Sensor Continuum: Distributed application in the compute continuum

Project A3 – Sistemi Distribuiti e Cloud Computing

Maurizio Renzi
Studente del Corso di Laurea Magistrale in Ingegneria Informatica
Università degli Studi di Roma "Tor Vergata"
Matricola 0369222, A.A. 2024/2025
maurizio.renzi@students.uniroma2.eu

Francesco Masci
Studente del Corso di Laurea Magistrale in Ingegneria Informatica
Università degli Studi di Roma "Tor Vergata"
Matricola 0365258, A.A. 2024/2025
francesco.masci.02@students.uniroma2.eu

***Abstract* — Questo report presenta la progettazione, l'implementazione e la valutazione di un sistema di sensori distribuiti che sfrutta il paradigma del Compute Continuum. Il progetto affronta le sfide relative all'elaborazione e alla comunicazione dei dati in reti di sensori su larga scala, distribuendo i compiti computazionali su cinque livelli distinti: sensori, nodi edge, nodi proximity, nodi intermediate e servizi cloud. L'architettura proposta sfrutta le capacità uniche di ogni strato, dall'elaborazione a bassa latenza sull'edge all'alta potenza di calcolo del cloud, ottimizzando così l'uso delle risorse e riducendo al minimo la latenza di rete.**

Il sistema è stato concepito come un'applicazione completamente decentralizzata, impiegando architetture a microservizi e serverless per garantire scalabilità e resilienza. L'implementazione integra servizi AWS per funzionalità chiave, inclusa un'API client per recuperare e gestire i dati dei sensori.

Keywords* — *Compute Continuum, AWS, Architettura Distribuita, Microservizi, Serverless, IoT, MQTT, Kafka, Go

I. Introduzione

L'esponenziale crescita dei dati generati dall'Internet of Things e dalle reti di sensori ha reso le architetture tradizionali basate sul cloud computing insufficienti. L'invio di tutti i dati grezzi al cloud introduce infatti latenze significative, costi elevati di banda e problemi di scalabilità, compromettendo l'efficienza delle applicazioni in tempo reale. Per superare queste limitazioni, il Compute Continuum emerge come un nuovo paradigma.

Il Compute Continuum estende la capacità di calcolo dal cloud fino al "bordo" della rete, creando un'infrastruttura distribuita e olistica. Questo approccio integra cinque livelli di elaborazione—dai sensori ai dispositivi edge, proximity, intermediate, fino ai servizi cloud—per allocare le risorse computazionali nel punto più opportuno, riducendo la latenza ed eseguendo l'elaborazione il più vicino possibile alla fonte.

Il presente lavoro descrive la progettazione e l'implementazione di un sistema di sensori distribuiti che sfrutta pienamente questo paradigma. L'architettura proposta distribuisce dinamicamente i carichi di lavoro su cinque livelli di calcolo, impiegando microservizi e serverless con servizi AWS. Il nostro contributo principale consiste nella dimostrazione pratica di come questa architettura gestisca in modo efficace un sistema di sensori su larga scala, fornendo un esempio concreto e funzionale per l'applicazione del Compute Continuum in ambienti reali. Il resto del paper è strutturato come segue: nella Sezione 2 si presenta il Background; la Sezione 3 illustra la Progettazione della Soluzione; nella Sezione 4 si approfondiscono i Dettagli di Implementazione; la Sezione 5 presenta i Risultati mentre la Sezione 6 la Discussione; infine, le Conclusioni sono riportate nella Sezione 7, seguite dai Riferimenti.

II. Background

Il presente lavoro risponde alla traccia del corso di Sistemi Distribuiti e Cloud Computing, che richiede la progettazione e l'implementazione di un'applicazione distribuita che sfrutti il concetto di Compute Continuum. Per l'implementazione è stato scelto il linguaggio Go, data la sua efficienza e la gestione nativa della concorrenza.

Il nucleo teorico risiede nella transizione dal modello centralizzato del Cloud Computing a un'architettura pervasiva. Sebbene il cloud offra scalabilità e risorse on-demand, esso introduce limiti intrinseci per le applicazioni IoT a bassa latenza, come ritardi significativi, elevati costi di banda e problemi di scalabilità derivanti dall'invio di tutti i dati grezzi.

Il Compute Continuum affronta queste sfide estendendo l'infrastruttura di calcolo ai confini della rete. Non si tratta semplicemente di Edge o Fog Computing come entità separate, ma di un ecosistema coeso in cui le risorse computazionali collaborano in modo trasparente, dal dispositivo più piccolo al data center.

L'adozione di un'architettura a Microservizi si sposa perfettamente con questo paradigma, consentendo lo sviluppo e lo scaling indipendente di ogni servizio. La natura leggera dei microservizi ne facilita il posizionamento sui nodi edge con risorse limitate. Parallelamente, l'architettura Serverless offre un modello di esecuzione on-demand che ottimizza i costi e l'efficienza, eliminando la gestione dell'infrastruttura sottostante. La combinazione di Microservizi e Serverless costituisce la base per la progettazione della nostra soluzione, un sistema di sensori resiliente ed efficiente che sfrutta l'intero spettro del Compute Continuum.

III. Progettazione della Soluzione

La nostra architettura è stata progettata su una gerarchia a cinque livelli. Ogni strato ha compiti specifici che contribuiscono all'efficienza complessiva del sistema, dalla raccolta dei dati grezzi all'analisi di alto livello. Il sistema è intrinsecamente distribuito e autonomo, con ogni componente che si registra al suo avvio per una gestione dinamica e scalabile.

