My Own Serverless Application: Nonno Care

Falvo Simone
Università degli Studi di Roma Tor Vergata
smvfal@gmail.com

Pietrangeli Aldo
Università degli Studi di Roma Tor Vergata
pietrangeli.aldo@gmail.com

***Abstract*—Lo scopo del progetto è stato quello di realizzare un'applicazione serverless sfruttando i servizi di Amazon AWS per processare ed acquisire dati relativi ad un bracciale che fornisce assistenza a persone affette da demenza senile. I dati inviati dai sensori fanno scaturire degli eventi che abbiamo gestito attraverso AWS Lambda, in particolare ogni funzione gestisce una tipologia diversa di evento. La logica del sistema è infatti organizzata in funzioni che rappresentano i flussi alternativi che un dato evento può produrre.**

Una di queste funzioni integra il servizio di machine learning AWS SageMaker con il quale è stato addestrato un modello in grado di classificare le attività registrate dal bracciale e distinguere le cadute da normali gesti quotidiani a cui corrispondono valori dei sensori di accelerazione e giroscopio piuttosto simili.

Oltre a questo, sempre tramite i relativi servizi di Amazon, il sistema realizza la persistenza dei dati acquisiti e consente di effettuare query su di essi.

L'applicazione è stata testata tramite l'impiego di un simulatore di eventi opportunamente sviluppato, in modo da produrre dati corrispondenti ad un carico variabile di 500, 1000 e 2000 utenti. I risultati sperimentali mostrano che il sistema mantiene una latenza costante pari in media a 930 millisecondi, mentre il throughput aumenta linearmente con il numero di utenti per il carico testato, ottenendo valori pari in media a rispettivamente 3.65, 7.20, 14.65 completamenti al secondo.

I. Introduzione

Il bracciale è dotato di sensori di movimento, di posizione, frequenza cardiaca e di un pulsante per la richiesta d'aiuto.

Il dispositivo invia dati su posizione e battiti periodicamente con una frequenza fissa (ogni 10 minuti), inoltre, se il bracciale registra un'accelerazione repentina, invia un batch di dati provenienti da giroscopio e accelerometro generando un evento aperiodico.

Ogni evento viene acquisito da un sistema a coda che lo memorizza per poi essere elaborato da una prima funzione Lambda che ne realizza la persistenza, lo pre-processa e genera uno o più nuovi eventi aventi topic specifici per i quali è necessario un ulteriore processamento per la verifica di un possibile incidente. Questi eventi vengono pubblicati in un sotto-sistema di tipo publish/subscribe a cui sono sottoscritte indirettamente le funzioni Lambda dedicate, che processeranno i relativi eventi.

Il termine "incidente" è rappresentativo di 4 possibili casistiche:

1) Fuoriuscita dalla zona circoscritta corrispondente alla security zone dell'anziano;
2) Il valore del battito cardiaco è al di fuori di un intervallo basato sul valor medio e sulla varianza dell'anziano stesso;
3) L'attività registrata è associabile ad una caduta;
4) È stato premuto il pulsante di S.O.S.

In caso di accertamento di questi incidenti, vengono intraprese le opportune azioni che consistono nella loro persistenza e nella notifica ad un utente registrato, quest'ultime realizzate da una specifica funzione lambda.

II. Architettura

L'architettura del sistema può essere esaminata analizzando separatamente le diverse fasi a cui i dati prodotti dal bracciale sono sottoposti:

A. **Acquisizione**: fase in cui i dati vengono immessi nel sistema;
B. **Processamento** e analisi: i dati acquisiti vengono processati ed interpretati dalle funzioni lambda;
C. **Persistenza**: i dati processati e interpretati vengono memorizzati in un database;
D. **Visualizzazione o Interrogazione**: i dati resi persistenti vengono interrogati e forniti all'utente che ne ha fatto richiesta.

I vari servizi che compongono il sistema sono gestiti tramite AWS CloudFormation che permette di usare un semplice file di configurazione per modellare tutta l'infrastruttura, inoltre semplifica e velocizza i processi di sviluppo, test e deployment dell'applicazione avendo un unico punto di controllo e offrendo semplici comandi per l'istanziazione e la modifica delle risorse.

