Alma Mater Studiorum · Università di Bologna

SCUOLA DI SCIENZE

Corso di Laurea in Informatica per il management

SERIE TEMPORALI IOT IN CASSANDRA: MODELLAZIONE E VALUTAZIONE SPERIMENTALE

Relatore:
Chiar.mo Prof.
MARCO DI FELICE

Presentata da:
SALVATORE FIORILLA

Sessione III
Anno Accademico 2017-2018

Alla mia Famiglia...

Introduzione

Nuove tipologie di dati sono state sviluppate negli ultimi tempi; non è più solo l'uomo l'artefice della loro creazione, ma lo sono anche le macchine o più in generale agenti, che sulla base di percezioni agiscono autonomamente e di conseguenza. Oggi archiviare dati sembra una soluzione trovata ormai da molto tempo, ma con la generazione continua di enormi quantità di dati eterogenei, sono necessari nuovi strumenti e tecniche di storage per evitare che queste informazioni possano andare perse insieme al loro valore. Una corretta gestione della archiviazione progettata sin dalla prima fase del ciclo di vita dei dati, di sicuro può evitarlo. L'internet of things, ad esempio è una fonte constante che produce serie di dati temporali. A fronte di questi problemi, occorrono nuove tecniche di gestione della loro archiviazione. Una buona soluzione in termini di budget, sembra arrivare dalle alternative basate su software open source e comodity hardware. Il lavoro svolto in questo progetto, ha prodotto una architettura di gestione dell'archiviazione composta da un set di software open source integrati fra di loro. L'architettura è in grado di archiviare dati da qualsiasi tipo di oggetto iot e può essere usata per sviluppare una applicazione web. Successivamente si è progettato, sviluppato e testato sperimentalmente un caso pratico in cui i dati sono stati rilevati da una rete ZigBee domestica di sensori, creata per misurare temperatura ed umidità. Si propone e di seguito, infine, una logica di modellazione dei dati di serie temporali in Cassandra implementata e testata.

Obiettivo

A conclusione di questo lavoro verranno mostrati i risultati dei test sperimentali di una base di dati distribuita su un cluster di quattro macchine virtuali, che è stata utilizzata per monitorare la temperatura e l'umidità del laboratorio Ranzani, del dipartimento DISI dell'università di Bologna. La base di dati creata fa parte di un architettura software in grado di gestire l'archiviazione e di il recupero dei dati provenienti da ogni tipo di oggetto iot. Per arrivare ad ottenere questo risultato, è stato svolto in una prima fase del lavoro, uno studio sulle tecniche di archiviazione distribuite e delle soluzioni proposte dalle tecnologie open source per l'archiviazione di dati di serie temporali; successivamente, si è approfondito lo studio sul database non relazionale Cassandra, tecnologia da utilizzare come strumento di archiviazione distribuito. Questi concetti verranno maggiormente approfonditi nei paragrafi 2.2 e 4.5 A seguito di questa prima fase, si è passati ad analizzare come poter far confluire in un database distribuito in Cassandra le informazioni provenienti dai sensori; la soluzione che si è trovata aggrega due tipi di architetture software che risolvono ognuna di loro, problemi di messaggistica diversi. L'architettura publisher - subscriber e l'architettura client- server. I dettagli verranno ripresi al capitolo 2 in cui è spiegata la progettazione. Nella seconda parte del lavoro, si è applicata l'architettura ideata al caso pratico; L'architettura software implementata comprende 3 entità software: un broker mqtt per la comunicazione con gli oggetti iot, un servizio principale da cui passano tutte le richieste di lettura e scrittura per il terzo elemento dell'architettura che è il database. Proprio da quest'ultimo, si è partiti per sviluppare lo scenario ; è stato progettato un modello di archiviazione per data time series che sfrutta le funzionalità di Cassandra, successivamente, si è passati alla fase di implementazione su macchine virtuale della piattaforma cloud GARR, dove appunto è stato implementato un broker mqtt e messo online su una macchina virtuale. A seguito del implementazione del broker è stata implementata la base di dati su un database distribuito in un cluster composto da 4 elaboratori, ed infine è stato creato un programma in Nodejs

che collega le due entità software e fornisce un interfaccia di visualizzazione dei dati. Durante la fase di progettazione ed implementazione l'architettura è stata integrata con una rete ZigBee di sensori che monitoravano la temperatura del laboratorio. La comunicazione fra il broker mqtt dell'architettura e la rete di sensori è avvenuta tramite un microcontrollore ESP8266. Inoltre, per migliorare ed ottimizzare lo spazio di archiviazione dei dati, sono stati creati programmi background che diminuiscono la dimensione della base di dati con tecniche di downsampling. Questo ambiente di test può facilmente essere esteso ad ampi scenari e contesti per merito della sua architettura. I risultati dei test sperimentali sul database creato si trovano al capitolo 4.