A. Suddivisione Gerarchica del Sistema

La struttura si fonda su una suddivisione spaziale e gerarchica dell'ambiente. L'intero spazio di interesse è suddiviso in regioni. Ogni regione è a sua volta composta da diverse macrozone. Il livello più granulare di questa gerarchia è la zona, un'area fisica più piccola (ad esempio, un piano di un edificio). Infine, all'interno di ogni zona, sono situtati i sensori, che rappresentano i punti di raccolta dati.

B. Architettura e Componenti

1) Sensor Agent

Il livello più basso del continuum è rappresentato dal Sensor Agent, un'entità software che simula il comportamento di un sensore fisico. La sua responsabilità principale è la generazione periodica di misurazioni per una specifica metrica (ad esempio, temperatura, umidità, pressione, ecc.). Questi valori sono simulati a partire da una distribuzione basata su un dataset reale, garantendo che i dati generino flussi che riflettono le dinamiche di un ambiente fisico. I dati vengono inviati all'Edge Hub di competenza.

2) Edge Hub

L'Edge Hub agisce come hub di zona, gestendo un gruppo di sensori. La sua funzione primaria è l'elaborazione dei dati grezzi a bassa latenza. Riceve i valori dai sensori e li filtra per rimuovere gli outlier, scartando le misurazioni che si discostano in modo significativo dalla distribuzione storica del sensore. I dati validati sono salvati in una cache locale volatile per l'analisi immediata. Periodicamente, viene attivato un servizio di aggregazione che calcola la media dei dati in un intervallo di tempo specifico e li inoltra al Proximity Hub. Per mantenere l'affidabilità del sistema, l'Edge Hub pulisce regolarmente la sua cache, rimuovendo i dati dei sensori inattivi per troppo tempo, che vengono considerati non operativi (*unhealthy*).

3) Proximity Hub

Il Proximity Hub gestisce un insieme di zone, configurandosi come un hub di macrozona. Riceve i dati aggregati dagli Edge Hub e li salva nella sua cache locale persistente con uno stato di "pending". Utilizzando il pattern Transactional Outbox, i dati vengono poi inviati all'Intermediate Hub. Oltre alla gestione della cache, il Proximity Hub esegue il primo livello di aggregazione statistica, calcolando periodicamente le metriche aggregate sia a livello di singola zona che di macrozona. Entrambi i dati (i valori aggregati della zona e le statistiche della macrozona) vengono inoltrati all'Intermediate Hub.

4) Intermediate Hub

L'Intermediate Hub è responsabile della gestione dei dati a livello di regione. Riceve i dati aggregati dal Proximity Hub, li raccoglie in batch e, al raggiungimento di una capacità massima o allo scadere di un timeout, li salva in blocco nel database della regione, che funge da archivio a lungo termine. Questo approccio a batch processing ottimizza l'efficienza di scrittura. L'Intermediate Hub calcola anche le statistiche a livello di regione, aggregando i dati provenienti da tutte le macrozone sotto la sua giurisdizione.

5) Servizi Cloud

Il livello più alto del continuum è il Cloud, dove sono stati implementati servizi serverless e API. Questi servizi offrono funzionalità di accesso ai dati a lungo termine memorizzati nei database di regione. Funzioni serverless aggiuntive sono utilizzate per eseguire analisi complesse, come il calcolo delle variazioni annuali, la correlazione tra le variazioni annuali di nodi vicini e l'analisi di similarità tra il trend di una macrozona e quello della sua regione di appartenenza.

C. Comunicazione tra Componenti

Per gestire in modo efficiente il flusso di dati attraverso i diversi livelli del continuum, il sistema adotta una duplice metodologia di comunicazione basata sui protocolli MQTT e Kafka.

1) MQTT

MQTT viene impiegato nel livello più basso dell'architettura per la comunicazione tra i Sensor Agent e gli Edge Hub, ma anche per quella tra gli Edge Hub e il Proximity Hub. La scelta di MQTT è giustificata dalla sua natura leggera, dal basso consumo di banda e dalla sua efficienza in ambienti a risorse limitate.

I sensori pubblicano i loro dati su topic specifici, consentendo agli Edge Hub di iscriversi selettivamente ai flussi di dati di loro interesse. Questo modello di comunicazione publish-subscribe riduce il traffico di rete e garantisce la flessibilità necessaria per scalare il numero di sensori senza sovraccaricare la rete.

A livello di macrozona, il sistema può adottare due configurazioni: un singolo broker MQTT a livello di macrozona per gestire la comunicazione sia tra i sensori e gli Edge Hub che tra gli Edge Hub e il Proximity Hub, oppure un broker per ogni zona, dedicato esclusivamente alla comunicazione Sensor Agent-Edge Hub, con un broker separato a livello di macrozona per la comunicazione tra gli Edge Hub e il Proximity Hub.

2) Kafka

A partire dal livello Proximity Hub e a salire verso il cloud, la comunicazione tra i componenti principali è gestita tramite Apache Kafka.

Kafka è stato scelto per la sua capacità di agire come un message broker distribuito e per la sua robustezza, garantendo un'alta disponibilità e la tolleranza ai guasti. Il suo modello di append-only log è ideale per gestire flussi di dati a volume elevato e per consentire ai consumatori (gli hub superiori) di leggere i dati in modo efficiente e asincrono. L'uso di Kafka facilita il decoupling tra i vari livelli, assicurando che un componente non debba attendere che il successivo sia

disponibile, e offre una soluzione scalabile per il buffering dei dati e la gestione di picchi di traffico.