In un unico file di configurazione è presente la descrizione di tutte le risorse del sistema e la loro composizione, ogni riferimento alle risorse è specificato in maniera implicita (i nomi e gli identificativi vengono ricavati in maniera automatica a seguito della creazione dello stack), inoltre il servizio consente di ottenere i nomi delle risorse necessari per l'accesso dall'esterno, come le Url della coda SQS e dell'API Gateway.

A. Acquisizione Dati

La parte di acquisizione dati è composta da un sistema a coda di messaggi realizzato tramite il servizio AWS SQS, che conferisce elevate prestazioni, scalabilità automatica e tolleranza ai guasti.

Nello specifico è stata configurata una coda di tipo standard, poiché l'ordine degli eventi non è un aspetto rilevante per l'applicazione ed inoltre, in tal modo, si ottiene un throughput pressoché illimitato, grazie alla semplicità della semantica at least once ed al ridimensionamento dinamico automatico. Inoltre, tale semantica, garantisce comunque la consegna

del messaggio (timeout-based delivery), necessaria in quanto un messaggio non deve andare assolutamente perso perché potrebbe contenere informazioni su una caduta o una richiesta di SOS. In accordo con le linee guida di Amazon, la coda è stata configurata con un visibility timeout di 90 secondi pari a 6 volte il timeout della funzione lambda che processa i messaggi.

B. Processamento e Analisi Dati

Il sistema a coda di messaggi precedentemente discusso è configurato in modo tale da fungere da sorgente di eventi per la prima funzione lambda "eventProcessor" che li memorizza in DynamoDB e genera nuovi eventi per le funzioni di accertamento degli incidenti ("checkPosition", "checkHeartRate" e "checkFall"), le quali, a loro volta, compiono le opportune azioni ed eventualmente innescano la funzione "Notify" per la notifica e persistenza degli incidenti accertati. Quest'ultime vengono invocate a seguito della pubblicazione di messaggi associati a topic gestiti con il servizio AWS SNS, infatti sono configurate come trigger di code SQS sottoscritte ai topic. La scelta di AWS SNS consente di ottenere un elevato throughput ed una scalabilità automatica, inoltre le varie funzioni risultano così completamente disaccoppiate dalla eventProcessor. Le code SQS sono necessarie poiché SNS non garantisce che un evento venga processato in caso di fallimento/timeout della funzione lambda.

Di seguito sono descritte nel dettaglio le singole funzioni lambda.

1) Lambda - eventProcessor: La funzione "eventProcessor" è configurata in modo tale da avere come trigger la coda SQS posta in ingresso al sistema, in particolare il servizio AWS Lambda effettua un polling per controllare la presenza di nuovi messaggi, ed in caso ve ne siano, quest'ultimi vanno a costituire un batch che viene processato dalla funzione conseguentemente invocata. La dimensione massima del batch è impostata a 10, il massimo consentito, in modo tale da ridurre il numero di chiamate concorrenti in caso di carichi elevati e di conseguenza la possibilità di throttling.

Per ogni messaggio del batch, la funzione esegue una fase di pre-processamento delle informazioni contenute al suo interno, che sono relative alle misurazioni dei sensori del bracciale a cui sono associate. Nello specifico un generico messaggio corrisponde ad un evento con i seguenti campi:

- Identificativo dell'utente;
- Timestamp relativo alla misurazione sul dispositivo;
- Posizione attuale al verificarsi dell'evento, espressa in latitudine e longitudine;
- Frequenza cardiaca registrata;
- Dati del giroscopio relativo ai tre assi nel caso il bracciale abbia registrato una possibile caduta;
- Flag SOS presente se l'anziano effettua una richiesta di aiuto esplicita.

Il preprocessamento consiste in un controllo sequenziale dei campi ed una eventuale generazione di nuovi messaggi, questi vengono pubblicati nel sistema di consegna che realizza il pattern publish-subscribe a cui sono sottoscritte le code associate alle funzioni lambda di accertamento.

Per prima cosa viene confrontata la misurazione della frequenza cardiaca con un valore di soglia fisso uguale per ogni utente, se tale soglia viene superata viene pubblicato un messaggio con topic specifico, dopodiché viene controllata la presenza dei campi opzionali relativi al flag di SOS e alle misurazioni associabili ad una caduta e se accertata vengono pubblicati nuovi eventi con i corrispondenti topic.

