Resvag: una Fog application per la generazione di itinerari turistici

SDCC 2020 - Progetto A1

Pasquale Caporaso
studente di ingegneria informatica
Università degli Studi di Roma "Tor Vergata"
Roma, Italia
caporasopasquale97@gmail.com

Ezio Emanuele Ditella
studente di ingegneria informatica
Università degli Studi di Roma "Tor Vergata"
Roma, Italia
ezioemanuele.ditella@gmail.com

I. Descrizione del sistema

Il sistema che noi presentiamo vuole fornire ad un turista che sta visitando Roma un itinerario dei monumenti più popolari nelle sue vicinanze tenendo conto del suo tempo a disposizione per visitare la città. L'obiettivo di questo progetto è di garantire una bassa latenza all'utente nonostante il carico computazionale necessario per la generazione dell'itinerario, per raggiungere questo scopo abbiamo progettato le seguenti componenti:

A. Ariadne

Ariadne è l'applicazione disponibile per dispositivi android, che è usata dal turista per generare richieste. L'applicazione offre come funzionalità principali quelle di selezionare un punto di partenza diverso dalla posizione geografica, salvare risultati delle richieste effettuate, salvare itinerari preferiti.

B. Pythia

Pythia è un cluster di server bootstrap che mantiene le informazioni sui nodi fog (Hermes) attivi e funzionanti, che viene interpellato dagli utenti di Ariadne per potersi far restituire la lista dei fog più vicini alla loro posizione geografica.

C. Ulixes

Ulixes è il centro computazionale del nostro progetto, il suo compito è quello di gestire richieste provenienti sia da Hermes, sia da utenti di Ariadne e rispondere con l'itinerario corretto. Ulixes si trova nel Cloud in modo da poter essere elastico e meglio gestire un carico a volte molto pesante.

D. Hermes

Hermes è la componente FOG del nostro sistema, l'obiettivo di questa parte dell'infrastruttura è quello di ricevere richieste dagli utenti di Ariadne nelle sue vicinanze e, se è in grado, di rispondere con l'itinerario corretto o, se non può, di spedirle alle macchine sul Cloud (Ulixes) facendo da ponte per la comunicazione. Hermes è un server che va posizionato fuori dal Cloud, in zone con un numero elevato di utenti e monumenti per assicurare una bassa latenza ed elevata efficacia.

Identify applicable funding agency here. If none, delete this.

II. Implementazione

L'implementazione di Resvag è stato un processo particolarmente complesso, abbiamo dovuto usare, infatti, diversi linguaggi per le componenti e abbiamo fatto uso di numerose API gratuite. Inoltre la raccolta delle informazioni riguardanti i vari monumenti è stato un processo lungo e costoso per via della scarsità di sorgenti di dati e dell'elevato costo delle Google maps API. Una rappresentazione grafica dell'intera infrastruttura si trova in figura 1.

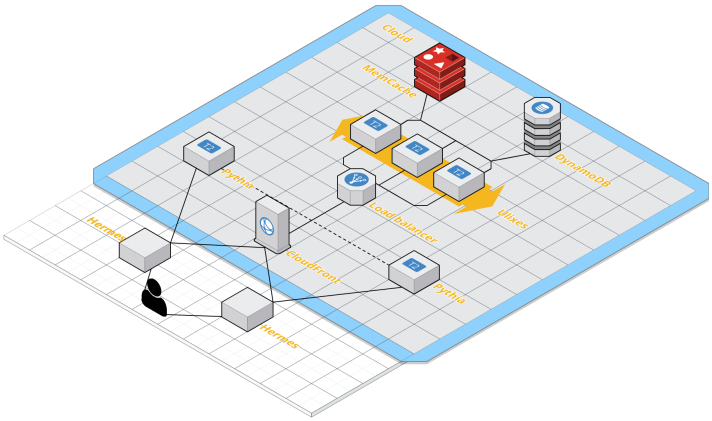

Fig. 1. Uno schema dell'infrastruttura del sistema

Nei prossimi paragrafi procediamo con una descrizione dettaglaita dell'implemntazione di ogni componente del sistema

A. Ariadne

Ariadne è una semplice applicazione Android, in essa sono state inserite diverse funzionalità, in particolare:

- un interfaccia di ricerca, una semplice GUI in cui l'utente può selezionare la posizione di partenza (di default viene scelta quella attuale) il tempo di visita e il mezzo di transporto utilizzato
- lista degli ultimi itinerari cercati.
- lista degli itinerari salvati

