazione questo problema, cercando di creare un sistema per il monitoraggio di queste prestazioni e il confronto tra i vari provider pubblici utilizzando una metrica comune.

# Capitolo 3 Monitoraggio delle performance

## 3.1 Sistemi già esistenti

Date le dimensioni del mercato mondiale del cloud e la quantità di enti che sono in competizione in questo settore è facile reperire una grande quantità di analisi effettuate per confrontare i servizi offerti dai vari provider. Buona parte di queste analisi ad ogni modo si sofferma particolarmente sulle prestazioni dell'infrastruttura offerta (potenza computazionale, velocità di accesso al disco, disponibilità ecc....) piuttosto che sulle performance di rete. Le soluzioni più significative che si occupano di fornire un sistema di controllo della rete sono quelle elencate di seguito.

### 3.1.1 awsspeedtest.xvf.dk

Questo sito permette di vedere facilmente la latenza tra la propria macchina e i data center del provider Amazon:





















Region	Average Latency		History
 EU (Frankfurt) <i>eu-central-1</i>	27ms	-	
 EU (Paris) <i>eu-west-3</i>	42ms	-	
 EU (London) <i>eu-west-2</i>	44ms	-	
 EU (Ireland) <i>eu-west-1</i>	53ms	-	
 US East (N. Virginia) <i>us-east-1</i>	136ms		
 Canada (Central) <i>ca-central-1</i>	140ms		
 US East (Ohio) <i>us-east-2</i>	150ms		
 Asia Pacific (Mumbai) <i>ap-south-1</i>	157ms		

Figura 5 Interfaccia del sito [awsspeedtest.xvf.dk](http://awsspeedtest.xvf.dk) [4]

Il sito si limita a mostrare solamente i dati relativi al provider Amazon ma sono facilmente reperibili siti analoghi che svolgono la stessa funzione per altri provider, ad ogni modo ognuno di questi siti condivide le seguenti problematiche:

- Non è disponibile un test di banda
- I test vengono effettuati verso la macchina cliente, non è possibile effettuare test tra i data center delle varie zone
- I test non sono programmabili

### 3.1.2 azurespeed.com

Il seguente sito risulta senza dubbio quello più avanzato attualmente disponibile in rete fornendo un ottimo grado di personalizzazione per quanto riguarda i datacenter interessati e soprattutto la tipologia di test mettendo a disposizione un test per la latenza:

Latency Test			
Geography	Region	Location	Average Latency (ms)
Europe	West Europe	Netherlands	39 ms
America	West US	California	183 ms
Asia	East Asia	Hong Kong	212 ms

*Figura 6 Esempio test latenza sul sito azurespeed.com [5]*

Ed un test di upload/download della banda:

Upload Speed Test				
<div>Start Upload Speed Test &gt;&gt;</div>				
Geography	Region	Location	Upload Progress	Upload Speed
Europe	West Europe	Netherlands	100%	591.22 KB/s
America	West US	California	100%	214.95 KB/s
Asia	East Asia	Hong Kong	100%	190.33 KB/s

*Figura 7 Esempio test di banda sul sito azurespeed.com [5]*

Ad ogni modo anche in questo caso ci sono notevoli limitazioni:

- Test limitati al solo provider Microsoft Azure
- I test vengono effettuati verso la macchina cliente, non è possibile effettuare test tra le varie zone
- I test sono “one-shot” e non programmabili

### 3.1.3 Problematiche delle soluzioni già esistenti

Da una veloce analisi delle soluzioni che sono attualmente disponibili risultano evidenti le problematiche che questi hanno in comune:

- Mancanza di un sistema universalmente compatibile, che permetta di confrontare un qualsiasi provider con un altro
- Mancanza di test sia di latenza che di banda
- Il test è effettuabile solo verso il cliente, invece è necessario poter monitorare le performance anche tra datacenter in zone diverse (informazione di interesse per tutte quelle applicazioni che necessiterebbero di lavorare contemporaneamente su più datacenter diversi) oppure anche tra servizi erogati dallo stesso datacenter (due macchine sullo stesso datacenter che devono lavorare in parallelo)
- Possibilità di raccogliere costantemente i dati, in modo da monitorare anche la variazione delle performance nel tempo
- Possibilità di personalizzare e programmare i test e la loro frequenza



## 3.2 Lo scopo del progetto

Il presente progetto parte come proseguimento del lavoro svolto dal collega Alberto Bagnacani, che nel suo progetto di laurea “Modeling4Cloud” ha sviluppato un prototipo del sistema di monitoraggio della latenza per i cloud provider Amazon Web Services e Google Cloud Platform.

L’obiettivo è quello di espandere il software:

- Integrando dei test per la velocità di banda
- Rendere i test personalizzabili e completamente programmabili
- Modificare il sistema rendendolo compatibile con i cloud provider IBM e Microsoft Azure e potenzialmente portabile verso un qualsiasi provider a piacere
- Gestire la scalabilità del sistema in modo da poter garantire il funzionamento su più provider e più data center contemporaneamente
- Migliorare i tempi di risposta del sistema durante l’interrogazione dei dati, in modo da ottenere in un tempo accettabile dei riassunti dettagliati riguardanti un ampio numero di dati

# Capitolo 4 Strumenti utilizzati

## 4.1 MERN

Mern rappresenta l'acronimo di 4 differenti tecnologie, nate da scopi, fonti ed in tempi completamente indipendenti fra loro, che però se accoppiate forniscono un vero e proprio framework full-stack per gestire un'applicazione web in tutte le sue parti (persistenza, backend e frontend). Le 4 tecnologie in questione sono **MongoDB** (persistenza), **Express** (backend), **React** (frontend), **Node** (piattaforma di esecuzione).

Tutte le componenti di questo sistema condividono il linguaggio di programmazione che le compone: **Javascript**. Aspetto che ha facilitato una grande diffusione tra la comunità di sviluppatori web, in quanto permetteva lo sviluppo completo di un'applicazione web, in tutte le sue parti richiedendo di conoscere un solo linguaggio di programmazione, inoltre trattandosi di Javascript, ampiamente conosciuto nella comunità degli sviluppatori web da anni, l'introduzione di queste nuove tecnologie è stata molto apprezzata.

## 4.2 MongoDB

MongoDB è un DBMS non relazionale, classificato come un database di tipo NoSQL<sup>1</sup>, cioè non relazionale, che abbandona il metodo tradizionale per il salvataggio di dati in tabelle relazionali utilizzando un approccio completamente nuovo, molto più flessibile basato sul formato JSON.

JSON (Javascript Object Notation) è un formato sempre più utilizzato per lo scambio di dati nelle applicazioni web fra client e server grazie alla sua semplicità e flessibilità. In MongoDB ogni dato che viene salvato viene convertito nel formato BSON (Binary JSON) variante in codice binario di JSON che permette ricerca nei valori più veloce e una gestione dello spazio su disco più efficace rispetto al formato originale.

---

<sup>1</sup> NoSQL è un movimento che promuove la persistenza dei dati nei sistemi software caratterizzata dal fatto di non utilizzare database a modello relazionale. L'espressione NoSQL fa riferimento al linguaggio SQL, il più comunemente utilizzato nei database relazionali.

In MongoDB ogni dato salvato viene chiamato documento, i vari documenti sono raggruppati in collezioni che compongono un database. La differenza fondamentale rispetto ai database relazionali è la completa flessibilità nello schema di ogni documento, infatti, mentre nei database relazionali tutte le tuple di una determinata tabella condividono lo stesso schema di campi, in MongoDB lo schema dei campi di ogni documento è salvato nel documento stesso. Questo facilita enormemente l'inserimento di nuove tipologie di dati rendendolo preferibile all'approccio tradizionale per molte nuove applicazioni.

#### 4.2.1 Aggregazione

Il fatto che ogni documento venga salvato in maniera autonoma senza riferimenti esterni ad altri oggetti rende le interrogazioni più complicate privandole di operatori come il JOIN. Inizialmente questo problema veniva risolto grazie all'utilizzo della famosa funzione NoSQL *MapReduce* [6], che permette allo sviluppatore di scrivere funzioni javascript per operare su più dati. Ad ogni modo questo processo può risultare complicato e poco ottimizzato, a questo scopo in MongoDB è stato introdotto un sofisticato meccanismo di aggregazione che mette a disposizione una lista di operazioni da organizzare in delle pipeline (sequenze) per risolvere query complicate. Nonostante la funzione *MapReduce* rimanga disponibile per coprire quei pochi casi che non sono implementabili dall'aggregazione, è consigliato utilizzare il nuovo meccanismo quando possibile in quanto molto più semplice e più ottimizzato.

Le operazioni più importanti sono:

- *Match*: filtra i documenti
- *Group*: permette di raggruppare più documenti in base a uno o più campi in comune, mettendo a disposizione degli operatori cumulativi per il calcolo sui campi raggruppati
- *Project*: permette di modificare i campi aggiungendone di nuovi o rimuovendo quelli superflui
- *Sort*: ordina i risultati in base ai valori di uno o più campi
- *Unwind*: permette di scomporre un documento che contiene un array in più documenti, ognuno con un solo valore dell'array

L'aggregazione è quindi uno strumento in grado di operare su un grande numero di dati, permettendo di svolgere anche operazioni cumulative sui loro campi, questo processo, portando ad accedere a molti dati contemporaneamente può richiedere molto tempo. Per questo motivo a partire dalla versione 3.2 di MongoDB alcuni processi di aggregazione (ad esempio Sort e Match) possono fare uso degli indici.

Gli indici sono delle strutture dati che salvano una piccola porzione dei dati di una collezione, vengono creati specificando i campi da contenere e vengono ordinati in base al valore dei campi in ordine crescente o decrescente. A questo punto possono essere utilizzati per svolgere in maniera più leggera ed efficiente delle operazioni di comparazione, ordinamento o selezione di un intervallo di valori sui documenti che rappresentano.

#### 4.2.2 Scalabilità orizzontale e replicazione

Per garantire un massimo grado di scalabilità MongoDB permette la creazione di un cluster di database con lo scopo di migliorare le prestazioni e al tempo stesso di aumentare la capacità di storage. Questo processo, chiamato Sharding, consiste nel distribuire il database su più nodi, dove ogni nodo contiene una porzione del database totale, creando uno Sharded Cluster.

L'architettura di un cluster prevede tre tipologie di componenti:

- **Config Server:** è quella parte che gestisce le informazioni riguardanti la struttura del cluster, come i dati sono distribuiti su di esso e come reperirli. All'interno di un cluster, soprattutto in un ambiente di produzione, può essere replicato (diventando un Config Server Replica Set) dato il compito vitale che svolge.
- **Shard Server:** è il componente dove vengono salvati i dati. Può essere un singolo database, oppure per garantire una maggiore affidabilità, può essere replicato (Shard Server Replica Set) evitando anche perdite di dati in caso di guasto.
- **Router mongos:** è una parte che non contiene dati ma rappresenta il punto di accesso per tutti i clienti che vogliono utilizzare il cluster, permettendogli di effettuare le normali operazioni come su un semplice database.

Per quanto riguarda la replicazione e la resistenza ai guasti quindi, MongoDB mette a disposizione la possibilità di replicare i suoi componenti in dei Replica Set. Ogni Replica Set può creare un qualsiasi numero di componenti paralleli, ad ogni modo solamente uno di questi sarà eletto come Primary (primario) e tutti gli altri Secondary (secondari) saranno aggiornati in base alle modifiche applicate al primario. Nel caso di qualsiasi tipo di guasto al primario un secondario verrà automaticamente eletto per prendere il suo posto.

Per ogni database che viene salvato sul cluster sono possibili due metodologie di sharding (strategie per l'organizzazione dei documenti sugli Shard) basate sul campo che viene scelto come chiave di Shard (un campo presente su tutti i documenti):

- Range-based: si effettua una suddivisione in base ai valori della chiave. Salvando elementi con valori del campo “vicini” sullo stesso nodo. Questo però potrebbe portare ad avere nodi più carichi di altri nel caso i valori siano distribuiti in modo non equo.
- Hash-based: viene effettuato un hash<sup>1</sup> della chiave di Shard e poi la suddivisione effettuata in base ad esso. In questo modo il carico diventa più omogeneo sugli Shard, però recuperare oggetti con valori simili potrebbe richiedere di accedere a vari nodi.

#### 4.2.3 Vantaggi e svantaggi

Come già evidenziato, MongoDB si adatta molto bene ad un utilizzo in applicazioni con una grande mole di dati eterogenei in cui le informazioni da salvare possono cambiare spesso nel tempo o non avere uno schema ben preciso. Allo stesso tempo però, questa flessibilità rende MongoDB non particolarmente adatto alla gestione di oggetti con numerose relazioni tra loro, dove può essere importante mantenere una integrità fra di essi, aspetto che sarebbe molto più facile gestire con un database relazionale.

Al contrario dei database tradizionali dove tutte le tuple condividono lo stesso schema, dove questo si può salvare una sola volta a livello di tabella, in MongoDB per ogni documento è necessario salvare, oltre al valore dei

---

<sup>1</sup> Nel campo informatico un hash è una funzione non invertibile che mappa una stringa in un'altra stringa di lunghezza minore. Questa funzione garantisce che non ci siano stringhe di hash più probabili di altre.

campi, anche come questi campi sono composti. Tutto questo ovviamente comporta un incremento nello spazio utilizzato a parità di informazioni salvate sul database.

MongoDB è stato progettato e sviluppato in maniera da garantire il più elevato grado di prestazioni in termini di velocità e latenza possibili per garantire una scalabilità in grado di gestire una grande mole di dati, come può testimoniare il fatto che sia utilizzato per la gestione di dati da aziende del calibro di Facebook ed Ebay. L'aggregazione è stata progressivamente ottimizzata per rendere prestazioni sempre migliori, ad ogni modo risulta difficilmente scalabile su una grande mole di dati.

## 4.3 Node.js

Node è una piattaforma realizzata sul motore Javascript V8 di Chrome (Google), che tramite l'utilizzo di un modello ad eventi e un sistema di I/O non bloccante permette di realizzare applicazioni web veloci ed efficienti; Infatti permette in maniera particolarmente semplice di creare server web http, particolarmente adatti per applicazioni real-time che elaborano dati e distribuibile su più sistemi.

Il modello di esecuzione asincrono di Node, non si basa su processi concorrenti ma è completamente comandato dagli eventi di I/O, infatti ogni volta che un determinato evento avviene, Node riceve una relativa notifica dal sistema operativo, che richiama una specificata funzione, detta di callback, che esegue le operazioni necessarie. Questa strategia, nonostante non sia molto indicata per funzioni con un utilizzo CPU intensivo a causa dell'esecuzione single-threaded di Node, si rivela molto conveniente quando c'è un elevato traffico di rete e numerose operazioni di I/O.

### 4.3.1 Moduli e NPM

Uno degli aspetti più interessanti di Node è proprio la disponibilità di un elevatissimo numero di moduli, cioè delle soluzioni già implementate per i problemi più comuni che uno sviluppatore può affrontare, utilizzabili per evitare di dover scrivere una determinata parte di codice se è già stato fatto da qualcun altro.