Struttura tesi

La tesi è suddivisa in 5 capitoli. Dopo l'introduzione vi è il primo capitolo che discute lo stato dell'arte. Nel secondo capitolo viene mostrato al lettore la progettazione svolta mentre sul terzo si parla dell'implementazione della architettura. Il capitolo quattro mostra i test svolti ed i risultati sperimentali ottenuti relativi al caso di studio e al capitolo cinque infine si conclude parlando degli sviluppi futuri.

Indice

Elenco delle figure

Capitolo 1

Stato dell'arte

1.1 Iot

Con il termine Internet of Things ci si riferisce ad oggetti che fanno uso di reti per comunicare tra di loro. Gli oggetti non sono solo strumenti di comunicazione, ma sono anche in grado di prendere decisioni autonomamente sulla base di dati che rilevano o ricevono, ed per questo sono detti oggetti "intelligenti". Secondo Business Insider Intelligence, il servizio di ricerca premium di Business Insider, nel libro The Iot Forecast Group 2018[1], si prevede che nei successivi 5 anni, quindi entro il 2023, si installeranno più di 40 miliardi di dispositivi IoT a livello globale. Ancora, secondo i calcoli di IDC (International Data Corporation), riportati sul sito web www.internet4things.it [2], la spesa mondiale relativa all'Internet of Things raggiungerà i $745 miliardi dollari nel 2019, con un aumento del 15,4% rispetto al $646 miliardi spesi nel 2018. IDC, si aspetta che la spesa in tutto il mondo manterrà un tasso di crescita annuale a due cifre in tutto il periodo di previsione 2017-2022 fino a superare $1 trilione nel 2022. Alcuni gli ambiti di utilizzono dell'iot oggi sono visibili in Figura 1.1. Trasporti, industrie, mercati, scuola, veicoli, domotica, Natura (sopra tutti Agricoltura), e Salute.

Vari esempi, nel loro utilizzo :

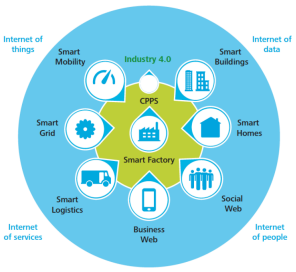


Figura 1.1: Ambienti d'uso dell'IoT

- Rilevazione dei livelli di spazzatura nei contenitori per ottimizzare i percorsi di raccolta dei rifiuti.

- Rilevazione dei livelli di spazzatura nei contenitori per ottimizzare i percorsi di raccolta dei rifiuti.

- Illuminazione intelligente e resistente alle intemperie nelle luci stradali.

- Autostrade intelligenti con messaggi di avvertimento e deviazioni in base alle condizioni climatiche e ad eventi imprevisti come incidenti o ingorghi.

- In agricoltura per esempio è possibile monitorare l'umidità del terreno e il diametro del tronco nei vigneti per controllare la quantità di zucchero nell'uva e la salute della vite.

- Ricerca di singoli oggetti in grandi superfici come magazzini o porti.

Figura 1.2: IoT elements

Figura 1.3: a sinistra un sensore di temperature, a destra una board

- Controllo della temperatura all'interno di frigoriferi industriali e medicali con merce sensibile.
- Monitoraggio dei livelli di gas tossici e ossigeno all'interno degli impianti chimici per garantire la sicurezza dei lavoratori e delle merci.

Un oggetto smart, è un oggetto che si distingue dagli altri per poter essere identificato, poter avere capacità di connessione, di localizzazione e di elaborazione dati, ed inoltre per essere in grado di interagire con l'ambiente esterno e prendere decisioni intelligenti.