L'itinerario mostrato all'utente è fatto da una serie di riquadri, ciscuno contenente l'immagine e nome del monumento dell'itinerario e orario stimato dell'arrivo. All'utente viene data la possibilità di avere informazioni sulla descrizione di ciscun monumento semplicemente cliccando su questi riquadri.
Gli itinerari sono localmente salvati in un database *SQLite*.
La comunicazione con i server avviene tramite web API per Hermes/Ulixes e tramite socket TCP per Pythia con richieste in formato json. In particolare il processo di ricerca degli itinerari procede nel seguente modo:

1) Viene contattato Pythia per ottenere una lista dei nodi Hermes disponibili in ordina di vicinanza all'utente. Queste informazioni sono salvate localmente nel dispositivo e invalidate dopo 3 ore;
2) Vengono inviate richieste per itinerari ai nodi della lista in ordine finchè uno di essi non risponde, se nessuno di essi è attivo la richiesta viene mandata a Ulixes;
3) Le informazioni ricevute vengono mostrate all'utente, se nessun server è raggiungibile viene mostrato un messaggio di errore.

B. Pythia

Pythia è stato implementato in python, e dispone delle seguenti funzionalità

- Memorizzare le informazioni dei nodi fog che richiedono di registrarsi
- Monitorare lo stato dei nodi fog registrati
- Inviare la lista dei nodi fog attivi in ordine di vicinanza geografica rispetto al richiedente
- Effettuare *flooding* della lista di nodi accettati verso tutte le istanze Pythia

Pythia accetta richieste di registrazione in formato json, in cui viene specificato: l'ip e coordinate geografiche del richiedente, la porta alla quale saranno inviati gli health check (beat), sulla quale i fog dovranno mettersi in ascolto subito dopo la registrazione.
Il monitoraggio dello stato dei nodi è stato fatto implementando il meccanismo dell'*heartbeat*. Tale meccanismo consiste nell'invio periodico di un segnale, o beat, verso i nodi fog. Un nodo è segnato come inattivo se non ha risposto a più di 3 beat consecutivi.
Il calcolo della distanza geografica tra i nodi fog e un richiedente della lista di nodi è fatto usando la formula di *Haversine*.
Per aumentare la tolleranza ad i guasti, e evitare *single-point-of-failure*, l'infrastruttura è dotata di più istanze di Pythia. Nel momento in cui un'istanza di server Pythia è lanciata, il suo indirizzo IP viene associato ad un record DNS di tipo A con nome *pythiaX-resvag.crabdance.com*[1], con *X* pari al numero dell'istanza in esecuzione.
Sono poi creati dei record DNS di tipo ALIAS in cui tali nomi di dominio vengono associati al dominio *pythia-resvag.cloudns.cl*.[2]A questo punto un istanza Pythia andrà ad effettuare il flooding dei nodi attivi a tutti gli IP, diverso dal suo, restituiti dalla query DNS a *pythia-resvag.cloudns.cl*.
Lo stato dei nodi ricevuti con il flooding è eventualmente aggiornato dalle stesse istanze Pythia che li hanno inoltrati.
Il modulo dell'heartbeating è stato progettato in modo tale che nel caso in cui le istanze Pythia dovessero crashare, gli Hermes registrati sono in grado di accorgersene e cercheranno di riconnettersi inviando una nuova richiesta di registrazione a intervalli regolari, per evitare di sovraccaricare Pythia in un eventuale fase di reboot.
C'è da dire che nel caso in cui un istanza Pythia abbia floddato la sua lista di nodi, per poi crashare, la lista di nodi ricevuta diventerebbe inaffidabile, non sapendo se qualche fog appartenente a tale lista sia attivo o meno. Tale scelta è stata effettuata per alleggerire la complessità di Pythia e per mantenere elevato lo scambio di informazioni. Nel caso peggiore sarà un utente di Ariadne ad accorgersi che un nodo fog è inattivo.

C. Ulixes

Ulixes è stato implementato come un server multi-thread python in grado di gestire richieste http.

All'accensione Ulixes recupera da DynamoDB le informazioni sui monumenti e sulle loro distanze e le organizza in due grafi di adiacenza, uno che gestise le richieste con mezzo di trasporto "a piedi", l'altro quelle per il mezzo di trasporto "macchina". Vengono recuparate dal Database anche tutte le informazioni per ogni singolo monumento, in particolare la sua popolarità che verrà utilizzata dall'algoritmo per trovare gli itinerari migliori.