Il metodo più semplice e più diffuso per la gestione di questi package o moduli è proprio NPM (Node Package Manager), strumento completamente gratuito che tramite linea di comando permette di gestire, installare e distribuire i moduli con un accurato meccanismo di versioning e di dipendenze di progetto.

## 4.4 Express

Express.js è il framework gratuito ed open-source per la creazione della parte di back-end di applicazioni web più famoso e utilizzato di Node. Il successo di express è stato dettato dalla semplicità che introduce nel processo della gestione delle chiamate REST API<sup>1</sup> in Node e dal minimalismo che gli permette di svolgere le funzioni di base in maniera molto semplice, senza escludere però tramite l'utilizzo di “middleware”, cioè parti di software intermedie che svolgono determinate funzioni, la gestione di logiche complicate. Un middleware non è altro che una parte di codice “filtro” al quale vengono fornite le parti di Request e Response delle chiamate web ricevute e svolgono operazioni su di esse.

## 4.5 React

React è una libreria Javascript creata da Facebook poi diventata Open Source particolarmente orientata alla creazione di Single Page Application (applicazioni web con una sola pagina). Tramite l'utilizzo di un approccio dichiarativo molto innovativo e simile all'HTML, basato su componenti dinamici e riutilizzabili, componibili tra loro, si presenta come un framework molto intuitivo per la gestione della parte di presentazione di una applicazione web e la creazione di applicazioni native per dispositivi mobile come Android e iOS.

---

<sup>1</sup> REST è l'acronimo di REpresentational State Transfer, strategia che non fa altro che associare i quattro metodi dei form HTML con le quattro operazioni base degli oggetti: GET con Read (Select), POST con Create, PUT con Update e, ovviamente, Delete con Delete. Meccanismo utilizzato dalle applicazioni web per una più semplice gestione delle richieste proveniente dall'utente. L'utilizzo avviene tramite delle chiamate http agli indirizzi (endpoint API) definiti dall'applicazione.

## 4.6 Hping

Hping è uno degli strumenti più diffusi per il monitoraggio e test di firewall e reti. Oltre a fornire uno standard ping ICMP<sup>1</sup> permette un'analisi anche dei protocolli UDP e TCP e delle relative porte o funzione di traceroute per ricavare il percorso seguito dai pacchetti inviati sulla rete. Inoltre, tra i vari strumenti che permettono questo tipo di test (come qperf, psping, paping) risulta essere l'unico che mostra anche il parametro Time To Live del pacchetto.

Hping è completamente gratuito ed utilizzabile tramite linea di comando su sistemi Linux, MacOS o Windows, utilizzabile per effettuare quello che viene chiamato port scan TCP o UDP, cioè per controllare se una determinata porta è aperta (i pacchetti vengono accettati). Può essere anche utilizzato con lo scopo di “fare un ping” su una macchina che ad esempio potrebbe aver bloccato il traffico ICMP impedendo l'utilizzo del metodo più tradizionale, inviandogli un pacchetto di sincronizzazione (SYN) che automaticamente scatena una risposta con un pacchetto SYN/ACK dal ricevente, misurando il tempo di attesa tra l'invio e ricezione della risposta. Mentre il ping tradizionale si tratta di una coppia di messaggi ICMP “botta e risposta” nel caso del ping TCP viene introdotto un piccolo overhead applicativo, è quindi comprensibile che i risultati dei due test non siano equivalenti, come si può vedere nel test in figura sotto.

```
ubuntu@ip-172-31-27-127:~$ ping 35.180.31.118
PING 35.180.31.118 (35.180.31.118) 56(84) bytes of data:
64 bytes from 35.180.31.118: icmp_seq=1 ttl=48 time=7.67 ms
64 bytes from 35.180.31.118: icmp_seq=2 ttl=48 time=7.76 ms
64 bytes from 35.180.31.118: icmp_seq=3 ttl=48 time=7.77 ms
64 bytes from 35.180.31.118: icmp_seq=4 ttl=48 time=7.88 ms
64 bytes from 35.180.31.118: icmp_seq=5 ttl=48 time=7.78 ms
^C
--- 35.180.31.118 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4007ms
rtt min/avg/max/mdev = 7.673/7.776/7.882/0.066 ms
ubuntu@ip-172-31-27-127:~$ sudo hping3 -S -p 22 35.180.31.118
HPING 35.180.31.118 (eth0 35.180.31.118): S set, 40 headers + 0 data bytes
len=44 ip=35.180.31.118 ttl=48 DF id=0 sport=22 flags=SA seq=0 win=26883 rtt=9.0
ms
len=44 ip=35.180.31.118 ttl=46 DF id=0 sport=22 flags=SA seq=1 win=26883 rtt=8.9
ms
len=44 ip=35.180.31.118 ttl=46 DF id=0 sport=22 flags=SA seq=2 win=26883 rtt=8.7
ms
len=44 ip=35.180.31.118 ttl=48 DF id=0 sport=22 flags=SA seq=3 win=26883 rtt=8.7
ms
len=44 ip=35.180.31.118 ttl=47 DF id=0 sport=22 flags=SA seq=4 win=26883 rtt=8.5
ms
^C
--- 35.180.31.118 hping statistic ---
5 packets transmitted, 5 packets received, 0% packet loss
round-trip min/avg/max = 8.5/8.8/9.0 ms
```

*Figura 8 Differenza tra ping ICMP e TCP*

---

<sup>1</sup> ICMP (Internet Control Message Protocol) è il protocollo utilizzato nelle reti per il controllo, diagnostica e segnalazione di malfunzionamenti



## 4.7 Iperf3

Iperf3 è uno strumento a linea di comando interamente scritto in C, disponibile per piattaforme UNIX e Windows, nato per il monitoraggio del collegamento tra due macchine in rete e per aiutare nella relativa configurazione, utilizzando entrambi i protocolli TCP e UDP. A tale scopo prevede che agli estremi di tale collegamento siano eseguite due istanze dello strumento, un client ed un server, attraverso i quali è possibile generare un flusso dati TCP o UDP (da client a server o viceversa) e monitorare le performance di rete come larghezza di banda, perdita pacchetti e soprattutto come queste cambiano in base al variare della configurazione di determinati parametri. A tale scopo ogni test è configurabile con una serie di opzioni che permettono di scegliere quanto il test deve durare, permettendo anche di dividerlo in sotto intervalli e effettuare più connessioni in parallelo.

Client(Mittente)										Server(Ricevente)									
<pre>ubuntu@ip-172-31-5-30:~\$ sudo iperf3 -c 18.222.248.68 -p 6500 Connecting to host 18.222.248.68, port 6500 [ 4] local 172.31.5.30 port 37580 connected to 18.222.248.68 port 6500 [ ID] Interval      Transfer    Bandwidth  Retr  Cwnd [ 4] 0.00-1.04 sec  4.50 MBytes 36.3 Mbits/sec  53   188 KBytes [ 4] 1.04-2.00 sec  2.86 MBytes 25.0 Mbits/sec   3   174 KBytes [ 4] 2.00-3.05 sec  3.48 MBytes 27.9 Mbits/sec   0   191 KBytes [ 4] 3.05-4.03 sec  2.98 MBytes 25.6 Mbits/sec   0   197 KBytes [ 4] 4.03-5.00 sec  2.98 MBytes 25.7 Mbits/sec   0   202 KBytes [ 4] 5.00-6.00 sec  3.04 MBytes 25.4 Mbits/sec   0   214 KBytes [ 4] 6.00-7.08 sec  3.04 MBytes 23.8 Mbits/sec   1   178 KBytes [ 4] 7.08-8.01 sec  2.80 MBytes 25.0 Mbits/sec   0   202 KBytes [ 4] 8.01-9.00 sec  3.54 MBytes 30.0 Mbits/sec   0   214 KBytes [ 4] 9.00-10.02 sec  2.92 MBytes 24.2 Mbits/sec   5   164 KBytes ----- [ ID] Interval      Transfer    Bandwidth  Retr [ 4] 0.00-10.02 sec 32.2 MBytes 26.9 Mbits/sec  62 [ 4] 0.00-10.02 sec 30.9 MBytes 25.9 Mbits/sec iperf Done.</pre>										<pre>ubuntu@ip-172-31-42-206:~\$ sudo iperf3 -s -p 6500 Server listening on 6500 ----- Accepted connection from 107.21.185.163, port 37578 [ 5] local 172.31.42.206 port 6500 connected to 107.21.185.163 port 37580 [ ID] Interval      Transfer    Bandwidth  Retr [ 5] 0.00-1.00 sec  2.81 MBytes 23.6 Mbits/sec [ 5] 1.00-2.01 sec  2.99 MBytes 24.9 Mbits/sec [ 5] 2.01-3.03 sec  3.14 MBytes 25.9 Mbits/sec [ 5] 3.03-4.00 sec  3.02 MBytes 26.1 Mbits/sec [ 5] 4.00-5.05 sec  3.29 MBytes 26.4 Mbits/sec [ 5] 5.05-6.03 sec  2.92 MBytes 25.0 Mbits/sec [ 5] 6.03-7.00 sec  2.77 MBytes 24.0 Mbits/sec [ 5] 7.00-8.03 sec  2.96 MBytes 24.1 Mbits/sec [ 5] 8.03-9.02 sec  3.56 MBytes 30.3 Mbits/sec [ 5] 9.02-10.01 sec  2.79 MBytes 23.6 Mbits/sec [ 5] 10.01-10.15 sec  646 KBytes  36.1 Mbits/sec ----- [ ID] Interval      Transfer    Bandwidth  Retr [ 5] 0.00-10.15 sec 32.2 MBytes 26.6 Mbits/sec  62 [ 5] 0.00-10.15 sec 30.9 MBytes 25.5 Mbits/sec sender receiver</pre>									

Figura 9 Esempio di utilizzo di iperf3 da linea di comando

Iperf3 è una nuova versione del tool iperf, non retro-compatibile, completamente riscritta per creare un codice più semplice e utilizzabile anche da altri programmi sotto forma di libreria, ad ogni modo mentre iperf2 era multi-threaded, iperf3 è stato reso single-threaded quindi non particolarmente adatto ai test con più connessioni in parallelo (funzionalità che il tool continua a fornire).

# Capitolo 5 Presentazione progetto

In questo capitolo verrà prima descritta l'architettura del progetto originale che, dove possibile, è rimasta il più possibile invariata, si passerà poi a una descrizione delle varie parti e le responsabilità ricoperte da esse spiegando le motivazioni delle scelte prese.

## 5.1 Modeling4Cloud

Modeling4Cloud è il progetto realizzato dal collega Bagnacani Alberto, la quale architettura rappresenta il punto di partenza per lo sviluppo di questo sistema.

Nel progetto precedente tramite degli script era possibile avviare dei test eseguiti dallo strumento di ping ICMP del sistema operativo, i quali risultati, prima salvati su un file locale venivano poi inviati verso un database MongoDB centrale. Questo database era quindi interrogabile attraverso un'applicazione di back-end realizzata con Node.js che espose delle chiamate REST API. Al fine di mostrare i dati nella maniera più semplice e intuitiva il possibile la interfaccia web, utilizzando le chiamate api esposte dal back-end, permette di visualizzare i grafici riassuntivi dei test eseguiti. Lo sviluppo di questa parte è stato svolto con l'utilizzo di React per la realizzazione dell'interfaccia, associato all'utilizzo della libreria Chart.js per la realizzazione dei grafici.

Ad ogni modo il progetto presentava ancora delle problematiche:

- Le interrogazioni al database, gestite dal back-end utilizzando la pipeline di aggregazione però, una volta accumulati dati per qualche mese su un numero ristretto di datacenter hanno cominciato a comportare dei tempi di risposta eccessivamente lunghi.
- Per quanto riguarda la raccolta dati, il sistema si limitava solamente ai provider Amazon e Google in quanto erano i soli che permettevano tra le loro macchine un ping di tipologia ICMP (l'unico disponibile tramite lo strumento incluso nel sistema operativo)

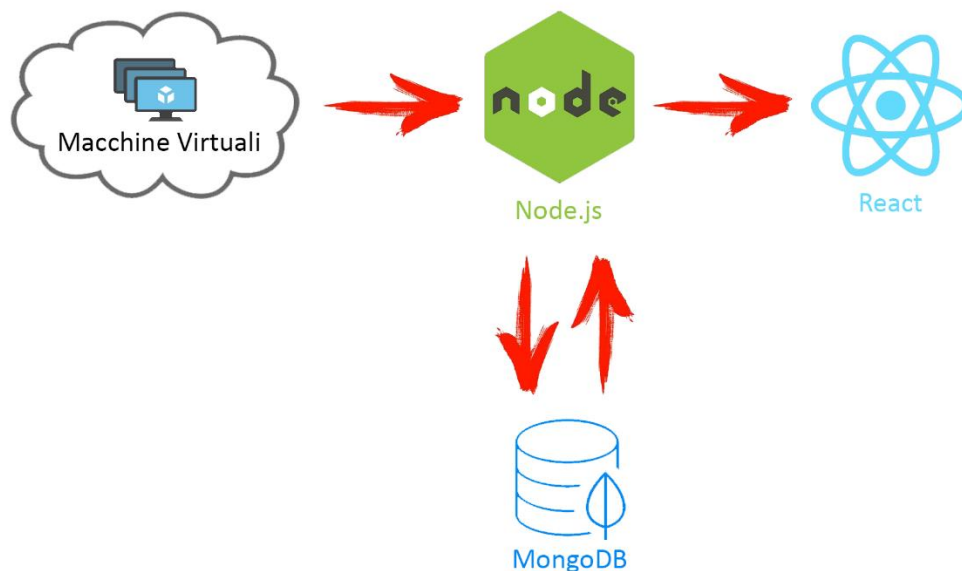
- Non era previsto alcun meccanismo per il controllo di banda e dei pacchetti persi.
- Inoltre non era presente un meccanismo di personalizzazione e configurazione dei test, che permettesse di gestire un monitoraggio su larga scala dei vari datacenter di un provider.

Le modifiche più sostanziali, come andremo a vedere, riguardano quindi la parte di test distribuita sulle varie macchine e la gestione della persistenza; mentre per l'applicazione web ci si limiterà ad una modifica per adattarla ai nuovi dati raccolti.

## 5.2 Parti e responsabilità

Il progetto, basandosi sull'utilizzo di strutture distribuite ed agendo su vari nodi della rete può essere scomposto in tre parti separate:

- Raccolta dati: svolta su macchine virtuali distribuite sui Cloud Providers nei vari datacenter di interesse
- Salvataggio dati: gestito su un database MongoDB
- Gestione e Presentazione dati: gestito da una applicazione Web creata con Node.js, Express e React



*Figura 10 Parti del sistema distribuito*

## 5.3 Raccolta dati

### 5.3.1 Macchine Virtuali

Al fine di effettuare dei test di rete sui vari provider è necessario distribuire una rete di macchine virtuali (stazioni di probing) sui vari nodi dei vari provider che si vogliono monitorare. Per ognuna di queste macchine, per convenienza in termini di costo, comodità e uniformità è stato scelto il sistema operativo gratuito Ubuntu, in quanto facilmente installabile indipendentemente dal provider scelto e particolarmente adatto per la gestione dei test di rete grazie all' evoluto linguaggio shell<sup>1</sup> (linea di comando) di cui dispone.