Infine la funzione pubblica un topic per l'invocazione della funzione che controlla la posizione dell'utente e memorizza l'evento in una tabella di un database non relazionale tramite il servizio di storage AWS DynamoDB.

2) Lambda - checkHeartRate: La funzione checkHeartRate viene invocata a seguito della pubblicazione di un topic specifico e si occupa di confrontare il valore corrente della frequenza cardiaca con i dati forniti dall'utente in fase di registrazione al sistema.

In particolare la funzione sfrutta il servizio di AWS DynamoDB per recuperare la media $\overline{X}$ e la deviazione standard σ della frequenza cardiaca e controlla se il valore corrente x appartiene ad un intervallo di tolleranza tramite la seguente formula:

$$\overline{X} - \alpha\,\sigma \;\leq\; x \;\leq\; \overline{X} + \alpha\,\sigma \qquad\qquad \sigma > 0$$

dove α è una costante che regola l'ampiezza dell'intervallo. Nel caso in cui il valore non cada all'interno dell'intervallo la funzione pubblica un messaggio con il topic che consente l'invocazione della funzione di notifica all'utente.

3) Lambda - checkFall: La funzione checkFall viene invocata a seguito della pubblicazione di un topic specifico e si occupa di verificare che i dati forniti da giroscopio e accelerometro corrispondano o meno ad una caduta ed in caso affermativo genera un evento su SNS destinato alla Notify.

In particolare dai dati ricevuti vengono rimosse misurazioni erronee ed assenti, i dati dei tre assi di ogni sensore vengono poi normalizzati usando come dati di riferimento quelli di training.
A questo punto è possibile generare i dati da inoltrare al modello, per ogni asse e per ogni sensore viene generata una riga di statistiche che contiene: massimo, minimo, media, varianza, curtosi e skewness. Questo payload viene inoltrato all'endpoint che fornisce la sua predizione sull'evento.

Per la scelta del modello sono state provate due strategie diverse note in letteratura: xgboost e liblinear.
Xgboost è una libreria che implementa la tecnica degli alberi di Gradient boosting, impostata per risolvere problemi di regressione lineare. Liblinear implementa invece l'algoritmo SVMLight.
Al termine della cross validation Xgboost ha avuto performance predittive migliori di Liblinear, in particolare le performance del modello sul test set sono riassunte nella tabella I i cui dati mostrano un'accuratezza del 98%, una precisione del 97.9% ed una recall del 94%.

TABLE I
MATRICE DI CONFUSIONE

	Caduta (classe reale)	Non Caduta (classe reale)
Caduta	47	1
Non Caduta	2	136

Come dataset è stato scelto l'UMAFall Dataset [1] che consta delle registrazioni di 19 soggetti a cui è stato richiesto di simulare delle attività usuali e delle cadute. I loro movimenti sono stati registrati da 4 dispositivi di cui uno posizionato sul polso contenente accelerometro e giroscopio. Per adattare il dataset al nostro caso d'uso tutti gli altri rilevamenti sono stati quindi rimossi.
Una volta diviso il dataset in training e test set sono stati rimossi i dati fuori range e rimosse le righe contenenti i dati mancanti. Dai dati di training sono stati poi estratti i valori medi di ogni asse per accelerometro e giroscopio da poter usare per normalizzare i dati.

4) Lambda - checkPosition: L'ultima funzione di accertamento è la checkPosition che verifica se l'utente fuoriesce dalla proprio zona di sicurezza.

Per fare ciò la funzione recupera dalla tabella DynamoDB i dati corrispondenti al centro ed al raggio della security zone, per poi calcolare la distanza della posizione corrente dal centro tramite la formula dell'emisenoverso e controllare se è maggiore del raggio che delimita l'area.

Anche questa funzione, nel caso in cui il controlla abbia esito positivo, invia un messaggio con un topic per invocare la funzione di notifica.

C. Persistenza Dati

La persistenza dei dati viene realizzata tramite il servizio di storage AWS DynamoDB che consente di sfruttare tutti i vantaggi di un database non relazionale.