Alla figura 1.2 vengono mostrati i sei elementi principali necessari e sufficienti affinchè un oggetto possa fornire le funzionalità dell'iot, invece alla figura 1.3, è visibile un esempio di sensore di temperatura e di una borad a cui viene collegato.

1.1.1 Architettura publisher subscriber

Negli ambienti IoT si tende a risparmiare le risorse a disposizione del dispositivo, per questo motivo, una connessione del tipo one-to-many, dove un singolo ente deve mantenere una connesione con molti altri, risulta molto costosa ed inefficiente. Si preferisce, piuttosto, ottenere lo stesso risultato ma attraverso una comunicazione indiretta fra i due oggetti. L'architettura publisher - subscriber è un pattern architetturale di comunicazione adatto a risolve questo problema, ed il protocollo MQTT è il protocollo standard che

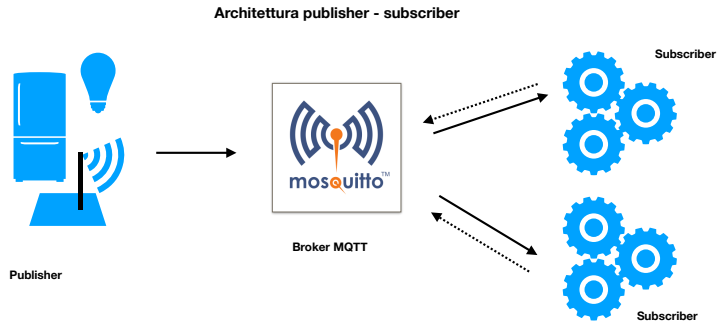


Figura 1.4: Architettura publisher subscribe

lo implementa fra dispositivi iot. I software che compongono l'architettura publisher subscriber sono tre, due clients ed un broker: Lo scopo è far comunicare i due client indirettamente, ovvero, passando attraverso il broker, il quale, con messaggi opportuni cerca di mantenere il più allungo possibile la connesione fra esso ed i client. I clients svolgono due ruoli distinti : il ruolo di publisher e quello di subscriber. Il subscriber si occupa di sottoscriversi ad un argomento e, in un secondo momento, quando un publisher invia dati sull'argomento al broker, quest'ultimo filtrerà i dati a tutti i sottoscrittori di quell'argomento, e quindi il subscriber riceverà i dati. Il publihser si occupa di produrre i dati ed inviarli ad un broker, definendo per ogni invio un argomento. Il broker, sta in mezzo, quando riceve i dati degli argomenti dai publisher li filtra a tutti i clients sottoscritti allo stesso topic. Un clients può svolgere entrambi i ruoli contemporaneamente. Mosquitto è un programma open source che implementa il ruolo di broker secondo lo standard del protocollo MQTT. Il protocollo MQTT garantisce 3 livelli di comunicazione delle informazioni, dette anche Quality of Service, a seconda del livello scelto può essere mantenuta o no una certa qualità del servizio di comunicazione fra i due. La qualità 0 non garantisce nessuna certezza che un messaggio arrivi al destinatario, per questo è anche detta "fire and forget", La qualità uno, invece assicura almeno una ricezione dell'informazione, ed infine, il servizio di qualità due, assicura il grado più alto fra i tre livelli di qualità, e cioè, si assicura che il il messaggio arrivi a destinazione e che non vi siano ripetizioni.