L'utilizzo delle varie macchine per i primi test è stato facile e gratuito grazie all'utilizzo di account utente gratuiti (con limiti di utilizzo) messi a disposizione dai cloud provider per gli studenti.

### 5.3.2 Gestione dei test

La gestione, installazione e esecuzione di tutti i test e delle relative impostazioni è completamente automatizzata sulle macchine, tramite l'utilizzo di una serie di script bash, che connettendosi ad ogni host tramite una connessione ssh<sup>2</sup>, installano tutti i software necessari, pianificano l'esecuzione dei test secondo quanto definito nel file di impostazioni e il caricamento dei risultati verso l'indirizzo specificato.

La configurazione di questi test è gestita dai file di impostazioni, uno per ogni cloud provider che si desidera monitorare, dove all'interno vengono elencate le macchine che devono essere utilizzate.

---

<sup>1</sup> La shell è quella parte di sistema operativo che permette agli utenti di interagire con esso, utilizzando comandi e avviando altri programmi.

<sup>2</sup> SSH (Secure Shell) è un protocollo che permette di aprire una sessione remota verso un altro host tramite una interfaccia a riga di comando. Sostituisce l'analogo ma insicuro Telnet, permettendo una cifratura della connessione tramite chiave asimmetrica.

Per ogni macchina, devono essere specificati:

- Il file contenente la chiave privata (di estensione .pem), utilizzato per l'autenticazione della connessione SSH
- L'indirizzo IP pubblico
- La zona geografica sul quale la macchina è ospitata
- L'indirizzo IP privato, se si intende utilizzare la macchina per dei test interni alla rete del provider

Sulle macchine sono quindi eseguibili 2 tipologie di test:

- Ping TCP
- Banda TCP

#### **5.3.2.1 Ping TCP**

Al fine di proteggere le macchine clienti da attacchi DoS<sup>1</sup> tramite ping ICMP, molti cloud provider scelgono di bloccare qualsiasi ping ICMP all'interno delle proprie reti. Per questo motivo è stato scelto di utilizzare lo strumento hping, misurando tempo di latenza (round trip time) e numero di hop intermedi (attraverso il parametro time to live) tramite quello che viene chiamato un ping TCP. Questa tipologia di soluzione sarà utilizzabile su tutti i cloud provider che si desidera testare in quanto ognuno di essi permette la configurazione delle connessioni TCP in entrata alla macchina sulle varie porte. A tale scopo basta configurare la macchina destinataria (quella che riceve il messaggio TCP) abilitando le connessioni su una determinata porta.

Bisogna però tenere in considerazione, che nel caso di hping, se si vogliono effettuare più test contemporaneamente, per evitare che vi siano dei conflitti è consigliabile utilizzare porte diverse.

Per il test, la porta TCP utilizzata e l'intervallo di tempo tra un ping ed il successivo sono configurabili tramite il file di impostazioni del provider, inoltre si può scegliere se la misurazione della latenza tra le zone deve essere effettuata in maniera monodirezionale o bidirezionale.

---

<sup>1</sup> Un attacco DoS mira a impedire l'uso di una risorsa di rete, solitamente attraverso un sovraccarico della vittima tramite l'invio di un elevato numero di pacchetti UDP o ICMP

### 5.3.2.2 Banda TCP

Il test di banda, effettuato grazie ad iperf3, consiste nell'aprire una connessione tra le due macchine (client e server) e cominciare a trasmettere dati dal client verso il server. Come nel caso del ping TCP, l'unica cosa necessaria è avere una porta aperta sul server. Utilizzando il protocollo TCP, oltre alla velocità di banda è anche possibile misurare quanti pacchetti inviati dal client non vengono ricevuti dal server (pacchetti persi), che quindi devono essere ritrasmessi.

Per il test di banda, tramite il file di configurazione possono essere impostati:

- Porta TCP da utilizzare per il test (su client e server)
- Intervallo di tempo tra un test e l'altro
- Durata del test in secondi
- Numero di connessioni in parallelo

Per quanto riguarda la durata, un valore più alto ovviamente garantisce una maggiore affidabilità e precisione, infatti svolgendo il test con una connessione TCP l'avvio lento<sup>1</sup> potrebbe falsificare i valori del primo periodo, ma questo porta anche ad un utilizzo dati più pesante. Come nel caso del ping anche per la banda si può effettuare un test bidirezionale o monodirezionale.

---

<sup>1</sup> TCP slow-start: è un algoritmo per il controllo di congestione implementato dal protocollo TCP, consiste nell'avviare la connessione trasmettendo un numero ridotto di pacchetti aumentandoli progressivamente in base a come la rete riesce a gestirli. Questo procedimento fa sì che i primi istanti di una connessione TCP siano più lenti della reale velocità raggiungibile

### 5.3.3 Risultati dei test

Ogni volta che un test viene eseguito il risultato viene prima salvato in locale come una nuova riga su un file di estensione .csv<sup>1</sup>, tutti i risultati vengono salvati in file diversi in base al tipo di test, la data in cui vengono eseguiti e un numero di sequenza generato automaticamente utile per distinguere quali erano i due nodi coinvolti in tale test.

Le informazioni comuni ad entrambi i test sono:

- Nome del provider
- Denominazione zona geografica macchina mittente
- Denominazione zona geografica macchina destinataria
- Indirizzo IP macchina mittente
- Indirizzo IP macchina destinataria
- Data e orario di esecuzione del test

Nel caso del ping TCP vengono salvati:

- Numero incrementale della sequenza di ping
- Time to live (TTL)
- Tempo di latenza in millisecondi (round trip time)

Nel caso del test di banda:

- Velocità di banda in MB/s
- Durata del test in secondi
- Numero di connessioni parallele
- Dati inviati dal mittente in MB
- Numero di pacchetti ritrasmessi

Questi file al termine di ogni giornata vengono caricati sul database centrale ed eliminati dalla macchina locale per limitare lo spazio utilizzato su disco su ogni macchina.

---

<sup>1</sup> CSV (Comma Separated Values) è un formato di file di testo utilizzato per importare/esportare i dati da un qualsiasi tipo di tabella

## 5.4 Salvataggio dei dati

Al termine di ogni giornata quindi, ogni macchina provvede ad inviare al database i risultati di tutti i test effettuati. Ogni risultato viene salvato come un documento nella relativa collezione (pings oppure bandwidths).

```
_id: ObjectId("5b96741b78ee6e5fbd9120e3")
provider: "AWS"
from_zone: "eu-west-2c"
to_zone: "eu-west-3c"
from_host: "18.130.223.116"
to_host: "35.180.92.154"
icmp_seq: 0
ttl: 48
time: 8.9
timestamp: 2018-07-26 18:50:26.000
```

*Figura 11 Ping salvato su MongoDB*

```
_id: ObjectId("5b96757a78ee6e5fbdfd3b12")
provider: "AWS"
from_zone: "eu-west-2c"
to_zone: "eu-west-3c"
from_host: "18.130.223.116"
to_host: "35.180.92.154"
timestamp: 2018-08-30 22:16:11.000
bandwidth: 266
duration: 1
parallel: 1
transfer: 31.7
retransmissions: 75
```

*Figura 12 Test di banda salvato su MongoDB*

Inizialmente il database era rappresentato da una singola istanza senza alcun tipo di replicazione o sharding, ma come è stato possibile analizzare dall'implementazione di Bagnacani questa soluzione portava a dei tempi di risposta troppo lunghi già dopo un periodo di analisi relativamente corto.



Basti pensare infatti che nel caso venga effettuato un ping ogni secondo, una sola macchina produce 86400 documenti al giorno.

Avendo come scopo principale quello di rendere il sistema il più scalabile ed efficiente possibile sono state esplorate diverse soluzioni, che integrate una con l'altra hanno permesso di ridurre i tempi di risposta risolvendo in pochi millisecondi delle interrogazioni che prima richiedevano decine di secondi o addirittura minuti.

#### 5.4.1 Cluster

La prima misura che è stata introdotta è stata quella di creare un cluster, distribuendo i dati su due Shard separati. Dividendo i dati sui due database in base al provider di appartenenza ha permesso far scalare orizzontalmente il sistema su più macchine, potendo utilizzare più potenza di calcolo soprattutto durante la risoluzione delle query.

Inoltre, tramite l'utilizzo di Replica Sets si è introdotta la possibilità di replicare gli Shard (quindi anche i dati che essi contengono) su più nodi, introducendo una capacità di resistenza ai guasti che prima non era prevista.

Dal punto di vista delle performance questo ha portato ad un drastico taglio nei tempi di risposta, in quanto i dati che prima venivano analizzati da una sola macchina, ora vengono gestiti da due macchine in parallelo. Questo ha portato, soprattutto grazie alla riduzione della quantità di dati che ogni macchina deve salvare in RAM mentre opera, ad una riduzione dei ritardi superiore al 50%.

Nonostante i risultati ottenuti avessero migliorato la situazione, effettuare un'analisi tramite aggregazione su risultati di test dell'intervallo di qualche mese richiedeva ancora una quantità di tempo che non era accettabile per l'utilizzo in una applicazione real-time.

#### 5.4.2 Ottimizzazione aggregazione e indicizzazione

Per ognuna delle query che veniva implementata dal back-end per ottenere un riassunto significativo era necessario ricorrere ad una pipeline di aggregazione. Questo potente strumento, permettendo di raggruppare

milioni di documenti in un riassunto di pochi risultati però, richiede di accedere ad ognuno di essi, scaricandoli in memoria centrale, che oltre ad impiegare tempo per il trasferimento dal disco fisso alla RAM può anche arrivare a riempirla nel caso si lavori con una quantità di dati maggiore della capacità di RAM disponibile, costringendo all'utilizzo di un file di paging su disco fisso con conseguenti peggioramenti di performance. Data la quantità di dati generati giornalmente questo scenario era prevedibile, portando quindi ad un'estrema inefficienza.

Per limitare l'impatto che la mole di dati aveva sull'efficienza sono stati introdotti degli indici e di conseguenza le operazioni delle pipeline di aggregazione sono state modificate in maniera da trarre il maggior vantaggio possibile introdotto dagli indici, soprattutto durante i primi stadi della pipeline, dove i dati che vengono analizzati sono più numerosi.

Questo processo di ottimizzazione ha portato ad ottimi risultati, più o meno evidenti in base alla query specifica, ad ogni modo una grande limitazione di MongoDB è quella di non supportare l'utilizzo degli indici durante lo stadio di grouping (raggruppamento) che risulta essere il più pesante, soprattutto in quelle query che mirano a calcolare una media dei risultati di vari test in una fascia di tempo.

Mentre le due soluzioni precedenti si sono rivelate più che sufficienti per la risoluzione delle query relative ai test di banda (che possono arrivare ad avere una frequenza di al massimo qualche decina di test al giorno) per quanto riguarda i ping, avendo una frequenza molto più elevata era ancora necessaria una pesante ottimizzazione.

### 5.4.3 Precomputing

La soluzione definitiva al problema dei tempi di risposta delle query sui ping si è rivelata quella di effettuare un precomputing<sup>1</sup> delle medie giornaliere su ogni zona.

Tutte le query che vengono effettuate verso il database richiedono una media di tutti i risultati dei ping effettuati per date, processo che, dovendo

---

<sup>1</sup> Con il termine precomputing si intende il calcolo e salvataggio preventivo di un valore che si prevede potrà rivelarsi utile più volte, tenendolo pronto per l'uso invece di ricalcolarlo ogni volta.

analizzare fino a 86400 risultati per ogni combinazione di zone geografiche disponibili, è senz'altro quello che richiede più risorse. Volendo comunque mantenere disponibili sul database tutti i risultati generati, si è deciso di introdurre una nuova collezione “pingdayavgs” contenente le medie giornaliere di una stazione di probing verso un'altra.

Ogni documento della collezione quindi conterrà i seguenti campi:

- Provider interessato
- Denominazione zona geografica macchina mittente
- Denominazione zona geografica macchina ricevente
- Data
- Media di tutti i test effettuati
- Numero di test effettuati

```
_id: ObjectId("5b969fb87baf7f1cd0d973cf")
provider: "GCP"
from_zone: "eu-west1-b"
to_zone: "eu-west2-a"
day: 2018-08-19 00:00:00.000
avg: 13.262972866601842
count: 86400
```

*Figura 13 Precomputing media giornaliera su MongoDB*

Il calcolo di queste medie, viene effettuato dal back-end, dopo aver ricevuto e salvato sul database il dettaglio di tutti i test di una determinata data, informazione che si è deciso di continuare a mantenere nel caso sia richiesta una consultazione dei dati più accurata.

In questo modo i tempi di risoluzione di una query si sono potenzialmente ridotti di 86400 volte, rendendo il sistema finalmente in grado di fornire in meno di un secondo dettagliati rapporti anche su ampi intervalli di date.

## 5.5 Consultazione dei dati

I dati salvati sul database MongoDB sono resi accessibili all'esterno dai punti di accesso esposti dal back-end eseguito su Node.js. Con lo scopo di fornire una visualizzazione dei risultati più intuitiva e significativa, la parte di front-end sviluppata con React, fornisce varie tipologie di grafici sui risultati.

La parte di backend quindi si occupa quindi di gestire tre funzionalità:

- Ricevere i risultati da ognuna delle macchine virtuali di test
- Convertire i risultati per aggiungerli al database, con relativo calcolo delle medie giornaliere nel caso dei ping
- Fornire delle api REST per consultare i dati salvati

### ***5.5.1.1 Ricezione risultati***

Per la ricezione dei risultati sono stati definiti due appositi endpoint api, ai quali ogni macchina di probing, tramite una chiamata http POST può effettuare l'upload del file csv contenente tutti i risultati del corrispondente test al termine della giornata.

### ***5.5.1.2 Conversione dei risultati e aggiunta al database***

Ogni volta che un nuovo file csv viene caricato, attraverso il metodo appena descritto, ha inizio la fase di importazione dei risultati, dal file al database.

Nel caso dell'upload di nuovi risultati di ping, è il backend che si prende in carico di effettuare il calcolo della media giornaliera (in base ai risultati ricevuti) e aggiungerla al database.

### ***5.5.1.3 Esposizione dei dati***

Tutte le interrogazioni sui dati provenienti dall'esterno (dall'applicazione web) non verranno indirizzate direttamente al database, ma vengono effettuate al back-end, dove per ogni tipologia di informazione che si prevede di richiedere è stato fornito un apposito end-point api.

Inoltre, per poter visualizzare direttamente i dati, per ognuna delle collezioni presenti sul database (pings, bandwidths e pingdayavgs) è stato fornito un

endpoint per consultare tutti i dati presenti tramite una paginazione dei risultati.

Per facilitare il lavoro dell'applicazione web, per ognuna delle due tipologie di test sono stati definiti degli endpoint che forniscano i dati, riassunti in base alle tipologie di informazioni che si desidererà mostrare sull'interfaccia utente. Le funzionalità raggiungibili dagli endpoint sono:

- Media di ogni provider sulle varie zone potendo filtrare solamente i test tra regioni diverse o sulla stessa regione
- Media di tutti i test tra zone di uno specificato provider
- Media di tutti i test da una zona specificata a tutte le altre e viceversa

## 5.6 Budget e costi

Inizialmente è stato possibile utilizzare degli account, gratuitamente offerti dai cloud provider agli studenti (ad esclusione di IBM), per effettuare dei test del sistema sui vari fornitori anche se su un numero limitato di zone.