L'impiego di DynamoDB è giustificato da diversi fattori, quali ad esempio:

- Prestazioni: l'applicazione non richiede un grado di consistenza forte o operazioni di join;
- Elasticità: le tabelle sono sottoposte ad un numero di richieste proporzionale al carico del sistema;
- Flessibilità: gli eventi memorizzati contengono dati opzionali quindi risulta vantaggiosa la proprietà dell'assenza di schema.

Lo strato di storage è composto da 3 tabelle principali:

1) Tabella utenti: memorizza i dati invarianti del sistema, tra cui le informazioni relative alla frequenza cardiaca e alla security zone di ogni singolo utente, ma anche l'email a cui va inviata la notifica di incidente;
2) Tabella sensori: memorizza le ultime misurazioni effettuate dai bracciali, ogni record ha un TTL (Time To Live) e permane nel sistema per un tempo finito configurabile;
3) Tabella incidenti: memorizza tutti gli incidenti, affinché sia possibile avere uno storico degli incidenti di ogni paziente.

Le tabelle relative ai sensori sono configurate in modalità on-demand, ovvero la capacità di lettura e scrittura si adatta istantaneamente al carico di lavoro, mentre per la tabella incidenti si è optato per un provisioning statico poiché le richieste sono molto più sporadiche.

Per sfruttare al meglio la scalabilità del servizio la chiave primaria è stata scelta in base alle query che vengono effettuate: poiché queste consistono esclusivamente in interrogazioni dei dati associate ad un singolo utente, la chiave di partizione è identificata dall'id dell'utente, mentre il timestamp costituisce la chiave di ordinamento.

1) Lambda - notify: Oltre alla eventProcessor, che memorizza i dati delle misurazioni nella tabella dei sensori, anche la funzione notify si occupa di rendere i dati persistenti, infatti memorizza gli incidenti accertati nella tabella corrispondente.

Lo scopo principale però è quello di effettuare una notifica di avvertimento al "supervisore" dell'anziano in difficoltà inviando una mail, tramite il servizio AWS SES, al contatto che viene recuperato interrogando la tabella degli utenti.

D. Visualizzazione e Interrogazione Dati

1) Lambda - apiServer: La funzione "apiServer" funge da entry-point per l'applicazione e opera in combinazione con il servizio AWS API Gateway, il quale espone una serie di API REST per consentire l'accesso ai dati e la registrazione dell'utente.

La funzione si occupa di elaborare le richieste HTTP degli utenti, in particolare ne analizza il corpo in cui è specificata l'operazione che deve essere svolta ed esegue le azioni sulle corrispondenti tabelle di DynamoDB.

Le operazioni implementate sono le seguenti query:

- Lista delle ultime misurazioni del bracciale;
- Lista di tutti gli incidenti avvenuti.

Inoltre è possibile registrare nuovi utenti, inserendo parametri quali, ad esempio, battito medio e varianza dell'utente stesso.

III. SIMULAZIONE E RISULTATI SPERIMENTALI

L'applicazione è stata testata tramite un simulatore opportunamente implementato, che riproduce il comportamento di un utente generando eventi che rispecchiano distribuzioni di probabilità con parametri configurabili. La simulazione è eseguita per un tempo necessario al raggiungimento dello stato di stazionarietà del sistema e include anche un breve periodo di transitorio, per questioni pratiche ed economiche il simulatore prevede anche la possibilità di comprimere il tempo di un fattore anch'esso configurabile.

All'inizio della simulazione vengono generati utenti con caratteristiche che seguono una distribuzioni Gaussiane di probabilità con medie e varianze associabili ad un utente di età avanzata, come per esempio la frequenza cardiaca, la velocità di spostamento oppure il tempo che intercorre tra una caduta e l'altra. Il simulatore non fa altro che generare

eventi periodici e aperiodici in accordo alle caratteristiche degli utenti, ad esempio, per generare una caduta di un utente, vi è un thread che si sveglia ed invia un messaggio alla coda SQS ad intervalli che seguono una distribuzione esponenziale con media pari al periodo di caduta dell'utente stesso.

I dati del giroscopio associati alle attività non periodiche provengono dal dataset UMAFall, ad ogni risveglio dei relativi thread, viene pescato casualmente da una cartella locale un file contenente misurazioni reali di una qualche attività o caduta.

Le metriche prese in considerazione per la valutazione delle prestazioni al variare del carico in ingresso sono il tempo di risposta ed il throughput, nello specifico sono stati raccolti i risultati sperimentati nel sistema sottoposto ad un carico di 500, 1000 e 2000 utenti.