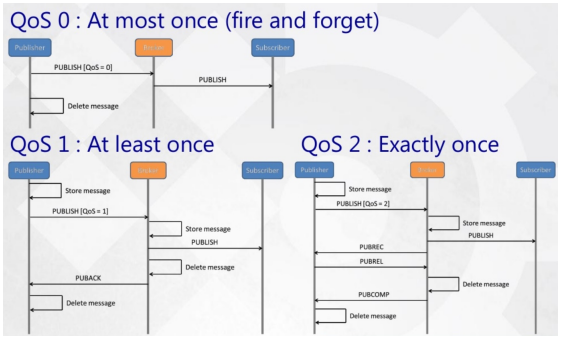


Figura 1.5: Diagramma di sequenza dei QoS del protrocollo MQTT

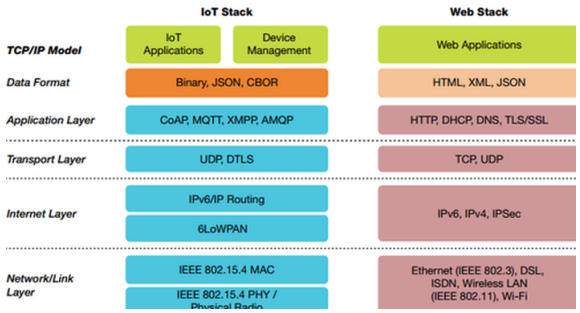


Figura 1.6: Stack protocolli di rete Mqtt e Http

In figura 1.5 vengono mostrati i diagrammi di sequenza delle 3 qualità del servizio:

1.2 Big Data

Con il termine BigData si intende un enorme mole di dati che eccede per volume,varietà e velocità. Secondo quanto riporta wikipedia le tre caratteristiche sono state attribuite per la prima volta in una analisi di Douglas Laney, vice presidente di Garther. E tutte meritano un approfondimento.
Il primo aspetto che caratterizza i Big Data, il volume, è indefinito; non si riesce più a stimare quale sia l'ammontare di dati attualmente nel mondo. Per dare un idea, c'è un analisi presentata nel 2012 fatta da IDC in cui si stima che dal 2005 al 2020, l'universo digitale sarebbe cresciuto di un fattore

di 300, e si sarebbe passato da 130 exabyte a 40.000 exabyte, o 40 trilioni di gigabyte (questo voleva dire che nel 2012 avremmo avuto più di 5.200 gigabyte per ogni uomo, donna e bambino). Bisogna comunque considerare che, per una azienda, o per chi elabora i dati scentifici come gli istituti di ricerca, l'ordine dei dati è quello dei terabytes o al massimo dei petabytes. Molto lontano dagli exabytes.
La seconda caratteristica dei BigData è la varietà di formato e di struttura dei dati. Classificando per varietà in struttura si può fare una distinzione fra: dati strutturati, semi e non strutturati. Per diversi anni si è analizzato solo il primo tipo, in quanto il più facile da archiviare all'interno dei database relazionali. Sono dati strutturati tutti quei dati che presentano una struttura fissa ed è raro che vi siano in essa cambiamenti o valori mancanti. Sono, invece, dati semi-strutturati i dati con una struttura irregolare e/o parziale. La struttura di una pagina web, per esempio può includere più o meno informazioni e ogni volta la stessa tipologia di informazione può essere sempre presente o no. Sono infine dati senza struttura, quei dati dove non esiste uno schema per sistemarli, per esempio, i file multimediali oppure i file di testo che contengono solo frasi in linguaggio naturale. La disciplina che studia come recuperare questo tipo di file si chiama Information Retrival.
Per velocità, considerata terza caratteristica, si intende la velocità di due processi : la rapidità con cui i dati vengono generati e la celerità che serve è ad interpretarli. IDC, sempre nella stessa analisi ha stimato anche che: il 23% delle informazioni nell'universo digitale (o 643 exabytes) sarebbe utile per i Big Data se fosse etichettato e analizzato. Tuttavia, la tecnologia è lontana da dove deve essere, e in pratica, ritengono, nello loro previsioni, che solo il 3% dei dati potenzialmente utili sia etichettato, e ancor meno viene analizzato. Successivamente, da poco, sono state aggiunte altre 2 caratteristiche ai Big Data, veridicità e valore.
Per valore si intende la capacità che un dato ha di generale del valore a seguito dei processi di analisi. Si parla di veridicità, invece, riferendosi all'affidabilità dei dati perché questi ultimi possono andare facilmente persi