All'arrivo di una richiesta il processo che porta al ritrovamento degli itinerari corretti è il seguente:

1) Vengono analizzati i parametri della richiesta, se non sono presenti tempo di visita, posizione di partenza e/o mezzo di transporto la richiesta viene ignorata e viene restituito un errore HTTP 404;
2) Se la richiesta è corretta viene controllato se è memorizzata in cache una richiesta simile[3], se assente si verifica la sua presenza in Memcached.
3) Se nessuna richiesta simile è stata trovata viene calcolata la distanza tra l'utente è il monumento più vicino, questo verrà utilizzato come punto di partenza per l'itinerario, questa distanza viene calcolata prendendo i 5 nodi più vicini in linea d'aria e poi utilizzando le Google Maps API per trovare il più vicino tra essi;
4) Utilizzando il grafo corrispondente al mezzo di transporto scelto e utilizzando il nodo di partenza dell'utente, viene usata una visita in profondità per trovare gli itinerari che contengono i monumenti più popolari;
5) La lista dei monumenti viene trasformata in un json aggiungendo tutte le informazioni disponibili contenute sul Database ed eventuali altre informazioni da 'calcolare' al volo (immagini dei monumenti). La richiesta viene

[1]Questa operazione è fatta tramite il servizio gratuito di DNS hosting *FreeDNS*

[2]Usando il servizio gratuito CloudDNS

[3]Due richieste sono simili se fatte usando lo stesso tempo, mezzo di trasporto e e se gli utenti che hanno effetuato la richiesta erano 'vicini'

associata ad un codice univoco e salvata in cache locale e su Memcached, per poi essere spedita all'utente che l'ha richiesta

Ulixes si trova sul Cloud ed è implementato come un gruppo di autoscaling, il collegamento con l'esterno avviene tramite un load balancer.

D. Hermes

L'implementazione di Hermes è molto simile a quelli di Ulixes, in quanto eseguono principalmente le stesse funzionalità.

La differenza è che il grafo di un nodo Hermes contiene solo un sottoinsieme dei nodi totali e quindi non è in grado di gestire tutte le richieste che riceve. Inoltre viene aggiunto al grafico un nodo speciale chiamato "Nodo all'infinito" che viene utilizzato per capire quale richieste Hermes è in grado di gestire.

Definiamo il "nodo all'infinito" come un nodo con popolarità zero che ha come distanza da tutti gli altri nodi *la minore distanza tra quel nodo e un qualsiasi nodo che non compare nel grafo di Hermes ma che esiste in quello completo.*

Un esempio di creazione del grafo di Hermes è rappresentato in figura 2, a sinistra abbiamo il grafo completo presente in Ulixes, a destra il grafo "contratto" presente in un Hermes, il nodo all'infinito è contrassegnato con la lettera I.

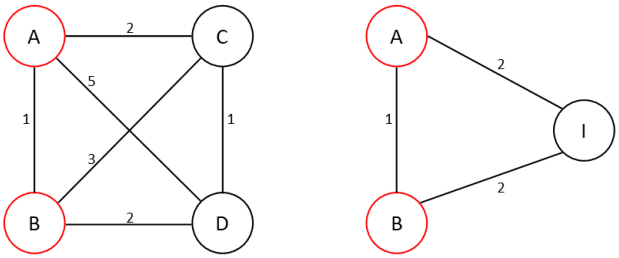

Fig. 2. Il passaggio da grafo completo a grafo di un nodo Hermes

Hermes, all'accensione, dopo aver caricato il suo grafo ridotto, utilizzerà l'algoritmo di Bellman-Ford per trovare la distanza minima tra il nodo all'infinito e tutti gli altri nodi del grafo. Facendo così il nodo sarà in grado di riconoscere le richieste che in grado di gestire nel seguente modo:

1) Viene estratto il tempo di visita dalla richiesta;
2) Viene trovato il nodo di partenza dell'utente utilizzando la stessa tecnica di Ulixes;
3) Si confronta la distanza dal nodo di partenza e il nodo all'infinito con il tempo di visita dell'utente (con il corretto mezzo di transorto), se l'utente è in grado di raggiungere il nodo all'infinito la richiesta non è gestibile dal FOG e viene mandata al cloud, altrimenti viene gestita in locale.