Successivamente, grazie all'utilizzo di account forniti e finanziati dall'azienda Imola Informatica [7] è stato possibile testare il sistema anche sul provider rimanente (IBM) e su un numero più ampio di regioni. Ad ogni modo questo ha introdotto la necessità di effettuare un'attenta analisi dei costi e delle risorse utilizzate.

Come è già stato sottolineato nel primo capitolo, nell'ambito cloud l'abitudine standard è quella di pagare una risorsa in base all'effettivo utilizzo, nella parte seguente si cercherà quindi di fornire un quadro dettagliato relativo a tutti i costi previsti per quanto riguarda l'insieme delle macchine virtuali per la raccolta dati sui vari provider.

### 5.6.1 Tipologia, costo e numero istanze

La maggior parte del costo mensile da sostenere è costituito dal costo fisso che il mantenimento di una macchina virtuale su un provider richiede.

Su ognuno dei vari provider vengono offerte tipologie di macchine di prestazioni differenti, dato che non sono richieste particolari risorse per la gestione dei test è sempre stata presa in considerazione quella a costo minore

al fine di ridurre i costi. Bisogna sottolineare però che, solitamente anche la velocità di connessione viene limitata in base alla tipologia di risorsa, quindi anche questo diventerà un fattore da considerare nei nostri test.

Su ogni provider, in base ai test che si intende effettuare si richiede:

- Una macchina per ogni zona geografica per effettuare i test tra un'istanza in quella zona relativamente alle altre
- Una macchina aggiuntiva per ogni zona geografica per effettuare un test delle prestazioni tra due istanze della stessa zona

#### **5.6.1.1 Amazon Web Services**

Nel caso di Amazon l'istanza più economica è chiamata t2.nano, dotata di:

- 1 CPU virtuale
- 0,5 GB di RAM
- 10 GB di spazio di disco fisso (Hard Disk)

Il costo dell'istanza varia in base all'area geografica scelta, da un minimo di 4,75\$ mensili, fino ad un massimo di 8\$. Per effettuare un test tra tutte le 15 regioni pubbliche offerte dal provider il costo mensile totale delle istanze sarebbe 85,36\$. Quindi nel caso si volesse integrare anche il test interno alle zone si salirebbe a 170,72\$ mensili.

#### **5.6.1.2 Google Cloud Platform**

Sulla piattaforma Google, l'istanza soprannominata *f1-micro* fornisce:

- 1 CPU virtuale
- 0,6 GB di RAM
- 10 GB di disco fisso (Hard Disk)

Il costo mensile delle istanze su ogni regione varia da un minimo di 5,95\$ fino a 9,21\$. Considerando un'istanza su ognuna delle 17 regioni offerte dal provider si arriverebbe ad un costo mensile di 85,64\$. Nel caso di due istanze per regione 171,28\$.

#### **5.6.1.3**

#### **5.6.1.4 Microsoft Azure**

Sulla piattaforma Microsoft, l'istanza soprannominata *A0* fornisce:

- 1 CPU virtuale
- 0.75 GB di RAM
- 20 GB di disco fisso (Hard Disk)

Il costo mensile delle istanze su ogni regione varia da un minimo di 11,68\$ fino a 21,90\$. Considerando un'istanza su ognuna delle 32 regioni offerte dal provider si arriverebbe ad un costo mensile di 468\$. Nel caso di due istanze per regione 936\$.

#### **5.6.1.5 IBM Cloud**

Sulla piattaforma IBM, l'istanza soprannominata *C1* fornisce:

- 1 CPU virtuale
- 1 GB di RAM
- 25 GB di disco fisso (Hard Disk)

Il costo mensile delle istanze su ogni regione è uguale a 26,98\$. Considerando un'istanza su ognuna delle 25 regioni offerte dal provider si arriverebbe ad un costo mensile di 674\$. Nel caso di due istanze per regione 1349\$.

### **5.6.2 Utilizzo di rete**

Per ognuna delle istanze fornite sui vari provider viene inclusa gratuitamente una quantità di traffico di rete utilizzabile mensilmente, questa quantità però non risulterà sufficiente per ricoprire i bisogni della rete di test a causa dell'elevato scambio di traffico dei test di banda, introducendo un costo variabile in base all'utilizzo effettivo della rete.

L'utilizzo di dati mensile di ogni test di ping TCP è di circa 1,5 GB al mese.

Per quanto riguarda la rete utilizzata dal test di banda, invece il calcolo risulta molto più complicato, variando in base alle regioni tra il quale viene effettuato, infatti avendo il test una durata fissa, la quantità di dati trasmessi varia in base alla velocità di banda. Per una stima più accurata sarebbe necessario lanciare un test preliminare su tutte le regioni (di breve durata per limitare i costi) in modo da avere una stima di tutte le velocità tra le varie

regioni, potendo poi andare a calcolare il costo mensile che dei test su tali connessioni comporterebbero. Come vedremo successivamente, questo procedimento è stato seguito solamente sul provider Amazon, per cui in questa fase ci si limiterà a fornire i prezzi di rete in base all'utilizzo per fornire un confronto tra i vari provider.

#### **5.6.2.1 Amazon Web Services**

Traffico mensile incluso per ogni regione geografica: 1 GB

Traffico interno alle regioni gratuito

Costo per ogni GB di utilizzo mensile extra:

- Regioni in Europa e Nord-America: 0,09\$
- Regioni in Asia: 0,11~0,14\$
- Regioni in Sud-America: 0,25\$

#### **5.6.2.2 Google Cloud Platform**

Traffico mensile incluso per ogni regione geografica: 1 GB

Costo per ogni GB di utilizzo mensile extra:

- Regioni in Europa e Nord-America: 0,09\$
- Regioni in Asia: 0,11~0,14\$
- Regioni in Sud-America: 0,25\$

#### **5.6.2.3 Microsoft Azure**

Traffico mensile incluso per ogni regione geografica: 5 GB

Traffico interno alle regioni gratuito

Costo per ogni GB di utilizzo mensile extra:

- Regioni in Europa e Nord-America (escluso Germania): 0,087\$
- Germania: 0,1\$
- Regioni in Asia: 0,12\$
- Regioni in Sud-America: 0,181\$

#### **5.6.2.4 IBM Cloud**

Nessun traffico mensile incluso, Traffico interno alle regioni gratuito

Costo per ogni GB di utilizzo mensile: 0,09\$



### 5.6.3 La scalabilità su Amazon Web Services

In conclusione del progetto, si è deciso di effettuare un test reale della scalabilità del sistema e della capacità di gestire un monitoraggio completo su vari datacenter in modo da verificarne il costo di mantenimento. In base al quadro costi esposto nella parte precedente risulta chiaro che per un test su larga scala di tutti i provider interessati il budget richiesto sarebbe eccessivamente alto. Per questo motivo si è deciso di effettuare un test limitato al provider Amazon in quanto il più economico.

Il test in questione prevedeva il monitoraggio della latenza ogni 10 secondi, ed un test di banda ogni 25 ore in modo da ricoprire i vari orari della giornata, dove ogni test durava 1 secondo. Entrambi i test sono stati configurati tra tutte le varie regioni in modalità monodirezionale (cioè la misurazione tra due regioni A e B viene effettuata solamente da A verso B e non da B verso A) per ridurre i costi di utilizzo della rete.

Per fornire una stima preventiva dei costi che si andranno a sostenere è stato prima lanciato un test di banda tra le varie regioni per stabilire indicativamente la quantità di dati trasferiti durante ogni test, constatando che per sostenere entrambi i test di banda e ping i costi variabili mensili legati all' utilizzo di rete ammontano a circa 15\$.

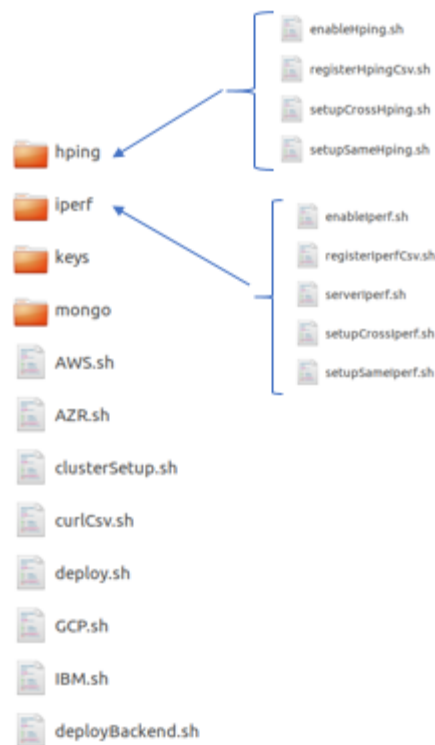
Concludendo quindi, il budget mensile richiesto, con le configurazioni attuali è di 85,36\$ mensili per le istanze, più 15\$ per i costi di rete, raggiungendo un totale di 100\$ mensili.

# Capitolo 6 Implementazione soluzione

In questo capitolo verranno analizzate le parti principali del sistema, esponendo nel dettaglio come le funzionalità più importanti sono state implementate.

## 6.1 Script per la raccolta dati

Le parti di codice che verranno descritte di seguito sono gli script di codice bash che in parte, vengono eseguiti da una macchina locale, per installare e configurare i test, in parte invece verranno periodicamente eseguiti sulle varie macchine per la raccolta delle statistiche.



*Figura 14 Panoramica dei file di script*

### 6.1.1 File di configurazione

Per ogni provider è previsto un file di configurazione contenente tutte le impostazioni relative ai test:

- Nome del provider in questione
- Porta di partenza per i test di hping (i test saranno effettuati tutti su porte consecutive a partire da questa)
- Intervallo di tempo in secondi tra i test di hping
- Bidirezionalità dei test di ping (0: Disattivata, 1: Attivata)
- Porta per i test di banda
- Intervallo di tempo in ore tra i test di banda
- Durata in secondi dei test di banda
- Numero di connessioni parallele per i test di banda
- Bidirezionalità dei test di banda (0: Disattivata, 1: Attivata)

Seguite da due array, contenente le informazioni relative:

- Alle macchine per i test tra regioni diverse (file chiave autenticazione, indirizzo IP pubblico, nome identificativo della zona)
- Alle macchine per i test interni delle regioni (file chiave autenticazione, indirizzo IP pubblico, nome identificativo della zona, indirizzo IP privato)

Nel caso dei test interni è richiesto l'IP privato delle macchine in maniera da sfruttare il traffico gratuito che i provider offrono nel caso di indirizzamento privato per le macchine all'interno della stessa regione.

### AWS.sh

```
#Global settings
PROVIDER=AWS

#Hping settings
HPINGBASEPORT=6000
HPING_SECONDS_INTERVAL=10
HPING_BIDIRECTIONAL=0

#Iperf settings
PORT=80
IPERF_HOUR_INTERVAL=25
DURATION=1
PARALLEL=1
```

```

IPERF_BIDIRECTIONAL=0

CROSS_REGION_VMS=(
    "./keys/virginia.pem XX.XX.XX.XX us-east-1"
    ...
)

SAME_REGION_VMS=(
    "./keys/virginia.pem XX.XX.XX.XX us-east-1 YY.YY.YY.YY"
    ...
)

```

### 6.1.2 Installazione dei test

Il file di configurazione appena descritto viene utilizzato dagli script per l'installazione dei test sulle macchine remote. Per ognuno dei 2 tipi di test (ping e banda) è previsto uno script per la gestione dei test tra regioni diverse ed uno per i test interni alle regioni (Successivamente per brevità ne vengono mostrati solamente i due più significativi dei quattro).

In ogni caso all'avvio dello script, con un ciclo su ogni macchina interessata vengono installati tutti gli strumenti necessari tramite lo strumento apt<sup>1</sup> e copiati tramite scp<sup>2</sup> tutti gli script che andranno eseguiti.

Successivamente, leggendo i dati delle macchine dal file di impostazioni, nel caso dei ping vengono direttamente avviati i monitoraggi, invece per i test di banda vengono prima messi in ascolto i server e poi avviate le stazioni di monitoraggio per i test.

#### setupSameRegionHping.sh

```

CONFIG=$1
BACKENDADDR=$2

source $CONFIG

for a in "${SAME_REGION_VMS[@]}"
do
    KEY=$(echo $a | awk '{print $1}')
    HOST=$(echo $a | awk '{print $2}')

    ssh -o StrictHostKeyChecking=no -i $KEY ubuntu@$HOST bash -c "
        sudo apt-get install expect -qq
    "
done

```

<sup>1</sup> Advanced Packaging Tool: strumento standard per la gestione di pacchetti software nei sistemi Debian (quindi anche del suo derivato ubuntu)

<sup>2</sup> Secure CoPy è un protocollo per il trasferimento sicuro di file tramite il protocollo SSH

```

        sudo apt-get install hping3 -qq"

scp -r -i $KEY ./hping/enableHping.sh ubuntu@$HOST:~
scp -r -i $KEY ./hping/registerHpingCsv.sh
ubuntu@$HOST:~/Modeling4Cloud/utils/
scp -r -i $KEY ./curlCsv.sh ubuntu@$HOST:~/Modeling4Cloud/utils/
done

for i in "${SAME_REGION_VMS[@]}"
do
    KEY=$(echo $i | awk '{print $1}')
    FROMHOST=$(echo $i | awk '{print $2}')
    FROMZONE=$(echo $i | awk '{print $3}')
    FROMPRIVATE=$(echo $i | awk '{print $4}')
    for j in "${SAME_REGION_VMS[@]}"
    do
        KEYTOHOST=$(echo $j | awk '{print $1}')
        TOHOST=$(echo $j | awk '{print $2}')
        TOZONE=$(echo $j | awk '{print $3}')
        TOPRIVATE=$(echo $j | awk '{print $4}')
        PORT=$HPINGBASEPORT

        if [ "$FROMZONE" = "$TOZONE" ]; then
            echo "Setup Hping $FROMHOST($FROMZONE) to $TOHOST($TOZONE)"
            ssh -i $KEY ubuntu@$FROMHOST bash -c "'.enableHping.sh
$PROVIDER $FROMZONE $TOZONE $FROMPRIVATE $TOPRIVATE $SEQNUMBER $PORT
$BACKENDADDR $HPING_SECONDS_INTERVAL'"
        fi
    done
done
done

```

## setupCrossRegionIperf.sh

```

CONFIG=$1
BACKENDADDR=$2
source $CONFIG
SEQNUMBER=1
for a in "${CROSS_REGION_VMS[@]}"
do
    KEY=$(echo $a | awk '{print $1}')
    HOST=$(echo $a | awk '{print $2}')
    #SERVER
    scp -r -i $KEY ./iperf/serverIperf.sh ubuntu@$HOST:~
    #CLIENT
    ssh -o StrictHostKeyChecking=no -i $KEY ubuntu@$HOST bash -c "'
        sudo apt-get install iperf3 -qq'"
    scp -r -i $KEY ./iperf/enableIperf.sh ubuntu@$HOST:~
    scp -r -i $KEY ./iperf/registerIperfCsv.sh
ubuntu@$HOST:~/Modeling4Cloud/utils/
    scp -r -i $KEY ./curlCsv.sh ubuntu@$HOST:~/Modeling4Cloud/utils/
done
iCount=1

```

```

for i in "${CROSS_REGION_VMS[@]}"
do
    KEYFROMHOST=$(echo $i | awk '{print $1}')
    FROMHOST=$(echo $i | awk '{print $2}')
    FROMZONE=$(echo $i | awk '{print $3}')
    jCount=1
    for j in "${CROSS_REGION_VMS[@]}"
    do
        KEYTOHOST=$(echo $j | awk '{print $1}')
        TOHOST=$(echo $j | awk '{print $2}')
        TOZONE=$(echo $j | awk '{print $3}')
        if ! [ "$FROMZONE" = "$TOZONE" ]; then
            if [ "$IPERF_BIDIRECTIONAL" -eq "1" -o "$jCount" -gt
"$iCount" ]; then
                echo "Enable Iperf from $FROMHOST($FROMZONE) to
$TOHOST($TOZONE)"
                #SETUP SERVER
                ssh -o StrictHostKeyChecking=no -i $KEYTOHOST
ubuntu@$TOHOST bash -c "'. /serverIperf.sh $PORT'"
                #SETUP CLIENT
                ssh -i $KEYFROMHOST ubuntu@$FROMHOST bash -c
                "'. /enableIperf.sh $PROVIDER $FROMZONE $TOZONE $FROMHOST $TOHOST $PORT
$SEQNUMBER $BACKENDADDR $IPERF HOUR_INTERVAL $DURATION $PARALLEL'"
                sleep $((DURATION*2)) # Delay to avoid overlap of
different bandwidth tests
            fi
            SEQNUMBER=$((SEQNUMBER+1))
        fi
        jCount=$((jCount+1))
    done
    iCount=$((iCount+1))
done

```

L'esecuzione remota sulle macchine degli script successivi è quella che si occupa della pianificazione dei test e del caricamento dei risultati alla fine di ogni giornata.