Le varie simulazioni sono state condotte con i seguenti parametri (riassunti in tabella II):

- SAMPLE_PERIOD = 600 secondi (6 minuti), tempo esatto che intercorre tra una misurazione periodica e l'altra del bracciale;
- AVG_FALL_PERIOD = 2592000 secondi (1 mese), media della distribuzione gaussiana per la generazione del periodo medio di caduta di un utente;
- AVG_ACTIVITY_PERIOD = 5400 secondi (16 eventi in 24 ore), media della distribuzione gaussiana per la generazione del periodo medio che intercorre tra le attività quotidiane di un utente;
- AVG_SOS_PERIOD = 1296000 secondi (2 eventi al mese), media della distribuzione gaussiana per la generazione del periodo medio che intercorre tra le richieste di aiuto di un utente;
- AVG_HRATE = 75 battiti al secondo, media della distribuzione gaussiana per la generazione della frequenza cardiaca media di un utente;
- VAR_HRATE = 15 battiti al secondo, media della distribuzione gaussiana per la generazione della varianza della frequenza cardiaca di un utente;
- AVG_SPEED = 0.0005 chilometri al secondo, media della distribuzione gaussiana per la generazione della velocità media di un utente;
- VAR_SPEED = 0.0001 chilometri al secondo, media della distribuzione gaussiana per la generazione della varianza della velocità di un utente;
- SAFETY_PROB = 0.7, probabilità che un utente non fuoriesca dalla sua security zone.

TABLE II
PARAMETRI DEL SIMULATORE

SAMPLE_PERIOD	600 sec (6 minuti)
AVG_FALL_PERIOD	2592000 sec (1 mese)
AVG_ACTIVITY_PERIOD	5400 sec (16 eventi in 24 ore)
AVG_SOS_PERIOD	1296000 sec (2 eventi al mese)
AVG_HRATE	75 battiti / s
VAR_HRATE	15 battiti / s
AVG_SPEED	0.0005 km / s
VAR_SPEED	0.0001 km / s
SAFETY_PROB	0.7

I parametri riguardanti le risorse del sistema (tabelle III, IV, V) sono stati opportunamente calibrati in maniera empirica e tenendo conto delle linee guide di amazon, nello specifico ogni funzione lambda processa un batch di dimensione massima pari a 10 con timeout impostati sulla base del tempo massimo sperimentato durante l'esecuzione di un evento. I visibility timeout delle code sono pari ad almeno sei volte il timeout delle funzioni lambda corrispondenti. Anche per quanto riguarda la capacità di scrittura/lettura dei database DynamoDB si è agito in maniera empirica laddove le richieste variano proporzionalmente al carico del sistema e mantenendo la soglia minima consigliata dove invece sono costanti.

TABLE III
PARAMETRI DELLE RISORSE LAMBDA

	BatchSize	Timeout [s]
eventProcessor	10	15
checkPosition	10	8
checkHeartRate	10	8
checkFall	10	8
notify	10	150

TABLE IV
PARAMETRI DELLE RISORSE SQS

	Timeout [s]
IngestionSQSQueue	90
PositionSQSQueue	50
HeartRateSQSQueue	50
FallSQSQueue	50
NotifySQSQueue	900

TABLE V
PARAMETRI DELLE RISORSE DYNAMODB

	ReadCapacityUnits	WriteCapacityUnits
User	on-demand	on-demand
Sensor	on-demand	on-deman
Accident	5	5

A. Throughput

Il throughput del sistema è stato calcolato considerando il numero di completamenti nell'unità di tempo (1 secondo), dove per completamento si intende l'istante in cui termina l'ultima funzione che processa un certo evento entrante nel sistema. Ad esempio, un evento di sospetta caduta è processato sicuramente dalla eventProcessor, dalla checkPosition e dalla checkFall, ma può anche essere processato dalla checkHeartRate se il valore dei battiti supera la soglia e dalla notify se si accerta la caduta, pertanto l'istante di completamento corrisponde al massimo tra gli istanti di completamento delle funzioni che coinvolgono l'evento.