Visto che le distanze dei vari nodi sono salvate in un dizionario e visto che la ricerca del nodo iniziale dell'utente fa parte dell'algoritmo per gli irinerari possiamo dire che Hermes è in grado di capire se può gestire una richiesta in tempo costante, indipendentemente dal numero di nodi o dal tempo a disposizione dell'utente.
Per simulare quanto più possibile la vicinanza dei fog all'utente, le istanze di Hermes si trovano a Milano, nella regione italiana di AWS.

E. Servizi extra - CloudFront, MemCached

I due servizi cloud, Clodfront e Memcache, vengono utilizzati per effetuare caching delle richieste già calcolate, in modo da non appesantire gli Ulixes con ulteriori compiutazione che sono in genere molto pesanti.

Una delle ragioni per cui abbiamo usato il protocollo http per la comunicazione è stato per poter utilizzare CloudFront e il suo servizio di CDN. Questo servizio non fa altro che salvare le richieste già calcolare e, se arriva una richiesta uguale (ovvero un utente che chiede un itinerario da un posizione vicina e con lo stesso tempo di visita) non la inoltra a Ulixes ma gli ritorna la risposta che è stata salvata.

III. Deployment

Il deploy di Resvag è stato fatto usando il paradigma dell'*Infrastructure as a Code*. In particolare il processo di creazione ed inizializzazione dell'intera infrastruttura è stato automatizzato con l'ausilio di *ansible*, il quale a sua volta utilizza *amazon AWS cli*, con l'esecuzione dei seguenti passi

1) Creazione di gateway, VPC, sottoreti (in diverse zone di disponibilità) e routing table nelle regioni di amazon AWS di Francoforte e Milano
2) Creazione di security groups per ogni componente dell'infrastruttura e per il load balancer che parlerà con gli Ulixes, definendo chi e come (quale porta e protocollo) può contattare i vari componenti
3) Creadione di un cluster di Memcached, fatto da 3 nodi, localizzato a Francoforte
4) Creazione di 2 Pythia, una di Ulixes a Francoforte. Ogni istanza in sottoreti, e dunque zone di disponibilità, differenti
5) Creazione di 2 istanze di Hermes a Milano, sempre in zone di disponibilità differenti
6) Creazione del Load Balancer
7) Registrazione dei domini di Pythia, Load Balancer e Cluster di Memcached nel DNS (usando contemporaneamente le API di 3 servizi di hosting gratuiti: FreeDNS, NoIP, CloudDNS)
8) Configurazione di CloudFrount
9) Configurazione delle istanze appena create di Pythia, Hermes ed Ulisse
10) Creazione del grupo di auto-scaling, associando l'istanza appena configurata di Ulisse alla configurazione di lancio
11) Definizione di un autoscaling group con, inizialmente, 3 istanze di Ulisse

IV. Limitazioni del sistema

Il sistema che abbiamo realizzato è soggetto a numerose limitazioni, molte dovute al fatto che è stato creato utilizzando esclusivamente servizi gratutiti, altre dovute a limitazioni fisiche o algoritmiche.

In questa sezione descriviamo le più importanti e discutiamo se e come possono essere superate.

A. Potenza computazionale

Per rimanere nei limiti del nostro grant tutte le macchine ospitate dal Cloud sono delle "T2 Micro" con una potenza computazionale decisamente ridotta. Per questo motivo le prestazioni del nostro algoritmo sono deludenti e non riusciamo ad ottenere risposta per tempi di visita superiori alle 7 ore senza che scatti il timeout della nostra applicazione cellulare. Inoltre, anche per tempi di visita relativamente bassi, ci aspettiamo che i server non possano gestire un numero di utenti elevato.

B. Hermes sul Cloud

Il nodo Hermes dovrebbe essere un nodo FOG posizionato sul territorio, idealmente come un totem situato in zone con altra contrentrazione di monumenti, tuttavia, per mancanza di risorse questo non è stato possibile. Attualmente gli Hermes vengono anch'essi hostati sul Cloud, abbiamo cercato di utilizzare la regione di Milano ma la latenza rimane comunque alta.

C. API a pagamento

Per questo progetto abbiamo dovuto recupare da internet tante informazioni riguardanti i monumenti e ci siamo imbattuti in numerosi problema dovuti alla mancanza di budget. Innanzitutto, il numero di monumenti che abbiamo potuto utilizzare alla fine è 100, questo limite è dovuto alle Google Maps API, infatti le richieste necessarie a trovare tutte le distanze, a piedi e con la macchina, tra questi monumenti ci sono costate all'incirca 200 euro, è stato quindi impossibile aggiungere altri monumenti, i restanti 100 euro del grant di Google, infatti, vengono utilizzati attivamente dal sistema per trovare il nodo di partenza dell'utente.