Per pianificare l'esecuzione dello script del caricamento dei dati (curlCsv.sh) è stato utilizzato lo strumento di sistema crontab, particolarmente adatto per pianificare un determinato comando da eseguire periodicamente ad un determinato orario (nel nostro caso mezzanotte del fuso orario UTC).

Lo script registerHpingCsv.sh che viene lanciato è il responsabile dell'esecuzione dei test e del salvataggio dei loro risultati sul file csv locale. Viene lanciato seguendo alcuni accorgimenti:

- Lanciato in background con l'utilizzo del carattere & finale

- Distaccato dal terminale corrente con il comando `nohup` [8], evitando che venga terminato quando la connessione ssh di setup viene chiusa
- Direzionando standard output ed error su file in modo da avere dei log da consultare in caso di necessità

## enableHping.sh<sup>1</sup>

```

PROVIDER=$1
FROMZONE=$2
TOZONE=$3
FROMIP=$4
TOIP=$5
SEQ_NUMBER=$6
PORT=$7
BACKEND_ADDR=$8
INTERVAL=$9

nohup ~/Modeling4Cloud/utils/registerHpingCsv.sh $PROVIDER $FROMZONE
$TOZONE $FROMIP $TOIP $SEQ_NUMBER $PORT $INTERVAL > $FROMZONE-$TOZONE-
$SEQ_NUMBER.out 2> $FROMZONE-$TOZONE-$SEQ_NUMBER.err < /dev/null &

cline="~/Modeling4Cloud/utils/curlCsv.sh $PROVIDER $SEQ_NUMBER
$BACKEND_ADDR ~/csv"
if ! crontab -l | grep -q "$cline" ; then
    (crontab -l ; echo '0 0 * * *' "$cline" ) | crontab -
    echo Added hpingResultCurl crontab
else
    echo hpingResultCurl crontab already setup
fi

```

Per la pianificazione dei test si è deciso di utilizzare una soluzione diversa dal `crontab`, in quanto quest'ultimo non prevede alcuna tipologia di mantenimento dello stato rispetto all'ultima esecuzione di un comando, quindi rende impossibile far traslare gli orari di test di giorno in giorno in modo da coprire parti della giornata diverse. Al suo posto è stato utilizzato il comando `at` [9], pianificando ad ogni avvio dello script di testing un'altra esecuzione dello stesso script dopo l'intervallo di tempo desiderato.

Qui di seguito viene mostrato lo script di registrazione dei risultati per `iperf`, quello relativo ad `hping` non è stato inserito in quanto concettualmente analogo.

---

<sup>1</sup> Per brevità si è omissso lo script `enableIperf.sh`, analogo a quello di `hping` con argomenti di avvio diversi

### 6.1.3 Esecuzione dei test

#### registerIperfCsv.sh

```
< ... ARGS ... >

if [ ! -d ~/csvIperf ]; then
    mkdir ~/csvIperf
fi

echo "~/Modeling4Cloud/utils/registerIperfCsv.sh $PROVIDER $FROMZONE
$TOZONE $FROMHOST $TOHOST $PORT $NUMBER $HOURL_INTERVAL $DURATION
$PARALLEL $INTERVAL > iperf-$FROMZONE-$TOZONE-$NUMBER.out 2> iperf-
$FROMZONE-$TOZONE-$NUMBER.err < /dev/null" | at now + $HOURL_INTERVAL
hour 2> /dev/null

RESULT=$(sudo iperf3 -c $TOHOST -p $PORT -t $DURATION -P $PARALLEL -i
$DURATION -f M)

SENDER=$(echo "$RESULT" | tail -n 4 | head -1)
RECEIVER=$(echo "$RESULT" | tail -n 3 | head -1)

TODAY=$(date +%Y-%m-%d)
TIMESTAMP=$(date "+%Y-%m-%dT%H:%M:%S-00:00")

if [ $PARALLEL -eq 1 ];then
    TRANSFER=$(echo $SENDER | awk '{print $5}' | cut -d= -f2)
    TRANSFERUNIT=$(echo $SENDER | awk '{print $6}' | cut -d= -f2)
    BANDWIDTH=$(echo $RECEIVER | awk '{print $7}' | cut -d= -f2)
    RETRANSMISSIONS=$(echo $SENDER | awk '{print $9}' | cut -d= -f2)
else
    TRANSFER=$(echo $SENDER | awk '{print $4}' | cut -d= -f2)
    TRANSFERUNIT=$(echo $SENDER | awk '{print $5}' | cut -d= -f2)
    BANDWIDTH=$(echo $RECEIVER | awk '{print $6}' | cut -d= -f2)
    RETRANSMISSIONS=$(echo $SENDER | awk '{print $8}' | cut -d= -f2)
fi

if [ $TRANSFERUNIT = "KBytes" -o $TRANSFERUNIT = "Bytes" ]; then
    TRANSFER=1
fi

FILE=~/csvIperf/$PROVIDER-$NUMBER-$TODAY.csv

if ! [ -s $FILE ]
then
    printf "provider,from_zone,to_zone,from_host,to_host,timestamp,
bandwidth,duration,parallel,transfer,retransmissions\n" >> $FILE
fi

printf "$PROVIDER,$FROMZONE,$TOZONE,$FROMHOST,$TOHOST,$TIMESTAMP,
$BANDWIDTH,$DURATION,$PARALLEL,$TRANSFER,$RETRANSMISSIONS\n" >> $FILE
```



### 6.1.4 Invio risultati

In base alla pianificazione effettuata tramite crontab dallo script di enable (enableHping.sh oppure enableIperf.sh in base ai casi) ogni giorno a mezzanotte (del fuso orario UTC) viene eseguito il lo script curlCsv.sh per ognuno dei file salvati durante la giornata. Lo script si occupa di inviare il file in questione all' URL specificato del backend.

#### curlCsv.sh

```
PROVIDER=$1
NUMBER=$2
SERVER=$3
FOLDER=$4

YESTERDAY=$(date -d "yesterday 13:00" '+%Y-%m-%d')
FILE=$FOLDER/$PROVIDER-$NUMBER-$YESTERDAY.csv

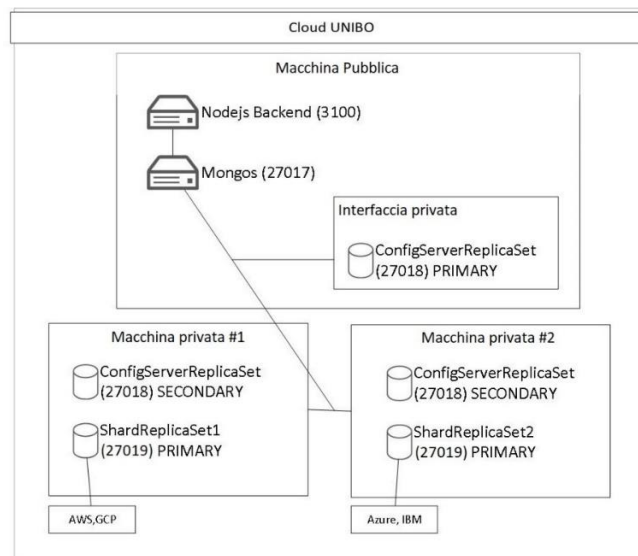
curl -F "data=@$FILE" $SERVER
sudo rm -f $FILE
```

## 6.2 Persistenza su MongoDB

Per l'implementazione del sistema sono state messe a disposizione, sulla rete privata dell'Università di Bologna, tre macchine virtuali. Per distribuire il carico applicativo sulle varie macchine si è deciso di collocare due Shard di MongoDB su due VM, mentre la macchina rimanente si occupava di gestire il Config Server e il router mongos (in aggiunta al backend con Node.js come vedremo più avanti).

Nell'architettura del cluster non è stato previsto alcun meccanismo per la replicazione dei dati, ad ogni modo sulle ultime versioni di MongoDB è richiesto che in ogni cluster, sia gli Shard contenenti i dati, sia i Config Server, facciano parte di un Replica Set, per questo motivo sono presenti nell'architettura le 3 componenti:

- ShardReplicaSet1: contenente il Replica Set (composto da un solo membro) del primo Shard dei dati (per i provider Amazon e Google)
- ShardReplicaSet2: contenente il Replica Set (composto da un solo membro) del secondo Shard dei dati (per i provider Microsoft e IBM)
- ConfigServerReplicaSet: contenente il Replica Set del Config Server



*Figura 15 Componenti della persistenza*

Per il database distribuito MongoDB è stato inserito un meccanismo di autenticazione per garantire una massima sicurezza. Questo meccanismo comprende due tipologie di autenticazioni:

- Autenticazione interna, tra i componenti distribuiti del cluster di MongoDB, tramite l'utilizzo di una chiave privata nota solo ai componenti legittimi.
- Autenticazione degli utenti, per l'accesso (locale o remoto) al database sono state inseriti appositamente creati due account utente, uno amministratore per la gestione del sistema e dei componenti del cluster, ed uno apposito per la lettura e scrittura dei dati dal backend (di privilegi più limitati)

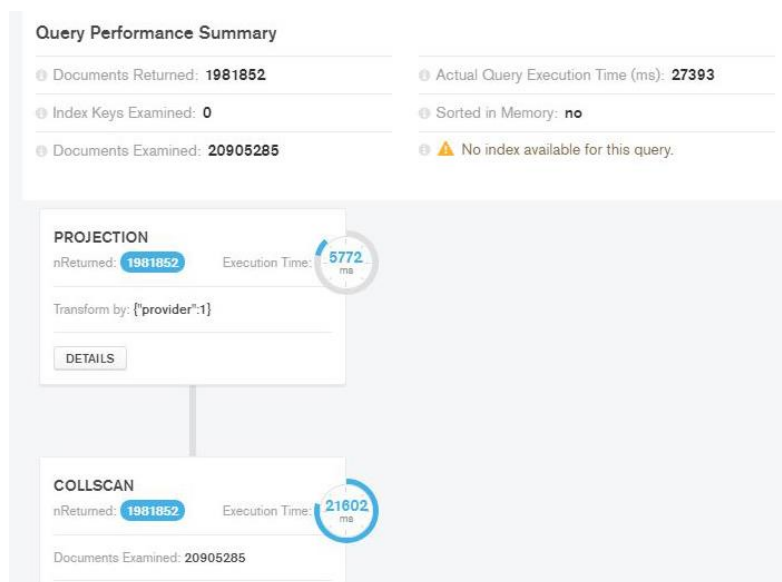
Si è scelto di suddividere i dati sugli Shard utilizzando come chiave il provider in maniera da poter eseguire in parallelo eventuali interrogazioni che verranno eseguite per uno specifico provider. Il database verrà sottoposto ad un utilizzo piuttosto intenso ogni mezzanotte (del fuso orario UTC, sul quale sono sincronizzate le macchine virtuali), momento nel quale vengono effettuati tutti i caricamenti dei risultati da parte delle macchine di probing, portando ad un calcolo di tutte le relative medie giornaliere. Scegliere come chiave di Sharding il nome del provider ha permesso di parallelizzare tutte le query che vengono effettuate al database (durante il precomputing) velocizzando il processo. Per quanto riguarda le collezioni di dati sulle quali vengono direttamente effettuate le interrogazioni dal backend ad ogni modo si è scelto di non effettuare alcuna suddivisione per evitare di aggiungere un grado di complessità non richiesto dato che un semplice database, ottimizzato tramite l'utilizzo di indici risulta più che sufficiente per gestire la quantità di dati con ottimi tempi di risposta.

### 6.2.1 Analisi delle performance

La rifinitura delle performance di MongoDB è stato uno degli obiettivi principali per quanto riguarda lo sviluppo dello strato di persistenza, portando a continui test delle varie soluzioni offerte da MongoDB per il miglioramento dei tempi di risposta. Per questo motivo di seguito viene

mostrato una comparazione quantitativa dei miglioramenti che le varie soluzioni hanno portato.

Come base di comparazione è stata utilizzata un'interrogazione che filtrasse i risultati in base al nome del provider in maniera da rispecchiare nella maniera più precisa possibile l'utilizzo che l'applicazione finale farà del database. Per ognuna delle modifiche applicate è stata eseguita la query verso il database e ne vengono mostrati i vari stadi dell'esecuzione con i relativi tempi richiesti:



*Figura 16 Punto di partenza, senza ottimizzazioni*

Come si può vedere dai tempi richiesti dai vari stadi, quello più oneroso è senz'altro quello di COLLSCAN, cioè durante il quale si effettua la selezione dei valori da restituire, che viene effettuata da MongoDB con un'analisi dei documenti uno ad uno. Per questo motivo è stato inserito un indice sulla collezione, che permettesse un controllo più veloce sui valori del campo.