Per effettuare questo calcolo è stato sfruttato il servizio AWS CloudWatch Logs con il quale è stato possibile tener traccia dell'evento processato da ogni funzione, stampando

nel log un identificativo dell'evento ottenuto dalla concatenazione dell'id del bracciale con il timestamp di generazione dell'evento, in modo da garantirne l'univocità all'interno del sistema. Infine tramite opportune query e manipolazioni dei log si riesce a calcolare il tempo di completamento di ogni evento che transita nel sistema.

I risultati delle simulazioni (figura 1 e tabella VI) mostrano come il valore medio e la varianza del throughput del sistema aumentano in maniera proporzionale al numero degli utenti, ottenendo vari pari a 3.65, 7.2 e 13.64 completamenti al secondo per un carico rispettivamente di 500, 1000 e 2000 utenti.

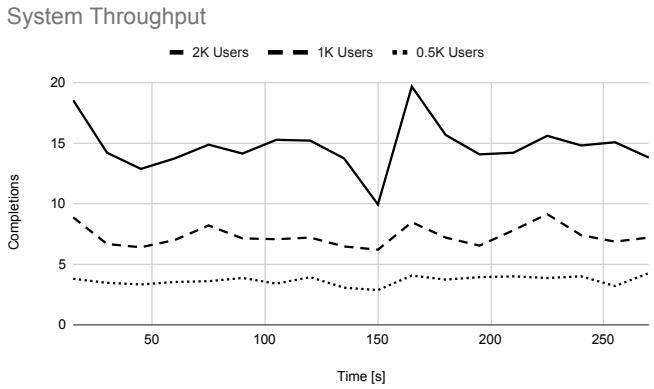

Fig. 1. Throughput di sistema

TABLE VI
THROUGHPUT DI SISTEMA

utenti	avg	std_dev
500	3.65	0.37
1000	7.2	1.15
2000	14.64	2.10

TABLE VII
THROUGHPUT EVENTPROCESSOR

utenti	avg	std_dev
500	54.75	5.56
1000	108.12	15.72
2000	219.9	34.91

TABLE VIII
THROUGHPUT CHECKPOSITION

utenti	avg	std_dev
500	54.75	5.56
1000	108.12	15.72
2000	219.9	34.91

Riguardo le singole funzioni lambda (tabelle VII, VIII, IX, X e XI) l'andamento è lo stesso, ma i valori sono influenzati dalla frequenza di invocazione delle funzioni, infatti si nota come il throughput medio della eventProcessor e della checkPosition sono molto maggiori delle altre funzioni poiché sono invocate per ogni evento entrante nel sistema.

TABLE IX
THROUGHPUT CHECKHEARTRATE

utenti	avg	std_dev
500	15.75	1.75
1000	32.85	2.47
2000	64.525	6.1

TABLE X
THROUGHPUT CHECKFALL

utenti	avg	std_dev
500	4.95	2.8
1000	9.56	3.96
2000	20.15	5.9

TABLE XI
THROUGHPUT NOTIFY

utenti	avg	std_dev
500	2.18	1.27
1000	3.95	1.43
2000	7.37	3.46

B. Tempo di Risposta

Il tempo di risposta del sistema è calcolato effettuando la media dei tempi di risposta di tutti gli eventi, ottenuti sommando le durate di esecuzione di ciascuna funzione lambda in cui un evento è coinvolto. Anche in questo caso è stato impiegato il servizio AWS Cloudwatch Logs per tenere traccia delle associazioni tra funzioni lambda ed eventi.

I risultati sperimentali (tabella XII) mostrano come, al variare del carico in ingresso, il sistema mantiene una latenza costante che si attesta intorno ai 930 millisecondi, ma subisce un lieve incremento nel caso relativo ai 2000 utenti. La causa è da ricercarsi nella funzione notify che si comporta come un collo di bottiglia, infatti SES ha un limite di invio di 10 mail al secondo che viene superato nel caso di duemila utenti. È comunque possibile richiedere al supporto AWS un incremento di tale frequenza.