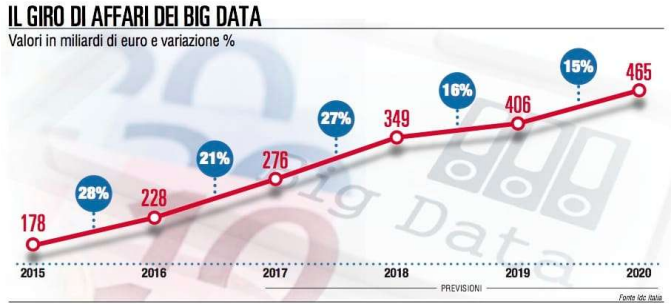


Figura 1.7: fonte: sole 24 ore

o essere archiviati con errori. Non si riesce a dare dei numeri in merito a questi dati con gli strumenti che abbiamo oggi, quindi la definizione di big data rimane astratta. Non c'è un momento in cui i dati diventano troppo grandi, troppo variabili o troppo veloci.Lo scopo finale dei big data è l'analisi di essi per creare un valore aggiunto. Visto che, si hanno diverse tipologie di analisi con diversi gradi di precisione, di errore e di affidabilità per cui anche la veridicità ed il valore di un dato non hanno limiti ben precisi.

Utilizzo e opportunità

In figura 1.7 viene mostrato un grafico che riportato da un articolo de il sole 24 ore in cui vengono illustratati i numeri di mercato di un analisi sui big data fatta da IDC, come è leggibile nell'immagine. Sono tanti i benefici nell'utilizzo dei BigData e le opportunità che essi apportano. Per fare un analisi generale si potrebbe guardare tre aspetti :

Il Business

dai big data possono nascere nuovi modelli di business o migliorare quelli attuali.

La tecnologia

negli ultimi anni si sono sviluppate nuove tecnologie per adeguarsi a gestire il volume la variabilità e la velocità dei dati e ognuna di queste è innovativa a modo suo.

L'aspetto finanziario:

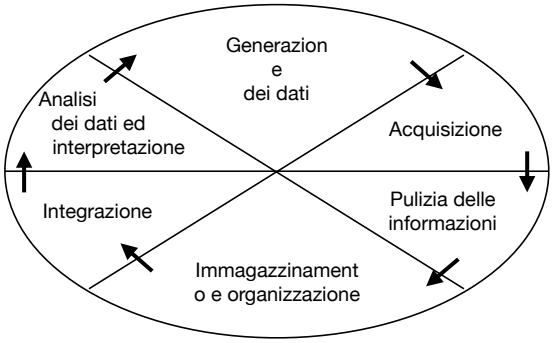


Figura 1.8: ciclo di vita dei BigData

in diversi scenari d'uso i big data hanno apportato vantaggi economici alle aziende che hanno adottato soluzioni alternative.

Ciclo di vita dei BigData

In figura 1.8 viene mostrato il ciclo di vita dei big data.
Le fasi del ciclo di vita dei BigData sono 2: la prima fase riguarda la gestione dei dati e comprende 4 processi: Generazione dei dati, acquisizione, pulizia delle informazioni ed infine immagazzinamento e organizzazione di questi ultimi. La seconda fase invece, riguarda l'analisi dei set dei dati, ed è composta da due processi: integrazione prima ed analisi poi.

Generazione dei dati

In questa fase i dati vengono generati ed è qui che si determina la loro struttura. Sono parecchie le fonti di generazionie dei dati e ad esempio, possiamo trovare i file statici generati dalle applicazione, i dati originati in tempo reale, oppure, non bisogna dimenticare dei dati generati da altri archivi di dati.
Facendo una classificazione per tipo di fonte, si possono distinguere tre tipi di sorgenti: dati generati dalle macchine, dati generati dagli umani e dati generati dal business.
Le sorgenti macchine principali sono i dispositivi del mondo IOT, e la categoria spazia dai frigoriferi ai sensori di movimento, RFID, GPS, sistemi High-FrequencyTrading dei mercati finaziari, dispositivi biomedicali ecc. L'uomo

invece, è fonte dei big data soprattutto attraverso il web: l'uso dei social network, dei siti di e-commerce, le wiki, i siti di forum come YahooAnswer, ad esempio.
Infine, per dati generati dal business si intendono tutti quei dati prodotti da una azienda nell'esercizio delle sue funzioni attive. La maggior parte di questo tipo di dati sono dati storici, per esempio la lista di tutti gli ordini avvenuti dentro l'azienda.

Acquisizione

L'acquisizione dei Big Data da questi canali d'informazione può avvenire con diverse modalità:

- Attraverso API messe a disposizione dai servizi Web, grazie alle quali è possibile interfacciarsi ad essi per esaminarne i contenuti. Un esempio sono le Twitter API, Facebook Graph API e le API fornite da motori di ricerca come Google e Bing;

- Utilizzando software di web scraping , cioè software che estraggono e raccolgono automaticamente dati da i siti web o documenti presenti su internet. Un esempio è il framework Apache Tika.