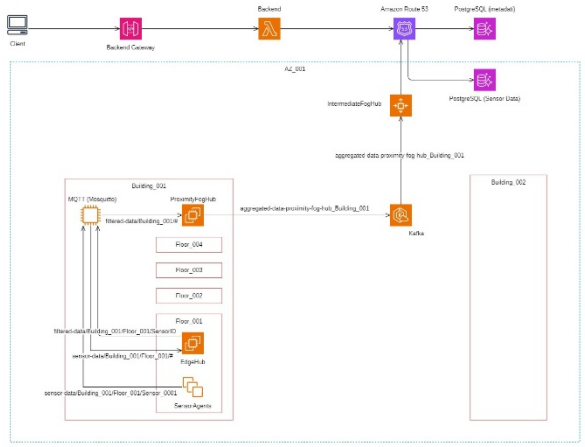
D. *Flusso dei Messaggi di Configurazione e Heartbeat*

Oltre al flusso dei dati, il sistema gestisce in modo proattivo la propria configurazione e lo stato di salute dei componenti attraverso un flusso di messaggi di controllo. Al momento dell'avvio, gli hub (Edge, Proximity e Intermediate) inviano un messaggio di registrazione al nodo gerarchicamente superiore, il quale include al suo interno i metadati dell'hub. Queste informazioni permettono all'hub superiore di mantenere un registro aggiornato a livello di regione.

Per il monitoraggio continuo, gli hub inviano periodicamente messaggi di heartbeat al nodo gerarchicamente superiore. Il Sensor Agent non invia heartbeat, poiché il suo stato di salute viene dedotto dal flusso continuo dei dati di misurazione. Se un hub smette di inviare heartbeat, il suo stato viene considerato "non attivo", consentendo agli operatori del sistema di identificare tempestivamente eventuali guasti e di agire di conseguenza.

E. *Processo di Deployment*

Il processo di deployment è stato interamente automatizzato per garantire coerenza, efficienza e riproducibilità. L'infrastruttura AWS è definita mediante script automatici che sfruttano AWS CloudFormation per la creazione e la gestione degli ambienti. Ogni componente software è containerizzato in Docker, e il deployment a livello locale o su istanza è gestito tramite Docker Compose, che ne orchestra le dipendenze essenziali come cache esterne o database.



Basandoci sul diagramma architetturale fornito, la distribuzione dei componenti segue una logica a strati, partendo dai nodi più vicini alla fonte dei dati e salendo verso il cloud. I Sensor Agent e gli Edge Hub sono distribuiti sui nodi periferici, rappresentando il primo livello del continuum.

Il Proximity Hub è posizionato strategicamente tra i nodi edge e lo strato fog, gestendo una serie di Edge Hub e fungendo da punto di aggregazione intermedio. Idealmente, tutti i componenti all'interno della stessa macrozona comunicano su una rete interna dedicata, garantendo bassa latenza e isolamento del traffico. Questo posizionamento ibrido è cruciale per la riduzione della latenza e l'ottimizzazione del traffico di rete.

L'Intermediate Fog Hub, che gestisce la logica di aggregazione e il salvataggio dei dati in un database dedicato, è implementato su una macchina virtuale o un container in un ambiente fog, strettamente legato ai nodi periferici. Il sistema Kafka è anch'esso gestito in questo strato intermedio per una comunicazione efficiente. L'accesso all'Intermediate Hub, che avviene tramite Kafka, può essere configurato per essere esposto o meno tramite connessione internet, a seconda dei requisiti di sicurezza e connettività del deployment specifico.

Per il livello cloud, il Backend è implementato come funzione serverless AWS Lambda, attivata tramite un Backend Gateway (API Gateway) per servire le richieste dei client. I metadati e le configurazioni di regioni, zone e macrozone sono archiviate in un database PostgreSQL separato e gestito globalmente.

L'architettura sfrutta anche Amazon Route 53 per la gestione del routing DNS, garantendo un accesso affidabile al backend.

IV. Dettagli di Implementazione

L'implementazione del sistema è stata concepita per rispecchiare fedelmente l'architettura distribuita e gerarchica del progetto, garantendo robustezza, scalabilità e autonomia per ciascun componente.

La configurazione di tutti i parametri operativi, come gli indirizzi dei broker e gli intervalli di tempo, è gestita tramite variabili d'ambiente, semplificando il deployment e la manutenzione.

Ogni modulo software si avvale di una libreria di logging personalizzata e condivisa che registra informazioni dettagliate su eventi ed errori. Per la gestione dei guasti, si è adottato un approccio fail-fast: in caso di errori critici, il processo termina con un codice di errore (1), consentendo a un orchestratore di container come Docker Compose di attivare un riavvio automatico, che riporta il servizio a uno stato iniziale e pulito.

L'implementazione interna di tutti gli hub fa un uso estensivo delle Go routines e dei canali, dove i servizi dedicati alla raccolta e all'invio dei dati utilizzano Go routines separate per leggere i messaggi dai broker e inviarli alle Go routines di elaborazione tramite canali. La gestione interna dei casi in cui un canale è pieno prevede di riprovare ad inviare il messaggio o scartarlo.