Degno di nota è anche il comportamento della eventProcessor, che nel caso di 2000 utenti, presenta una latenza media ridotta rispetto ai carichi inferiori. Ciò può essere giustificato dal fatto che la funzione processa un batch più numeroso e la componente del tempo di esecuzione relativa all'avvio della stessa è condivisa da più eventi, pertanto ne consegue una riduzione del tempo di esecuzione per ognuno di essi. Nella figura 2 sono mostrati i dati della latenza al variare del carico aggregati tramite media ogni 5 secondi e nelle tabelle XIII, XIV, XV, XVI, XVII sono presenti media e varianza per la latenza ed inoltre il valore massimo registrato corrispondente al "cold start" delle singole funzioni.

TABLE XII
TEMPO DI RISPOSTA DEL SISTEMA

utenti	avg	std_dev	max
500	927.74	353.02	3825.92
1000	933.93	349.51	3713.27
2000	1139.86	1606.12	16034.96

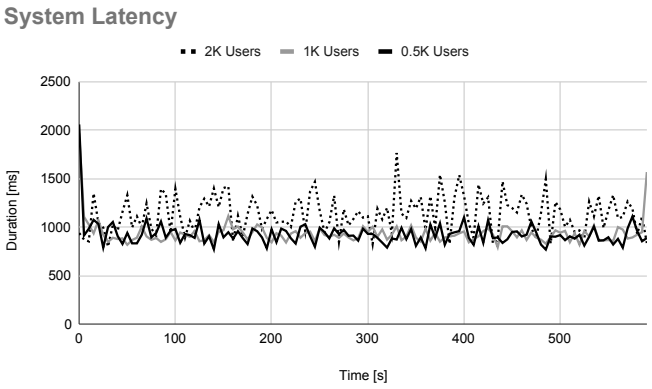

Fig. 2. Tempo di Risposta del sistema

TABLE XIII
TEMPO DI RISPOSTA EVENTPROCESSOR

utenti	avg	std_dev	max
500	600.96	206.24	2382.2
1000	606.79	210.62	2893.1
2000	565.96	151.86	1969.33

TABLE XIV
TEMPO DI RISPOSTA CHECKPOSITION

utenti	avg	std_dev	max
500	259.89	72.55	1438.83
1000	260.84	70.48	1420.19
2000	259.53	63.09	1415.02

TABLE XV
TEMPO DI RISPOSTA CHECKHEARTRATE

utenti	avg	std_dev	max
500	271.035	125.97	1635.53
1000	268.675	114.82	1384.67
2000	261.160	97.15	1394.47

TABLE XVI
TEMPO DI RISPOSTA CHECKFALL

utenti	avg	std_dev	max
500	173.24	71.39	790.35
1000	169.53	56.86	849.42
2000	165.63	49.86	848.3

TABLE XVII
TEMPO DI RISPOSTA NOTIFY

utenti	avg	std_dev	max
500	942.79	210.57	2176.53
1000	925.57	202.17	2163.47
2000	8005.37	2663.86	14563.75

IV. INSTALLAZIONE, CONFIGURAZIONE ED ESECUZIONE

Il codice del progetto è disponibile pubblicamente al seguente link https://github.com/smvfal/nonno-care e contiene già il dataset utilizzato per l'addestramento del modello di machine learning.

A. Installazione e Configurazione

Per inizializzare l'ambiente di esecuzione ed effettuare il deployment dell'applicazione, prima di tutto è necessario avere un client aws-cli installato e configurato con un ruolo in grado di poter compiere operazioni di creazione e modifica di uno stack CloudFormation.
È anche necessario avere installato boto3 che va configurato con un ruolo IAM avente i requisiti sopra descritti, inoltre devono essere installate le librerie utilizzate nello script python per l'addestramento del modello predittivo, questo può essere fatto tramite pip3 con il comando:

```
$ pip3 install -r ml_training/requirements.txt
```

Installare anche i requisiti per il simulatore se si vuole testare l'applicazione:

```
$ pip3 install -r wirlband_simulator/requirements.txt
```

B. Deployment

Per effettuare il deploy occorre creare un bucket S3 su cui verrà salvato il package contenente le informazioni necessarie per la creazione dello stack di risorse ed il modello predittivo con i relativi dati di training e test. Dopodiché è sufficiente avviare lo script dell'addestramento del modello

```
$ BUCKET=sample-bucket-name
$ bash train_model.sh $BUCKET
```

ed infine avviare lo script per il packaging ed il deployment delle risorse passando il nome del bucket e la regione scelta[1]

```
$ REGION=sample-region
$ bash deploy.sh $REGION $BUCKET
```

Le operazioni potrebbero richiedere qualche minuto per completare il processo di training e di instanziazione delle risorse.