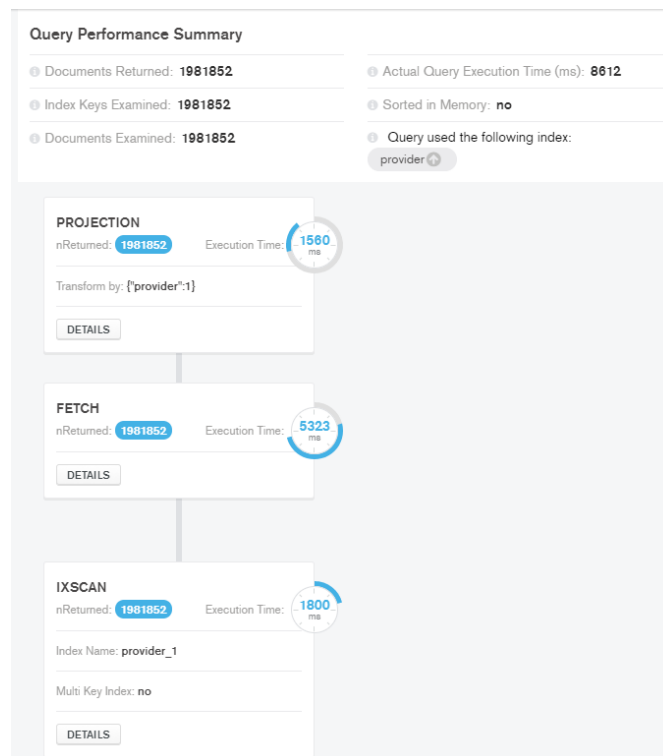
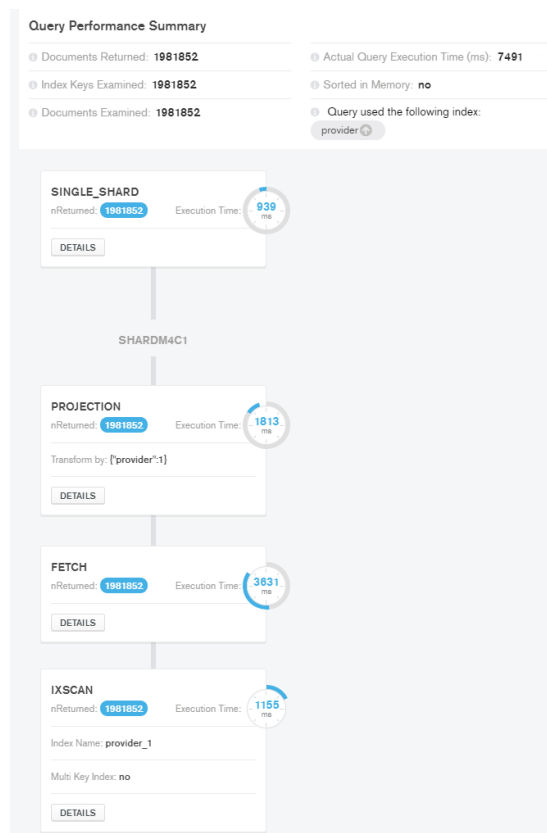


Figura 17 Aggiunta di un indice sulla collezione

La fase di COLLSCAN è stata sostituita da quelle di IXSCAN e FETCH, che si occupano rispettivamente di controllare i valori sul database e successivamente estrarli. Nell'esempio l'indice utilizzato è quello su provider data la query che viene effettuata, per quanto riguarda il progetto è necessario inserire degli indici adeguati (anche composti da più campi) in base alle query che vengono effettuate. Per ottenere una performance ottimale, quando possibile, è anche consigliato estendere i campi compresi nell'indice non solo a quelli sul quale si effettuano delle ricerche ma anche a quelli che vanno semplicemente restituiti, questo permette di effettuare le query senza consultare direttamente la collezione vera (query di questo tipo vengono chiamate *covered queries*).

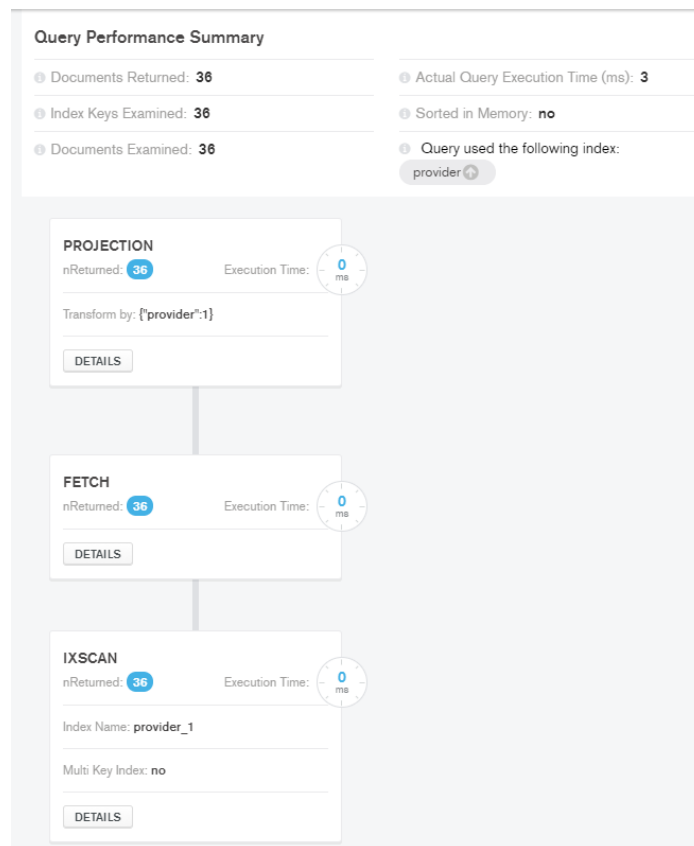
A questo punto è stato effettuato lo Sharding della collezione, dividendola sulle due macchine.



*Figura 18 Esecuzione query con Sharding*

I vantaggi introdotti dal processo di Sharding sono molto influenzati dal tipo di query che viene effettuata al database, infatti da come si può vedere nel caso di test il vantaggio è davvero minimo sulle fasi di IXSCAN e FETCH in quanto filtrando i risultati per un solo provider la query viene direzionata ad un solo Shard (quello contenente tutti i risultati relativi al provider scelto) introducendo il solo vantaggio di aver la metà dei documenti da filtrare. Nel caso però una query sia estesa a più provider si ha l'ulteriore vantaggio di dividere la query sui due Shard, in tal caso le fasi di IXSCAN, FETCH e PROJECTION verrebbero eseguite in parallelo, quindi nel caso i due Shard siano bilanciati (i dati siano spartiti in quantità simili sui due Shard) si avrebbe effettivamente una riduzione del tempo uguale alla metà.

Ultima ottimizzazione è stata l'introduzione delle media giornaliere tramite il precomputing.



*Figura 19 Risoluzione della query con precomputing*

Dai risultati appena mostrati risulta evidente come la soluzione del precomputing sia l'unica da considerarsi adatta per l'utilizzo in un'applicazione con interfaccia utente, ad ogni modo tutte le altre ottimizzazioni si sono rivelate utili proprio al fine di velocizzare la fase di precomputing. Il calcolo di ogni media giornaliera infatti avviene proprio tramite un'interrogazione al database che quindi trae vantaggio di tutte le ottimizzazioni viste prima.

## 6.3 Backend

Sul backend si concentra tutta la logica di business dell'applicazione per quanto riguarda la gestione dei dati, diventando l'unico punto di accesso verso il database. Per l'interazione verso MongoDB è stato utilizzato il modulo di Node Mongoose, che tramite un approccio simile a quello degli ORM<sup>1</sup> nei database tradizionali, ha reso l'intero processo più semplice.

Come anticipato, per il backend è prevista una sola istanza, eseguita su una delle tre macchine messe a disposizione nella rete interna di Unibo.

### 6.3.1 Consultazione dati

Per l'implementazione delle api di consultazione dei dati la maggiore attenzione è stata rivolta verso l'ottimizzazione della pipeline scritta con l'aiuto del plugin Mongoose.

In linea con le considerazioni fatte in fase di progettazione, al primo stadio della pipeline è stata sempre inserita l'operazione di match, cioè di filtro che riduceva il più possibile il numero di documenti su cui gli stati successivi della pipeline dovevano poi andare a lavorare, inoltre in seconda battuta viene effettuata una riduzione dei campi considerando solamente quelli necessari. La riduzione del numero di informazioni a questo stadio è essenziale in maniera da rendere il più veloce possibile lo stadio di raggruppamento successivo, che risulterà essere molto più pesante ed oneroso.

Le chiamate che forniscono i risultati relativi alle zone specifiche prevedono anche l'utilizzo di un parametro *crossRegion*. Questo dato, che non è presente nel database, viene aggiunto durante la fase di aggregazione da Mongoose e per ogni test indica se si è effettuato tra due regioni diverse (*crossRegion* = 1) oppure è interno ad una regione (*crossRegion* = 0) e quindi avrà i campi *from\_zone* e *to\_zone* equivalenti.

Esempio di implementazione dell'end-point API per l'accesso alle medie dei test di banda di una zona:

---

<sup>1</sup> L' Object Relation Mapping è una tecnica di programmazione per favorire l'integrazione tra linguaggi di programmazione ad oggetti e database relazionali.



```

router.route('/bandwidths/query/avgOfZoneOfSelectedDate').get(async
(req, res, next) => {
    var start, end, provider, from_zone, to_zone;

    start = new Date(req.query.start + "T00:00:00-00:00"); //YYYY-MM-DD
    end = new Date(req.query.end + "T23:59:59-00:00");
    from_zone = req.query.from_zone;
    to_zone = req.query.to_zone;
    if(req.query.zone != undefined){
        from_zone = to_zone = req.query.zone;
    }
    provider = req.query.provider;
    Bandwidth.aggregate().match(
        {$and: [
            {timestamp:
                {$gte: start,
                 $lte: end}
            },
            {provider: provider},
            {$or: [
                {from_zone: from_zone},
                {to_zone: to_zone}
            ]}
        ]}
    )
    .group({
        _id: {
            provider: "$provider",
            from_zone: "$from_zone",
            to_zone: "$to_zone"
        },
        avg: {$avg: "$bandwidth"}
    })
    .addFields({
        crossRegion:
            {$abs: {$cmp: ["$_id.from_zone", "$_id.to_zone"]}},
    })
    .project({
        "_id": 0,
        "provider" : "$_id.provider",
        "from_zone": "$_id.from_zone",
        "to_zone": "$_id.to_zone",
        "avg": "$avg",
        "count": "$count",
        crossRegion: "$crossRegion"
    })
});

```



Figura 20 Esempio di utilizzo delle api

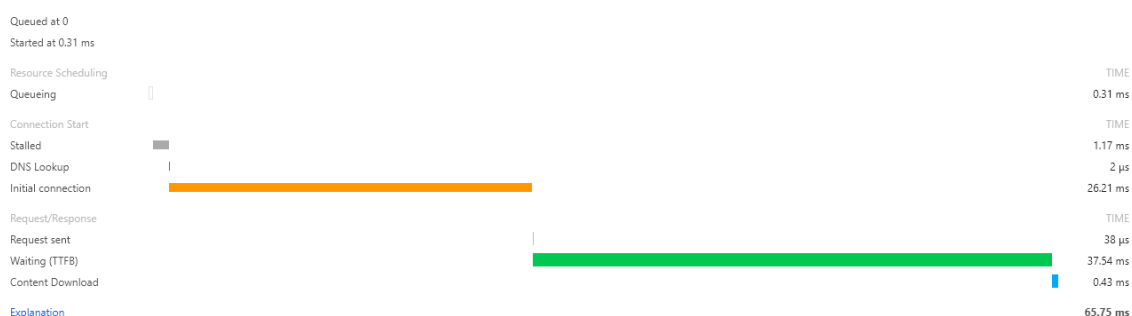


Figura 21 Tempi di risposta delle api

### 6.3.2 Caricamento risultati e precomputing

Il caricamento di ogni file di risultati avviene tramite una chiamata http Post verso il backend (tramite l'utilizzo del plugin multer di Node [10]), ad un indirizzo api specifico che distingue i test di ping da quelli di banda. In entrambi i casi il primo passo è quello di leggere e caricare i risultati dal file csv ricevuto al database MongoDB tramite l'utilizzo dell'apposito strumento mongoimport. Al termine di questo processo relativamente veloce i risultati di banda risulteranno già consultabili dal front-end. Per quanto riguarda i test di latenza invece verrà avviato il processo di calcolo della media giornaliera relativa.

Per ottenere informazioni relative ai test contenuti nel file che è appena stato caricato viene letta la prima linea (solamente la prima per velocizzare il processo, dando per garantito che tutte appartengano allo stesso tipo di test). Da questa linea vengono estratte le informazioni necessarie al calcolo giornaliero della media:

- Provider
- Zona di origine
- Zona di destinazione
- Data

Utilizzando questi dati, successivamente viene effettuata una query sul database al fine di calcolare la media della giornata nel test specificato.

Lettura dati dal file csv:

```
function readFirstLineCSV(inputFile) {
  var lineReader = readline.createInterface({
    input: fs.createReadStream(inputFile),
  });
  var lineCounter = 0;
  var wantedLine;
  lineReader.on('line', function (line) {
    if(lineCounter==1){
      wantedLine = line;
      lineReader.close();
    }
    lineCounter++;
  });
  lineReader.on('close', function() {
    var fields = wantedLine.split(',');
    var provider = fields[0];
    var from_zone = fields[1];
    var to_zone = fields[2];
    var date= new Date(fields[8]);
    precomputePings(provider, from_zone, to_zone, date);
  });
}
```

Calcolo media giornaliera:

```
async function precomputePings(provider, from_zone, to_zone, date) {

  var start = new Date();
  var end= new Date();
  start.setDate(date.getDate());
  start.setHours(0,0,0,0);
  end.setDate(date.getDate());
  end.setHours(23,59,59,999);
  var avg = await Ping.aggregate()
    .match({
      $and: [
        {provider: provider},
```

```

        {from_zone: from_zone},
        {to_zone: to_zone},
        {timestamp: {
            $gte: start,
            $lte: end
        }}
    ]
})
.group({
    _id: {
        provider: "$provider",
        from_zone: "$from_zone",
        to_zone: "$to_zone",
        day: {$dateToString: {format: "%Y-%m-%d", date:
"$timestamp"}}
    },
    avg: {$avg: "$time"},
    count: {$sum: 1}
})
return await updateDayAvg(avg[0]);
}

```

Aggiunta media giornaliera al database:

```

async function updateDayAvg(avg) {
    avg._id.day = new Date(avg._id.day+"T00:00:00");
    var pingDayAvg;
    var query = await PingDayAvg.find({
        provider: avg._id.provider,
        from_zone: avg._id.from_zone,
        to_zone: avg._id.to_zone,
        day: avg._id.day
    });
    if(query && query.length>0)
    {
        pingDayAvg = query[0];
        pingDayAvg.avg = avg.avg;
        pingDayAvg.count =avg.count;
    }
    else{
        pingDayAvg = new PingDayAvg({
            provider : avg._id.provider,
            from_zone: avg._id.from_zone,
            to_zone : avg._id.to_zone,
            day: avg._id.day,
            avg: avg.avg,
            count: avg.count
        });
    }

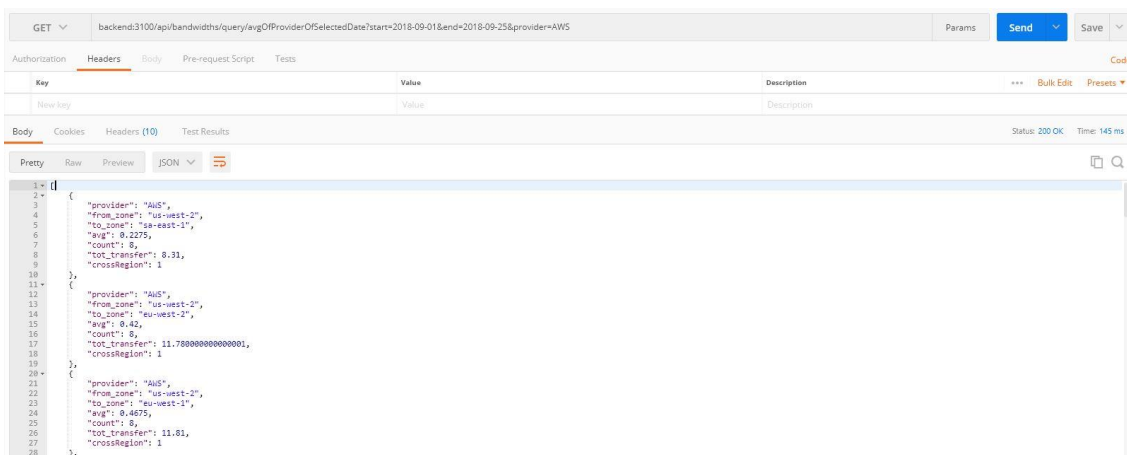
    const res = await pingDayAvg.save();
    return res;
}

```

Ricordando le considerazioni fatte nella fase di progettazione, risulta che una interrogazione di questo tipo su una tale quantità di dati può richiedere fino ad alcuni minuti per la risoluzione. Questo era evidente un problema nel caso la query mettesse l'utente in attesa, ma per quanto riguarda la fase di precomputing un tempo di attesa, anche di qualche minuto porta semplicemente a un minimo ritardo prima che i risultati del giorno precedente siano consultabili dall'applicazione web.