A. *Sensor Agent*

Il Sensor Agent è un componente software progettato per simulare in modo realistico un sensore fisico. Per creare una distribuzione di dati credibile, si avvale dei valori storici forniti dal sito *Sensor.Community*, che espone le rilevazioni di sensori in formato CSV. Fornendo il riferimento di un sensore e il tipo di misurazione (es. temperatura, umidità, pressione, …), il Sensor Agent scarica i dati di due giorni prima, estrae i valori dell'ora corrente e genera casualmente nuove letture con una media e una deviazione standard allineate a quelle dei dati reali in quell'ora. Per aumentare il realismo della simulazione, il componente introduce volontariamente outlier e dati mancanti, replicando guasti o perdite di messaggi.

Il file CSV viene scaricato all'avvio del container; pertanto, per mantenere uno stato pulito e aggiornare i dati di riferimento, i container devono essere riavviati automaticamente ogni notte.

Ogni sensore comunica pubblicando le proprie misurazioni su un topic MQTT a lui dedicato.

B. Edge Hub

L'Edge Hub (o Zone Hub) è implementato come un insieme di quattro servizi indipendenti, ottimizzati per l'esecuzione su dispositivi a risorse limitate ai bordi della rete.

Si assume che il deployment avvenga su un singolo dispositivo, e la sua architettura riflette questa premessa. I servizi condividono una cache locale implementata in Redis, definita tramite Docker Compose.

I servizi che compongono l'Edge Hub sono i seguenti:

- **Servizio di Filtro**: Gestisce i dati in ingresso dai sensori, li salva nella cache e scarta gli outlier. Per il calcolo degli outlier, l'hub necessita di una storia recente del sensore, pertanto i primi messaggi (il cui numero è configurabile) vengono considerati affidabili per costruire tale storia. La storia viene salvata in Redis per essere accessibile a tutti i servizi. La scalabilità orizzontale di questo servizio è garantita dalle shared subscriptions di MQTTv5, che distribuiscono i messaggi in arrivo a un solo consumer per gruppo, bilanciando il carico senza duplicazioni. La Quality of Service su MQTT per i dati in ingresso è impostata a at-most-once, poiché l'hub non gestisce i duplicati.
- **Servizio di Aggregazione**: Periodicamente, questo servizio aggrega i valori dell'ultimo minuto presenti nella cache e li inoltra al Proximity Hub tramite MQTT. Per compensare ritardi e latenze, l'aggregatore calcola l'intervallo di aggregazione con un offset di 2 minuti rispetto all'ora corrente, arrotondando per difetto il minuto. Ne consegue che i dati inviati al Proximity Hub hanno uno shift temporale di 2 minuti. Tali messaggi sono registrati con timestamp l'istante di fine intervallo.
- **Servizio di Pulizia**: Periodicamente, questo servizio controlla la cache per identificare i sensori i cui ultimi messaggi sono stati inviati prima di un timeout (impostato a 5 minuti), notificandoli come unhealthy. Un secondo timeout di 6 ore identifica i sensori ritenuti non più presenti nel sistema. Entrambi i meccanismi sono predisposti per notificare via email i manutentori, sebbene questa opzione non sia stata completamente implementata.
  La gestione della concorrenza è risolta tramite una leader election semplice, utilizzando un lock distribuito in Redis. Prima di eseguire le aggregazioni, il servizio relativo tenta di diventare leader. Se il tentativo fallisce, salta la computazione. I lock di leader hanno una durata precisa, allineata allo scoccare del minuto successivo di inizio di una nuova aggregazione. Per evitare che il servizio si avvii in un momento critico, l'edge hub ritarda l'inizio del suo timer periodico per le aggregazioni, se necessario. Il motivo è prevenire bug che potrebbero verificarsi se il servizio iniziasse l'aggregazione proprio mentre sta per scadere la validità del lock. Posticipando l'avvio, l'edge hub si assicura che le operazioni non si sovrappongano ai momenti di transizione, garantendo che l'acquisizione del lock avvenga senza errori.
- **Servizio di Configurazione**: Per la gestione dei messaggi di configurazione, l'Edge Hub riceve su un topic dedicato messaggi di registrazione dai sensori. Tali messaggi sono retained, in modo da essere mantenuti anche se gli hub non sono connessi in quel momento. Dopo averli processati e inoltrati al Proximity Hub, il messaggio sul topic viene eliminato. In questo caso, il lock distribuito è gestito direttamente da Redis, che garantisce l'esecuzione atomica dell'operazione di PUT dei metadati nella cache. Se due processi concorrenti tentano di inserire un sensore, uno dei due constaterà la presenza del valore ed eviterà di propagare il messaggio. Il comportamento è mantenuto anche per i messaggi successivi: l'hub salta l'inserimento e la propagazione se il valore è già presente, riducendo il traffico di rete. La QoS per i messaggi di configurazione è exactly-once.

C. Proximity Hub

Il Proximity Hub (o Macrozone Hub) è un componente cruciale dell'architettura Fog, progettato per operare su un server con capacità di calcolo superiori a quelle dei dispositivi Edge.

La sua architettura è modulare e si compone di sei servizi distinti che lavorano in sinergia. La cache locale è implementata utilizzando un database PostgreSQL, garantendo la persistenza dei dati a un livello superiore rispetto alla cache volatile di Redis dell'Edge Hub. Sebbene l'istanza di deployment tipica sia su un singolo server, l'architettura è predisposta per supportare una configurazione con un server dedicato per l'elaborazione e un server separato per il database di cache, garantendo maggiore sicurezza e scalabilità. I messaggi in ingresso sono distribuiti ai vari componenti tramite i gruppi MQTT per bilanciare il carico.