Un'altro problema è stato quello di trovare una API che ci fornisse delle foto dei monumenti scelti e una loro descrizione in italiano/inglese, dopo lunghe ricerche ne abbiamo trovata una gratuita di qualità, però, non eccezionale, infatti alcune descrizioni sono in francese e non tutte le foto rappresentano effettivamente i monumenti. Avendo avuto un budget a disposizioni questi problemi non si sarebbero presentati in quanto esistono numerose API a pagamento che offrono questi servizi.

D. Record DNS

Idealmente si avrebbe voluto un instrastruttura laddove non sia noto a priori il numero di istanze Pythia da creare, tuttavia, dovendo memorizzare dei record DNS di tipo A, ed essendo tale operazione *non* gratuita, ci si è dovuti limitare al massimo numero di record concessi da servizi gratuiti di hosting DNS.

V. Sviluppo

Lo sviluppo del sistema è avvenuto su diverse piattaforme e utilizzando diversi linguaggi, in particolare:

A. Android Studio

L'applicazione cellulare è stata sviluppata interamente in Android Studio, l'IDE di JetBrains offre un alto numero di funzionalità e facilitano lo sviluppo dell'applicativo. Nell'ottica del pattern di sviluppo BCE, i componenti software di Controllo ed Entità sono stati sviluppati in Java in quanto è l'unico linguaggio supportato, mentre i componenti della bounndary in XML.

B. PyCharm

Un IDE di JetBrains, che ha facilitato la programmazione in python e processo di continous integration grazie all'integrazione con GitHub.

C. Sublime Text

Sublime Text è un editor di testo con diverse feature che lo rendono molto adatto alla programmazione, entrambi abbiamo usato questo programma per scrivere lo script in Ansible per la creazione dell'infrastruttura e uno di noi l'ha usato anche per la programmazione in Python.

D. Git e Github

Durante la totalità del progetto abbiamo usato delle git repository caricate su GitHub per collaborare durante la scrittura del codice.

VI. Testing

Ogni componente dell'infrastruttura è stato testato finchè non sono stati raggiunti tutti i criteri di adeguatezza di tipo funzionale.

Non è stato possibile testare in modo automatizzato Ariadne in quanto necessita dell'interazione con l'utente.

Prima di fare il deploy di Pythia su AWS, le sue funzionalità sono state testate facendo il deploy di un istanza su PC ed un altra su raspberry, anche in questo caso manualmente.

Il test di sistema è stato fatto d'apprima in una rete locale, simulando il comportamento di un utente tramite due dispositivi mobili android, e poi su AWS. Test non funzionali sono stati effettuati manualmente a causa della complessità dell'infrastruttura.

Per testare la latenza tra un utente e la nostra infrastruttura abbiamo creato un script per mandare richieste con tempo di visita crescente prima a Hermes e poi direttamente ad Ulisse. La nostra speranza era di indentificare un pattern che mostrasse la maggiore velocità di Hermes per le richieste di piccole dimensioni, tuttavia, visto che anche Hermes si trova anch'esso alla fine sul Cloud, non è possibile osservare nessun cambiamento significativo dei tempi di risposta. I risultati in forma grafica sono rappresentati in figura 3.

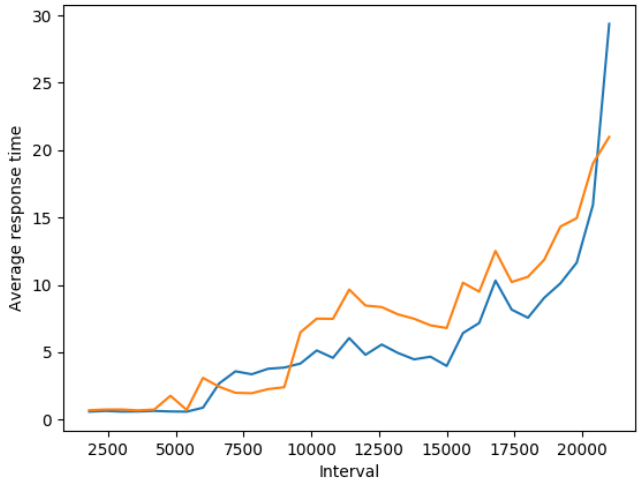

Fig. 3. Tempo medio di risposta, in rosso Ulixes, in blu Hermes