### 6.3.3 Analisi delle prestazioni

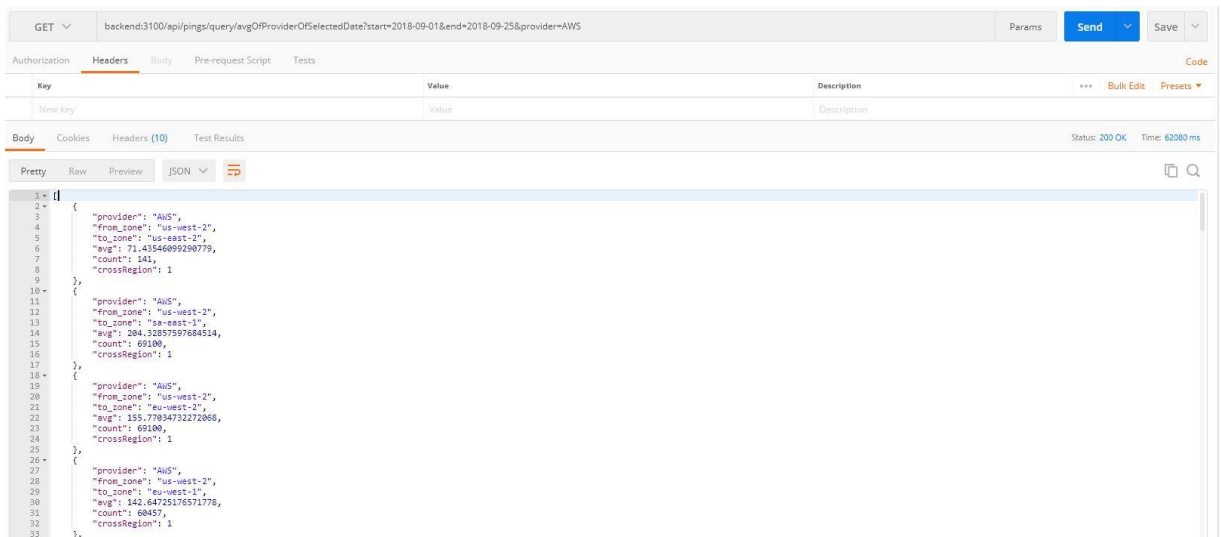
L'aspetto più critico nel sistema si è rivelato essere l'ottimizzazione dei tempi di risposta delle chiamate verso il backend per ottenere i risultati relativi ai test. Problema che è stato facilmente risolvibile per quanto riguarda i test di banda, che per loro natura vengono eseguiti più di rado, generando una quantità di risultati ampiamente più piccola dei test di ping.



*Figura 22 Chiamata backend relativa ai test di banda*

Mentre l'utilizzo di uno qualsiasi dei punti di accesso del backend che lavorava sui risultati dei test di ping, già dopo qualche settimana di utilizzo,

richiedeva di analizzare milioni di documenti, portando a tempi di risposta superiori al minuto.



*Figura 23 Chiamata al backend prima delle ottimizzazioni a MongoDB*

L'unico fattore che causava tale ritardo era ovviamente l'interrogazione verso il database che veniva effettuata dal backend, quindi dopo le ottimizzazioni viste nel capitolo di implementazione, il tempo di risposta si è drasticamente ridotto a qualche centinaio di millisecondi ed è soprattutto diventato molto più indipendente dall'intervallo di date sul quale si lavora, in quanto prima aggiungere un giorno all'intervallo del quale si desiderava effettuare un'analisi poteva significare dover analizzare migliaia di risultati in più, ora significava aggiungerne solamente uno.

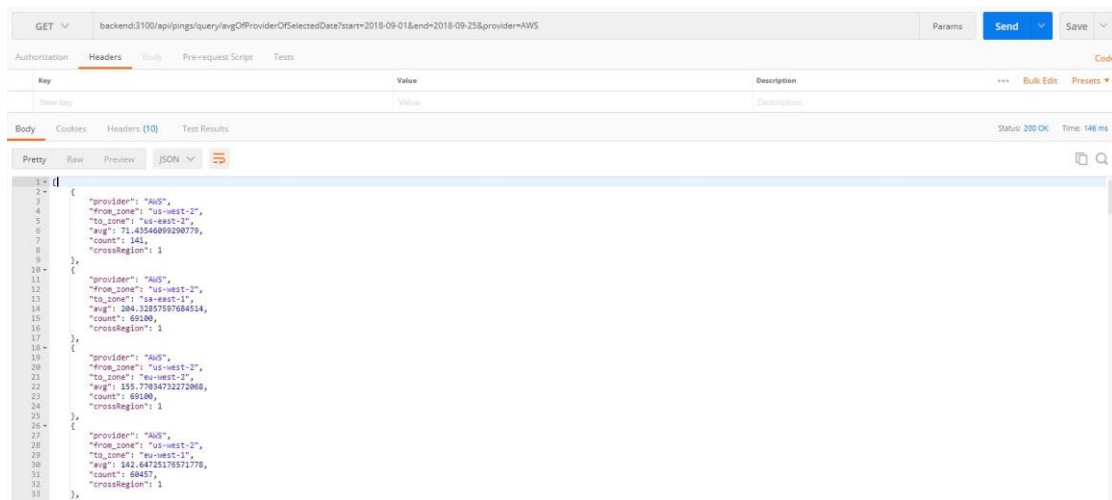


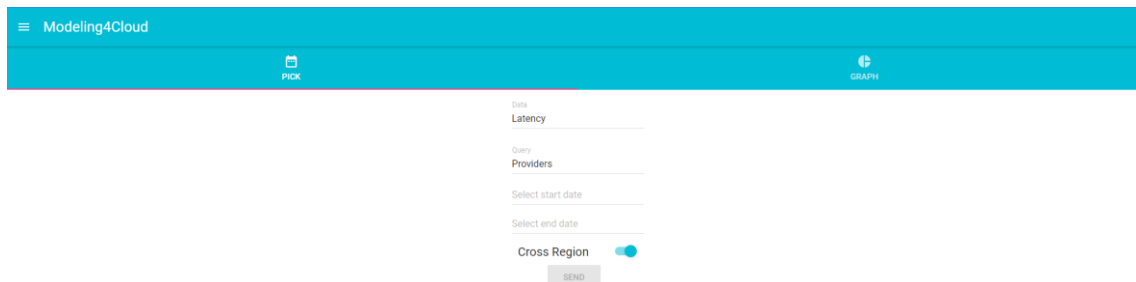
Figura 24 Chiamata al backend dopo le ottimizzazioni a MongoDB

Di fatto però, l'introduzione di questo calcolo delle medie giornaliere, nonostante abbia reso pressoché istantanea la risposta del backend al frontend, ha spostato la necessità di effettuare questi pesanti calcoli al momento dell'aggiunta dei risultati alla base di dati (quando ogni macchina di probing invia dei risultati al backend). In base alle ottimizzazioni fatte (aggiunta di indici, ottimizzazioni delle pipeline di aggregazione e sharding) per ogni file che una macchina invia al backend, il calcolo della media richiede circa 2 minuti. Il vantaggio è che in questo caso non c'è nessun utente in attesa del risultato, in quanto anche la macchina virtuale stessa che ha fatto l'upload del file viene liberata non appena il caricamento è terminato. Lo sviluppo del backend tramite Node, ed il suo modello di esecuzione, ha permesso che nonostante le intensive operazioni di accesso ai dati (tramite *mongoimport* per l'aggiunta dei nuovi dati al database, e le interrogazioni tramite *mongoose* per il calcolo delle medie giornaliere) il sistema continui ad essere reattivo indipendentemente dalla quantità di dati che si ricevono.

## 6.4 Interfaccia Web

La applicazione Web, sviluppata con React, mira ad essere il più semplice ed intuitiva possibile nello svolgere il compito di consultazione dei risultati. Si tratta di una One Page Application, infatti non è necessaria alcuna navigazione tra pagine e tutti i grafici possono essere consultati dalla stessa ed unica interfaccia.

La pagina principale presenta innanzitutto la scelta da due menu a tendina della tipologia di interrogazione da fare, cioè una scelta per il tipo di dati da ricercare (latenza o banda) e il tipo confronto che si desidera fare (tra provider, tra le zone di un provider, tra una specifica zona e le altre). Seguiti dai campi di ricerca relativi alla query scelta.



The screenshot displays the Modeling4Cloud web interface. At the top, there is a teal header bar with a hamburger menu icon and the text 'Modeling4Cloud'. Below the header, there are two icons: 'PICK' (represented by a folder icon) and 'GRAPH' (represented by a bar chart icon). The main content area is white and contains several search filters: a 'Data' dropdown menu set to 'Latency', a 'Query' dropdown menu set to 'Providers', a 'Select start date' input field, a 'Select end date' input field, and a 'Cross Region' toggle switch which is currently turned on (indicated by a blue circle). At the bottom of the filter section, there is a grey 'SEND' button.

*Figura 25 Interfaccia web*

Una volta effettuata la ricerca e ricevuti i risultati, sarà possibile consultare il grafico, che in base alla ricerca effettuata mostrerà i risultati specifici in un grafico a barre.



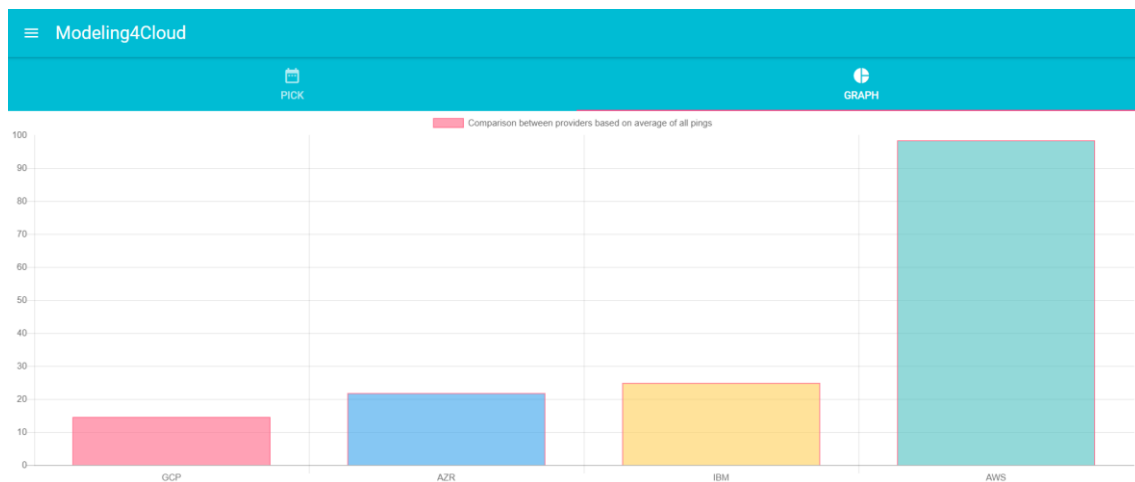


Figura 26 Confronto tra i risultati di ping di vari provider

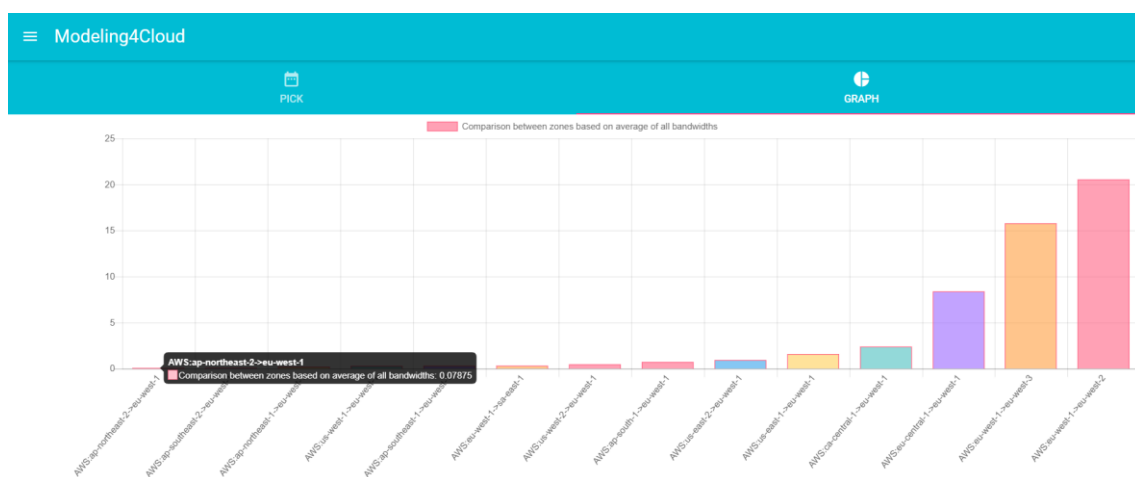


Figura 27 Confronto tra i risultati di banda di una zona

# Capitolo 7 Analisi dei risultati raccolti

Giunti al termine della realizzazione del progetto, è interessante fornire una piccola panoramica dei risultati che ha raccolto.

L'esecuzione dei test sui vari provider è stata resa possibile grazie all'utilizzo di credenziali per i vari fornitori cloud fornite e finanziate dall'azienda Imola Informatica. Questo ha ovviamente portato ad un'attenta analisi dei costi, ed in base alla analisi delle risorse che è stata precedentemente esposta, ad un inevitabile riduzione della scala dei test rispetto alla totalità dei provider previsti.