I sei moduli principali sono:

- **Modulo Local Cache**: Questo servizio è responsabile della ricezione dei dati grezzi aggregati dagli Edge Hub tramite MQTT e del loro salvataggio nella cache locale. A differenza dell'Edge Hub, questo modulo implementa una logica di de-duplicazione a livello di database, rendendo il servizio idempotente. La comunicazione sfrutta una semantica at-least-once con il broker MQTT, sapendo che i duplicati verranno gestiti e risolti dalle appropriate query nel database. Una volta inseriti, i record dei sensori vengono impostati

su uno stato *"pending"*, indicando che sono in attesa di essere inviati al livello superiore.

- **Modulo Aggregator**: Questo componente calcola le statistiche aggregate sia a livello di zona che di macrozona. Per mitigare i ritardi di pacchetto e garantire la stabilità temporale delle analisi, il calcolo non avviene in tempo reale. L'aggregatore elabora un intervallo di tempo di 15 minuti con un offset di 10 minuti rispetto all'ora corrente. Ad esempio, se l'ora attuale è 15:48, l'aggregazione viene calcolata sui dati tra le 15:15 e le 15:30. Le attivazioni del modulo sono temporalmente allineate a orari multipli di 15 minuti per assicurare una cadenza di calcolo più deterministica.
  Una caratteristica fondamentale di questo modulo è la sua capacità di colmare eventuali lacune nelle aggregazioni. Se per qualsiasi motivo (ad esempio, un riavvio del sistema) alcune aggregazioni non sono state eseguite, il modulo cerca l'ultima aggregazione completata e la utilizza come punto di partenza. Da quell'istante, calcola in successione tutti gli intervalli intermedi fino a quello massimo consentito dall'offset. Questo meccanismo garantisce che tutte le aggregazioni mancanti vengano recuperate e calcolate, mantenendo l'integrità e la completezza dei dati storici.
- **Modulo Dispatcher**: Componente centrale del pattern Transactional Outbox, il Dispatcher ha il compito di inoltrare i dati al livello superiore, l'Intermediate Hub. Periodicamente, il servizio preleva in blocco i messaggi con stato *"pending"* dal database locale e tenta di inviarli al broker Kafka. Una volta che l'invio è riuscito, lo stato dei record nel database viene aggiornato a *"sent"*. Questo meccanismo garantisce che, anche in caso di disconnessione della macrozona dalla rete regionale (che potrebbe sfruttare la rete internet), i dati non vengano persi e possano essere inviati non appena la connessione viene ripristinata.
- **Modulo Cleaner**: Questo servizio si occupa della manutenzione della cache. La sua funzione principale è eliminare i messaggi che sono stati inviati (ovvero con stato *"sent"*) e che superano un determinato periodo di conservazione, per evitare che i dati rimangano nella cache troppo a lungo. Il database stesso è configurato con un retention time massimo di 2 giorni, fungendo da rete di sicurezza per scartare anche i messaggi che non riescono a essere inviati per un periodo prolungato.
- **Modulo Proxy Heartbeat**: Agendo come proxy, questo modulo raccoglie i messaggi di heartbeat dagli Edge Hub tramite MQTT e li inoltra su Kafka, che garantisce persistenza e affidabilità nella comunicazione con l'Intermediate Hub.
- **Modulo Proxy Configuration**: Similmente al proxy degli heartbeat, questo modulo raccoglie i messaggi di registrazione e configurazione degli Edge Hub. Anche in questo caso, i messaggi sono retained, ma il Proximity Hub si occupa di cancellarli dal topic MQTT una volta che sono stati processati e inoltrati a Kafka, mantenendo pulito lo spazio del broker.

D. Intermediate Hub

L'Intermediate Hub (o Region Hub) è il componente responsabile di garantire la persistenza a lungo termine dei dati raccolti a livello di regione. Funge da punto di convergenza per i dati elaborati dai diversi Proximity Hub, gestendone l'aggregazione finale e il salvataggio in un database dedicato.

Questo hub è composto da una serie di servizi che operano in parallelo per ottimizzare le prestazioni e garantire l'affidabilità del sistema. L'elaborazione e il salvataggio dei dati sono eseguiti in batch, un approccio scelto per ridurre drasticamente l'overhead di I/O e migliorare l'efficienza complessiva.

- **Servizi di Data Ingestion**: Questi servizi leggono i messaggi da Kafka e li salvano nel database a lungo termine. Per gestire la mole di dati in modo efficiente, i messaggi non vengono salvati singolarmente, ma vengono prima raccolti in un batch in memoria. Il salvataggio nel database avviene in blocco, utilizzando la funzione *COPY FROM* di PostgreSQL, ottimizzata proprio per questo scenario. Il sistema comunica con Kafka con una semantica at-least-once, e la gestione dei duplicati è affidata alla logica di inserimento nel database, che previene l'inserimento di record identici. L'offset di lettura su Kafka è gestito manualmente anziché automaticamente. Questo approccio garantisce che i messaggi siano segnati come letti su Kafka solo dopo il successo del salvataggio nel database. In caso di guasto del servizio, Kafka ritrasmetterà i messaggi al prossimo reader pronto, garantendo che nessun dato venga perso.
  La gestione del traffico è fondamentale per mantenere la stabilità del sistema. I messaggi sono distribuiti in modo efficiente tra i vari servizi di ingestion tramite i Kafka Consumer Group. Tutti i servizi dell'Intermediate Hub sono associati allo stesso gruppo, permettendo una distribuzione bilanciata del carico. Inoltre, i messaggi vengono salvati su topic separati, con la chiave di partizione basata sul nome della macrozona di riferimento, il che ottimizza ulteriormente il flusso di dati. Per evitare che il salvataggio in blocco, che è più lento della lettura, intasi i canali di comunicazione, la lettura da Kafka viene temporaneamente sospesa durante l'operazione di scrittura sul database. Questo previene la perdita di un numero eccessivo di messaggi a causa di canali pieni, garantendo che le perdite in fase di concorrenza siano trascurabili.
- **Aggregazione Statistica**: L'ultimo servizio di questo hub si occupa del calcolo delle statistiche