C. Testing

Per effettuare il testing dell'applicazione è necessario innanzitutto registrare un indirizzo email sul quale verranno ricevute le e-mail di notifica:

```
$ bash verify_email.sh receiver@example.com
```

e concedere l'autorizzazione tramite il link conseguentemente ricevuto. Tale indirizzo va inoltre impostato nel file di configurazione config.ini presente nella cartella wirlband_simulator, sul quale è possibile impostare a piacimento tutti i parametri della simulazione.

Il simulatore può essere eseguito con un numero arbitrario di utenti, ad esempio con 2 utenti:

```
$ cd wirlband_simulator/
wirlband_simulator$ python3 simulator.py 2
```

Oppure se si vuole testare l'applicazione con carichi più consistenti si possono creare un numero arbitrario di processi (entro i limiti della propria macchina), ad esempio per creare 2 processi da 500 utenti ciascuno basta eseguire il comando:

[1]Nella regione scelta devono necessariamente essere disponibili i servizi utilizzati dall'applicazione (S3, CloudFormation, SageMaker, SES, DynamoDB, Lambda, SNS, SQS, API Gateway)

```
$ bash start_sensors.sh 2
```

e per terminare la simulazione:

```
$ bash stop_sensors.sh
```

In entrambi i casi gli utenti verranno registrati automaticamente nel sistema.

Infine è possibile effettuare le query tramite protocollo HTTP, ad esempio per ottenere i dati in formato json delle ultime misurazioni è possibile inviare una richiesta POST sfruttando i body di esempio presenti nella cartella sample_events e lo script describe_stack.sh per ottenere la URL dell'endpoint API Gateway:

```
$ bash describe_stack $REGION
$ API_ENDPOINT=api-endpoint-url
$ curl -d "@sample_events/read_sensor.json" \
> -X POST $API_ENDPOINT
```

V. Conclusioni e sviluppi futuri

In conclusione il sistema risulta quindi mantenere una latenza pressoché costante al variare degli utenti registrati con un throughput che è invece proporzionale a questi ultimi. Si è visto anche che, all'aumentare della dimensione dei batch di eventi, si ha un lieve incremento delle prestazioni per quanto riguarda i job da compiere per singolo evento grazie all'esecuzione di più eventi in una sola funzione.

AWS Lambda ha permesso di produrre rapidamente l'applicativo senza dover acquistare, affittare o gestire server, la grande eterogeneità dei servizi offerti da AWS ha consentito di integrare facilmente tecnologia nell'applicativo stesso: si è infatti potuto fornire agli utenti un sistema di riconoscimento delle cadute basato su tecniche di machine learning con ottima affidabilità e minimo sforzo dovuto all'implementazione degli algoritmi ed al deploy automatico su macchina virtuale.

I sistemi a coda utilizzati hanno permesso infine di garantire il processamento affidabile dei dati, requisito essenziale per un dispositivo di monitoraggio di persone parzialmente non autosufficienti.

I potenziali sviluppi futuri per questo applicativo sono molteplici, eccone solo alcuni tra i più semplici da immaginare:

- Distribuzione multiregionale, prevedendo un autoscaling per l'endopoint di machine learning. Per il numero di utenti testato la macchina più piccola disponibile è risultata essere molto sottoutilizzata, ragion per cui non è stata implementata tale caratteristica;
- Inserimento di un sistema di notifica tramite SMS[2] o applicazione mobile/web dal quale è possibile registrarsi e visualizzare gli ultimi eventi tramite una interfaccia user friendly;
- Aggiornamento dinamico e adattativo dei parametri dell'utente, come la frequenza cardiaca media, basato sui dati che si ricevono periodicamente dal bracciale.

[2]Per il momento il servizio di messaggistica SMS offerto da SNS è disponibile per numeri telefonici appartenenti soltanto ad alcuni Paesi, tra questi non è ancora presente l'Italia.

References

[1] E. Casilari and J. A.Santoyo-Ramón, "UMAFall: Fall Detection Dataset (Universidad de Malaga)," 6 2018. [Online]. Available: https://figshare.com/articles/UMA_ADL_FALL_Dataset_zip/4214283