Tutti i test di banda di iperf3, sono stati configurati per eseguire utilizzando una sola connessione parallela, della durata di un secondo.

## 7.1 Confronto tra provider

Durante una prima fase di sviluppo e prova del progetto il sistema ha monitorato il collegamento sui 4 provider fra due regioni per ognuno di essi:

- AWS: Tra Londra (Zona: eu-west-2c) e Parigi (Zona: eu-west-3c)
- GCP: Tra Belgio (Zona: eu-west1-b) e Londra (Zona: eu-west2-a)
- AZURE: Tra Europa occidentale (Zona: eu-occ) e Francia centrale (Zona: fr-cent)
- IBM: Tra Milano (Zona: MIL01) e Parigi (Zona: PAR01)

Nel periodo dal 27 luglio 2018 al 10 settembre 2018.

Grazie ai test raccolti in questo periodo è stato possibile effettuare un confronto tra i vari provider, specialmente tra le istanze di livello più economico offerte sulle varie piattaforme. Un fattore da tenere in considerazione riguardo questi risultati sono sicuramente le limitazioni che l'utilizzo di tali istanze comporta rispetto ai risultati che si otterrebbero con l'utilizzo di soluzioni di grado più alto. Mentre per le risorse computazionali e di storage dati i provider forniscono informazioni

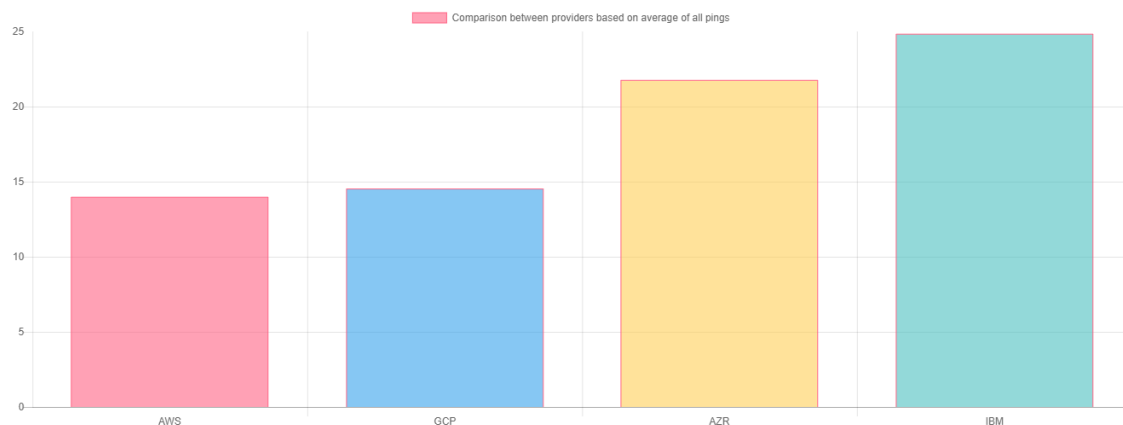
abbastanza precise riguardo le prestazioni in relazione al tipo di istanza scelta, per quanto riguarda la rete non esisto linee guida precise e limitazioni fisse, ma principalmente quella che viene fornita è una indicazione della precedenza e importanza che una macchina virtuale avrà nell'accesso alla rete rispetto alle altre. Questo fatto rende ancora più necessari tutti i test per il quale il sistema è stato costruito, fornendo una risposta ben precisa riguardo alle domande sulle prestazioni a cui nemmeno i cloud provider stessi sanno fornire risposta. Inoltre tutti i test sono stati eseguiti senza apportare nessun tipo di configurazione o ottimizzazione di rete alle macchine virtuali, utilizzando le impostazioni “di fabbrica”, bisogna comunque tenere in considerazione che una modifica (“*tuning*”) di alcuni parametri (come ad esempio MTU e TCP window) potrebbe portare a risultati diversi.

Costi delle istanze:

- Amazon Web Services: 5,42\$ mensili
- Google Cloud Platform: 7\$ mensili
- Microsoft Azure: 13,50\$ mensili
- IBM: 27\$ mensili

Latenza:

- Amazon Web Services: 13,99 ms
- Google Cloud Platform: 14,54 ms
- Microsoft Azure: 21,77 ms
- IBM: 24,84 ms



*Figura 28 Confronto della latenza tra i 4 provider (AWS, GCP, Azure, IBM)*

Nel caso del test della latenza, la distanza geografica tra i datacenter che partecipano è sicuramente il fattore più condizionante, ad ogni modo la scelta delle aree per ogni provider è stata fatta cercando di variare il fattore geografico il meno possibile da un provider all'altro in maniera da fornire un confronto equo.

Banda:

- Amazon Web Services: 278,4 MB/s
- Google Cloud Platform: 886,54 MB/s
- Microsoft Azure: 973,14 MB/s
- IBM: 124,12 MB/s



*Figura 29 Confronto della banda tra i 4 provider (AWS, GCP, Azure, IBM)*

In base ai dati raccolti, in entrambi i casi la soluzione meno performante è proprio quella di IBM, che oltre ad avere la latenza più alta e la banda più bassa nel caso considerato è anche la più costosa rispetto alle concorrenti. La soluzione più economica, Amazon, offre il tempo di latenza migliore, ad ogni modo soffre una velocità di banda abbastanza più bassa rispetto a Google e Microsoft. Il miglior rapporto qualità/prezzo è quello offerto da Google, che con un costo leggermente superiore a quello chiesto da Amazon, fornisce prestazioni pressoché simili ai più performanti sia in latenza che in velocità.

## 7.2 Monitoraggio completo su Amazon Web Services

Il monitoraggio completo, che per questioni di costo si è scelto di eseguire solamente sul provider Amazon, ha avuto inizio il 18 settembre 2018 ed è ad oggi (28 settembre 2018) ancora in corso.

La quantità di dati generati, pur limitandosi ad un solo provider è piuttosto notevole, portando il database a crescere di circa 1 Gigabyte di dati ogni 5 giorni, rendendo il monitoraggio anche un ottimo sistema di testing per il progetto e per la sua scalabilità.

Grazie ai dati raccolti è stato possibile confermare come le performance di rete tra le varie zone siano fortemente legate alla distanza che separa due zone geografiche anche se appartenenti allo stesso provider, soprattutto nel caso di continenti differenti.

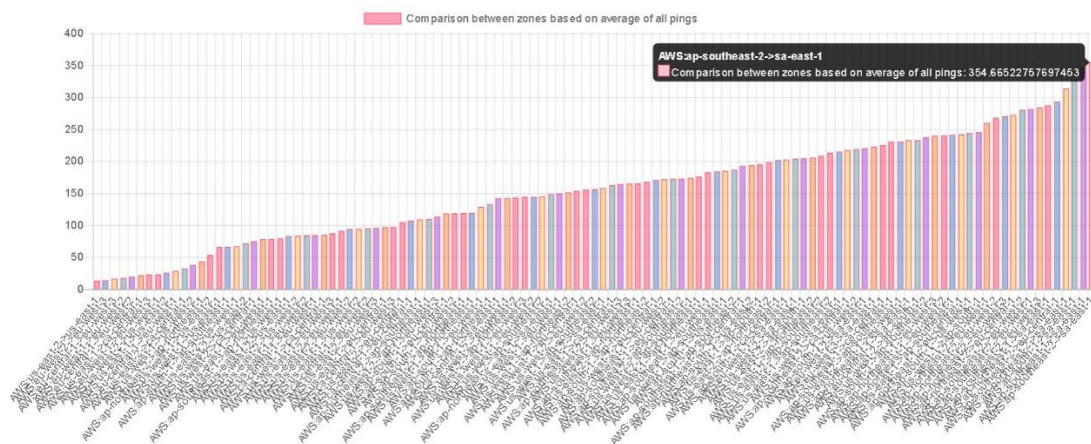


Figura 30 Confronto latenza tra le varie regioni del provider Amazon

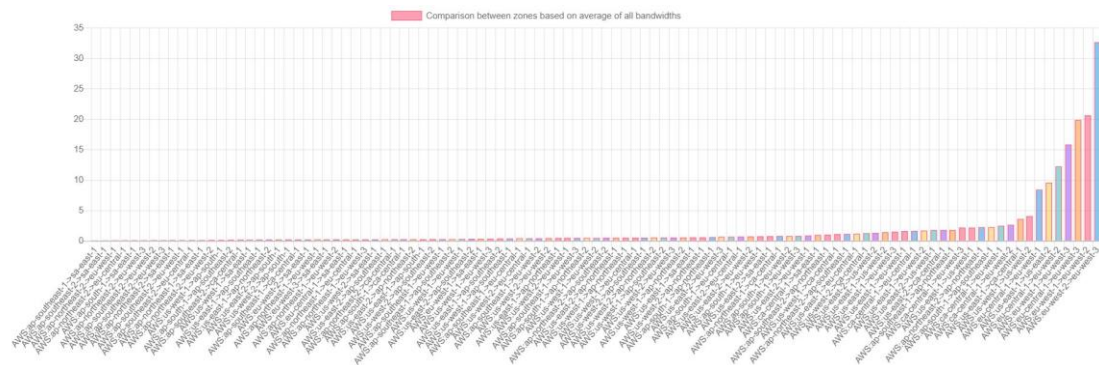


Figura 31 Confronto velocità di trasferimento tra le varie regioni del provider Amazon

Dai test risulta che i collegamenti più lenti, e dove si riscontra più ritardo, sono quelli tra Europa ed Asia, mentre il continente nel quale si riscontrano le performance in media più elevate è l'Europa, grazie all'elevato numero di datacenter disponibili in rapporto all'area coperta.

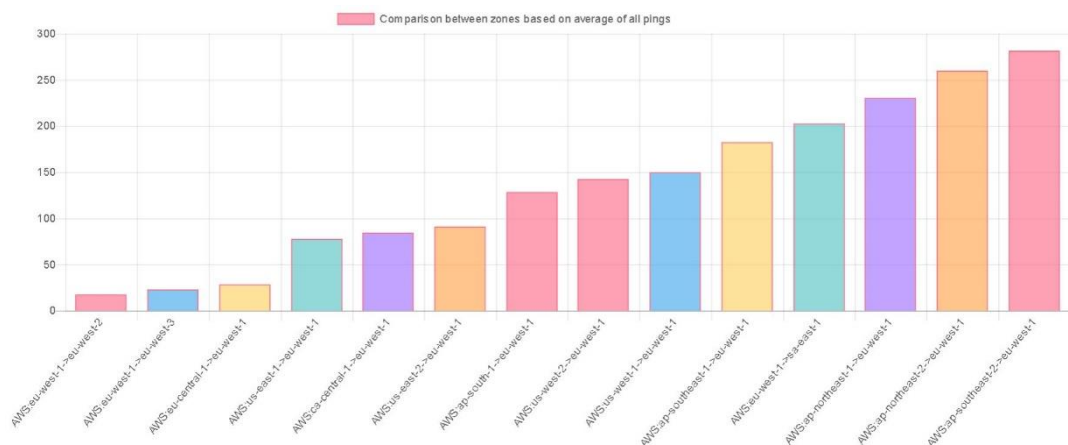


Figura 32 Confronto latenza dalla zona Amazon eu-west-1 (Irlanda) alle altre zone





# Capitolo 8 Conclusioni

Lo sviluppo del progetto, è riuscito a seguire le direttive che erano state preventivamente proposte, portando alla creazione di un sistema robusto, stabile e scalabile, dove tutte le funzionalità necessarie per un accurato confronto tra i vari fornitori Cloud sono correttamente implementate ed intuitive nell'utilizzo.

Durante tutto il processo di sviluppo l'ottica principale è sempre stata quella di fornire un sistema che fosse completamente indipendente dalla piattaforma sulla quale venisse distribuito e soprattutto dalla quantità di dati che dovesse gestire. Proprio questo aspetto, ha anche finito per rivelarsi quello più critico, portando alle problematiche principali che sono state affrontate durante la fase di implementazione. L'esecuzione del periodo di collaudo, specialmente quello su scala globale di Amazon Web Services, reso possibile grazie alla collaborazione dell'azienda Imola-Informatica, ha permesso di dimostrare a che tipo di carico il sistema può essere sottoposto senza subire alcun degradamento delle proprie funzionalità e proprietà. L'approccio di questi problemi con l'utilizzo di tecnologie che si possono considerare ancora nella loro infanzia e completamente nuove per il sottoscritto ha reso il progetto una sfida implementativa di elevato livello, per questo, vedere un sistema completo e all'altezza delle aspettative iniziali risulta un ottimo traguardo. A questo punto, ci si auspica che il percorso del progetto non sia terminato, effettuando un monitoraggio simile a quello che è stato svolto per Amazon anche sugli altri fornitori.

Nonostante il monitoraggio effettuato abbia fornito una dettagliata panoramica dell'infrastruttura analizzata, non mancano possibili rifiniture riservate a sviluppi futuri. La parte più debole dell'intero sistema è sicuramente quella di presentazione dei dati, attualmente molto semplice e limitata ad un minimo numero di viste, che potrebbe essere ampliata inserendo altre tipologie di grafici, fornendo raggruppamenti differenti dei dati (ad esempio in base ai continenti geografici o agli orari della giornata), oppure fornendo una panoramica del cambiamento di un particolare parametro prestazionale nel tempo. Per quanto riguarda la parte di

persistenza invece, l'implementazione e la parallelizzazione dei vari elementi del cluster sono sempre stati mirati al miglioramento delle performance soprattutto per quanto riguarda i tempi di risposta, ad ogni modo, semplicemente aggiungendo ulteriori elementi ai Replica Set già esistenti si riuscirebbe a introdurre anche una replicazione dei dati sui vari nodi, introducendo anche una elevata capacità di resistenza ai guasti del sistema, un aspetto che è stato fino ad ora trascurato.

# Riferimenti

- [1] «amazon.com,» [Online]. Available: <https://aws.amazon.com/it/about-aws/global-infrastructure/>.
- [2] «Atomia,» [Online]. Available: <https://www.atomia.com/home/>.
- [3] «Mokabyte,» [Online]. Available: <http://www.mokabyte.it/2012/09/soluzioniconcloud-1/>.
- [4] «awsspeedtest.xvf.dk,» [Online]. Available: <http://awsspeedtest.xvf.dk>.
- [5] «azurespeed.com,» [Online]. Available: <http://azurespeed.com>.
- [6] «mapreduce.it,» [Online]. Available: <http://www.mapreduce.it/cosa/>.
- [7] «Imola Informatica,» [Online]. Available: <https://www.imolainformatica.it/home/>.
- [8] «Linux Man Page,» [Online]. Available: <https://linux.die.net/man/1/nohup>.
- [9] «Linuxaria,» [Online]. Available: <https://linuxaria.com/howto/manage-planned-tasks-on-linux-with-the-command-at?lang=it>.
- [10] «Express upload with multer,» [Online]. Available: <https://scotch.io/tutorials/express-file-uploads-with-multer>.
- [11] «CableMap,» [Online]. Available: <https://www.cablemap.info/>.
- [12] «manuscavelli.it,» [Online]. Available: <https://www.manuscavelli.it/hping3/>.
- [13] «iPerf,» [Online]. Available: <https://iperf.fr/iperf-doc.php>.