a livello di regione. Questo processo sfrutta i dati pre-elaborati dai livelli inferiori, come le statistiche di macrozona, per evitare calcoli ridondanti. Ad esempio, per calcolare la media regionale, il servizio non rilegge tutti i dati grezzi, ma utilizza valori intermedi come la somma grezza delle rilevazioni, che vengono mantenuti nei record delle statistiche di macrozona e zona proprio a questo scopo.

- **Aggiornamento dell'ultima comunicazione**: Contestualmente all'inserimento del batch di dati, il sistema aggiorna anche il timestamp dell'ultima comunicazione dei sensori che hanno inviato messaggi all'interno del batch. Nel caso di messaggi di registrazione duplicati, il sistema aggiorna semplicemente il campo relativo all'ultima comunicazione, mantenendo il registro dei metadati sempre aggiornato.

E. *Cloud: Servizi Serverless e API*

Il livello superiore del Compute Continuum è stato implementato interamente nel cloud per garantire scalabilità, affidabilità e accesso globale ai dati aggregati. La scelta di un'architettura serverless si basa sull'utilizzo di AWS Lambda mentre le API per l'interazione con il sistema sono esposte tramite AWS API Gateway, che funge da front-end e da gestore delle richieste HTTP in ingresso.

Queste API permettono ai client di accedere ai dati aggregati e alle statistiche generate dai livelli inferiori dell'architettura. Nello specifico, tali API sono consumate sia da un sito web sviluppato in React, che fornisce un'interfaccia utente grafica per la lettura agevole dei valori, sia da un client minimale da terminale per l'accesso diretto ai dati.

Le funzioni Lambda, attivate da chiamate a endpoint specifici, interagiscono direttamente con i database regionali di persistenza, che sono separati in base alla loro funzione: il database PostgreSQL_sensors per i dati di misurazione e PostgreSQL_metadata per i metadati dei sensori e la loro configurazione. La logica di business implementata nelle funzioni Lambda è ottimizzata per eseguire query complesse in modo efficiente e fornire risposte in formato JSON.

F. *Discovery*

Un aspetto critico dell'architettura è la gestione della connettività tra i vari livelli, che deve avvenire in modo autonomo e dinamico. Questo è stato realizzato sfruttando un meccanismo di risoluzione dei nomi basato su DNS. Ogni risorsa di rete condivisa (database, broker, etc.) viene identificata tramite un nome simbolico (es. misurament-db.region-001.sensorc-ontinuum.it). Il sistema è in grado di calcolare il nome corretto di una risorsa a partire dalla sua posizione gerarchica (regione, macrozona, zona), consentendo ai componenti di connettersi autonomamente ai nodi pertinenti.

Questo approccio elimina la necessità di configurazioni rigide e statiche, supportando pienamente la natura distribuita e scalabile del sistema.

V. RISULTATI SPERIMENTALI

Per convalidare il corretto funzionamento e valutare le metriche prestazionali del sistema, è stata condotta un'ampia simulazione sfruttando i servizi cloud di *AWS*. La simulazione è stata eseguita in un ambiente *Learner Lab*, che ha imposto alcune limitazioni di deployment rispetto all'architettura ideale (come la necessità di co-locazione di più servizi sulla stessa istanza EC2: ad esempio Edge Hub e Sensor Agent o Broker MQTT e Proximity Hub).

Un aspetto cruciale è che le istanze EC2 sono state appositamente configurate per simulare ritardi di rete e perdite di pacchetti durante la comunicazione, replicando le condizioni non ideali tipiche di un ambiente Fog o Edge. Nonostante questi vincoli e le simulazioni di guasto, è stata mantenuta la fedeltà funzionale dell'architettura gerarchica per le analisi quantitative.

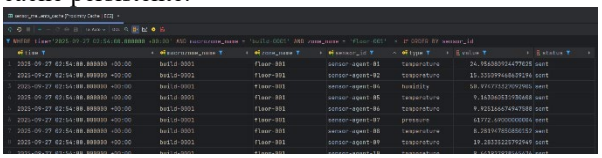
La configurazione del sistema simulato è stata la seguente: una Regione composta da due macrozone, ciascuna delle quali contenente due zone. Ogni zona era popolata da 50 Sensor Agent. Il modello di deployment ha seguito la logica definita in [III.E], con gli adattamenti necessari dovuti alle limitazioni imposte.

A. *Convalida Funzionale del Flusso Dati*

L'analisi iniziale si è concentrata sulla verifica della corretta elaborazione dei dati lungo l'intera catena del continuum.

Il log seguente dimostra la corretta connessione e l'inizio della trasmissione dei dati da parte di un agente specifico (e.g., *sensor-agent-01* di *build-0001* nel *floor-001*). Questi dati vengono successivamente ricevuti e processati dall'Edge Hub di competenza.

Una volta completato il filtraggio degli outlier e l'aggregazione temporale, l'Edge Hub inoltra i dati aggregati al Proximity Hub, il quale li salva nella propria cache persistente.



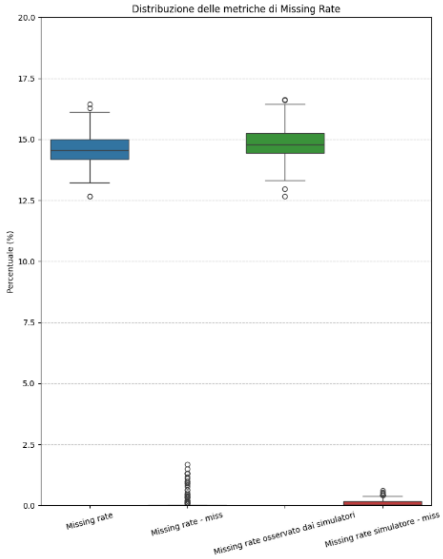
Infine, il processo di dispatching del Proximity Hub invia i dati all'Intermediate Hub, il quale li salva nel database delle misurazioni regionale, confermando che il sistema elabora e rende persistenti correttamente i dati immessi dai sensori.

B. *Analisi delle Prestazioni e Affidabilità*

Sono state condotte analisi approfondite sulle prestazioni del sistema, focalizzandosi su metriche chiave che ne definiscono l'affidabilità e la capacità di gestione del carico.

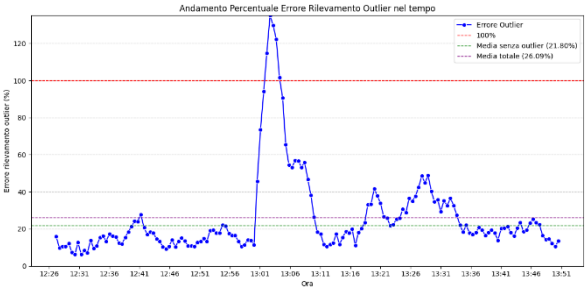
1) Miss Rate dei Pacchetti

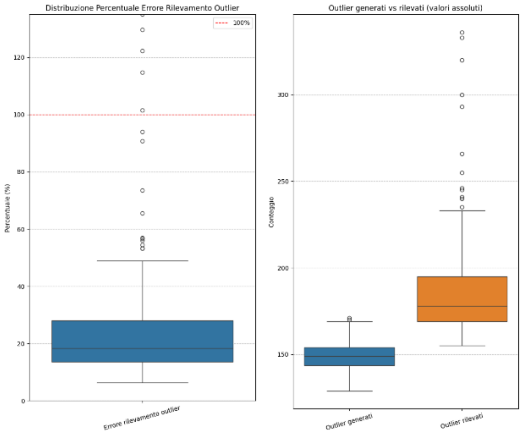
È stato calcolato il Miss Rate dei pacchetti, definito come la percentuale di pacchetti generati dai sensori che non raggiungono l'Edge Hub. Questo rate è stato determinato confrontando il numero atteso di pacchetti inviati (sulla base dei parametri di simulazione, che includono una probabilità intrinseca del 15% che il sensore non invii un pacchetto) e il numero reale di pacchetti inviati con il numero di pacchetti effettivamente ricevuti. L'analisi mostra la dispersione e la tendenza centrale del tasso di perdita nel livello più basso di comunicazione.



2) Accuratezza del Rilevamento Outlier

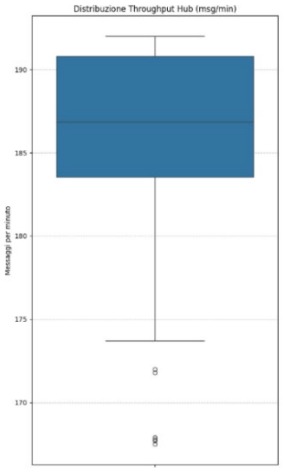
È stato valutato l'errore percentuale commesso dagli hub nel rilevamento degli outlier. Questo errore è calcolato confrontando gli outlier identificati dal sistema con gli outlier introdotti intenzionalmente dal simulatore. L'analisi è stata presentata sia tramite un box plot per valutare la distribuzione dell'errore, sia attraverso l'osservazione dell'andamento dell'errore nel tempo, al fine di carpirne eventuali stabilità o instabilità operative a lungo termine.



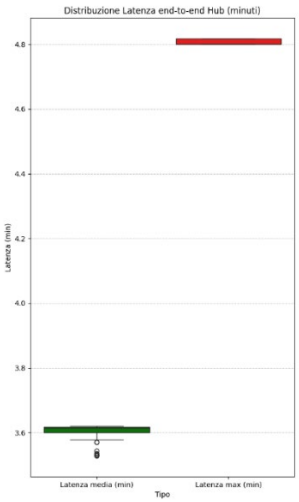


3) Throughput e Latenza End-to-End

Per osservare la reale capacità di gestione dei messaggi in arrivo da parte del sistema, è stato condotto uno studio sul throughput complessivo.



Inoltre, dato che il sistema introduce sistematicamente ritardi (mediante offset temporali) per gestire in modo robusto i messaggi con elevata latenza, è stata analizzata la latenza end-to-end del messaggio. Questa metrica misura il tempo trascorso dal momento in cui il pacchetto viene generato dal Sensor Agent fino al momento in cui viene inserito nel batch per il salvataggio nel database della regione, fornendo un'indicazione chiara del ritardo operativo intrinseco al sistema.



VI. Discussione dei Risultati

L'analisi dei risultati sperimentali mira a convalidare l'efficacia delle scelte architetturali adottate in termini di affidabilità, accuratezza e capacità di gestione del carico, come misurato dalle metriche di miss rate, rilevamento outlier, throughput e latenza.

A. Affidabilità del Flusso Dati: Miss Rate

L'analisi del Miss Rate dei pacchetti generati dai Sensor Agent e diretti all'hub di Zona ha mostrato valori osservati che si attestano stabilmente attorno al 15%. Tale risultato è del tutto auspicabile e in linea con i parametri della simulazione, che introducevano artificialmente una probabilità del 15% di non generare o perdere un pacchetto.

Non considerando questa percentuale intrinseca di pacchetti simulati come persi, i valori del miss rate (atteso e reale) si attestano attorno allo 0%, con oscillazioni minime. Ciò dimostra l'elevata affidabilità del meccanismo di consegna al primo livello del Compute Continuum e conferma che le perdite di pacchetto non attribuibili alla simulazione sono trascurabili.

B. Accuratezza e Adattabilità nel Rilevamento Outlier

L'osservazione del rilevamento degli outlier fornisce indicazioni cruciali sulla robustezza del filtraggio dei dati. Il box plot indica che l'hub tende a sovrastimare il numero di outlier di circa il 20%, mostrando un risultato accettabile per un sistema edge che privilegia la cautela (scartare dati potenzialmente validi) rispetto all'introduzione di valori anomali nel sistema.

Tuttavia, il box plot evidenzia la presenza di valori anomali che possono essere spiegati analizzando l'andamento dell'errore nel tempo. La media dell'errore si attesta a circa il 21.80%; l'inclusione degli outlier sposta tale media al 26.09%. I picchi di errore si osservano in corrispondenza del cambio dell'ora. Questi picchi sono dovuti alla natura del simulatore, che ricalcola i parametri di media e deviazione standard per i sensori su intervalli orari.

Dopo il ricalcolo, il sistema di filtraggio dell'hub scarta un numero inizialmente maggiore di pacchetti per poi tornare progressivamente alla stabilità precedente. Questo comportamento dimostra un carattere adattivo del meccanismo di outlier detection, in grado di rispondere correttamente a cambiamenti graduali, come ci si aspetterebbe in un contesto reale in cui le condizioni ambientali evolvono lentamente.

La metrica complessiva indica un comportamento accettabile, con il sistema che scarta un pacchetto in modo errato circa una volta ogni cinque outlier correttamente identificati.

C. Performance del Sistema: Throughput e Latenza

1) Throughput

Il throughput atteso dal sistema, basato sull'attività sincrona dei 200 sensori (2 Macrozone x 2 Zone x 50 Sensori), che inviano dati in tempo reale ogni minuto, è di 200 pacchetti/minuto. Il box plot mostra che il throughput reale osservato si attesta stabilmente tra 185 e 190 pacchetti al minuto. Questa leggera discrepanza rispetto al valore atteso di 200 è giustificata da potenziali errori minimi nel calcolo della metrica e dalla natura stocastica del Sensor Agent (non tutti i sensori inviano dati validi in ogni intervallo), evidenziando comunque la capacità del sistema di gestire correttamente il carico.

2) Latenza End-to-End

La latenza end-to-end media osservata si mantiene stabilissima attorno ai 3.6 minuti. Considerando la latenza massima osservata, si può affermare che il sistema introduce una latenza sistematica, relativa al percorso del messaggio dal Sensor Agent al salvataggio nel database regionale, di circa 4 minuti. Questa latenza è il risultato intenzionale delle diverse fasi di elaborazione e buffering che garantiscono la robustezza e la coerenza temporale dei dati.

VII. Conclusioni e Lavoro Futuro

Il presente studio ha delineato e convalidato l'implementazione di un sistema di sensori distribuito su larga scala che sfrutta in modo efficace il paradigma del Compute Continuum. L'architettura a cinque livelli, basata su Go routines, microservizi e l'integrazione di broker per la comunicazione, ha dimostrato di raggiungere gli obiettivi di scalabilità e resilienza richiesti, confermati dai risultati sperimentali.

Come prospettive per il lavoro futuro, si possono individuare diverse aree di miglioramento e sviluppo:

- **Rafforzamento dell'Intelligenza all'Edge**: È possibile migliorare l'accuratezza del rilevamento outlier integrando algoritmi di Machine Learning leggero direttamente negli Edge Hub. Questo permetterebbe di affinare la logica di filtraggio e ridurre la dipendenza dai parametri statistici.
- **Monitoraggio Proattivo**: È possibile implementare il meccanismo di notifica via email per i manutentori, previsto per i sensori etichettati come unhealthy, al fine di migliorare i tempi di reazione del sistema in caso di guasti.
- **Ottimizzazione della Latenza**: Sebbene la latenza sistematica sia intenzionale, si potrebbe studiare la possibilità di una pipeline *dati fast lane* per tipologie specifiche di misurazioni aggirando gli offset per fornire risposte con la minima latenza possibile in caso di emergenza.

VIII. Riferimenti

[1] Open Data Stuttgart, «Sensor.Comunity,» [Online]. Available: https://sensor.community/it/.

[2] F. Masci e M. Renzi, «GitHub repository containing the project code,» 2025. [Online]. Available: https://github.com/F-masci/SensorContinuum.

[3] V. Cardellini e G. Russo Russo, «Corso di Sistemi Distribuiti e Cloud Computing,» 2024. [Online]. Available: http://www.ce.uniroma2.it/courses/sdcc2425/.