- Importando i dati da una sorgente qualsiasi in un Datawhareous usando strumenti ETL.
  Il DataWharehouse è una collezione di dati in cui i dati vengono inseriti prima di eseguire la fase di analisi. Questi database sono creati a posta per agevolare le prestazioni dell'analisi.
  Il termine ETL vuol dire, invece, Estrazione, Trasformazione e Caricamento dei dati. Sono 3 fasi del processo utilizzato per spostare i dati da sorgenti qualsiasi (relazionali o non) su DataWharehouse. Lo strumeto ETL più famoso è senza dubbio Apache Sqoop.

- I dati possono arrivare pure sottoforma di un flusso continuo e costante. Il lavoro di archiviazione viene fatto da sistemi capaci di : catturare eventi, elaborare e salvare il dato in modo efficiente. Tra le

teconologie più diffuse ci sono Apache Flume, Apache Kafka e Microsoft StreamInsight.

Elaborazione dei dati e pulizia delle informazioni

I dati non sempre sono disponibili nel formato richiesto per l'archiviazione. In questa fase si utilizzano tecniche di pulizia dei dati che applicano vincoli sui di essi e vi controllano la validità.
Questa fase è strettamente legata al formato del dato che si utilizza. Se si sta utilizzando un sensore che produce dati, e quindi si tratta di dati molto strutturati, questa fase potrebbe essere un semplice parse di rappresentazioni di valori. Se invece si stanno utilizzando i dati dei social network e quindi dati semi - strutturati o ancor peggio non strutturati, l'analisi all'idoneità diventa un processo più complesso.

Immagazzinamento ed organizzazione

In questa fase i dati vengono sistemati in memoria (Processo di immagazzinamento,) ed eventualmente (Processo di organizzazione), possono subire ulteriori elaborazioni e trasformazioni in preparazione alla fase successiva di analisi.
Il primo processo deve essere in grado di poter memorizzare volumi di dati ampi e garantire la disponibilità continua dei sistemi, fornendo servizi altamente affidabili. Per questo motivo, sono stati sviluppati software per architetture che propongono di sfruttare il vantaggio del calcolo distribuito. Però, è necessario rendere trasparente la distribuizione di risorse fra più elaboratori e, per l'accesso, il recupero e la memorizzazione dei file su più macchine, vengono usati i file system distribuiti, invece, per i sistemi di organizzazione logica dei dati e per funzionalità come, per esempio permettere il recupero di un record in tempi rapidi, occorrono database distribuiti.
I File system più conosciuti sono: Google File System (GFS) e Hadoop Distributed File System (HDFS), mentre, per quanto riguarda i database distribuiti, si utilizzano db appartenenti ad una determinata categoria di basi di dati, i sistemi NoSql. Ognuno di loro implementa una logica di archiviazione

diversa ed ogni logica ha i suoi pro ed i suoi contro. Se ne parlerà in dettaglio al paragrafo 1.4 di questo capitolo.Esempi di db NoSql sono: HBase, BigTable, Cassandra, MongoDB.

Integrazione

Arrivati a questo punto, i dati sono stati acquisiti, caricati e organizzati nelle struttura Handoop/NoSQL, è necessaria una loro preparazione alla fase finale e, per questo motivo, se necessario, si svolgono operazioni di integrazione con altri dati di altri database. Superata questa fase, i dati possono trovarsi in formati diversi, per esempio un documento word, uno Excel, uno XML.
Il passo successivo è quello di dare lo stesso formato a tutti i dati da analizzare effettuando su di loro il processo di trasformazione. I software più famosi adatti a svolgere queste attività sono: Sqoop per l'integrazione, Apache Tika per trattare formati differenti in modo uniforme e Hive per aggregare dati, eseguire query e analizzare grandi dataset su Handoop.

Analisi dei dati ed interpretazione

Nell'ultima fase del ciclo di vita viene svolta l'analisi dei dati allo scopo di ottenere un informazionei utili. Vari framework sono di supporto all'interrogazioni dei dati e questi si differenziano per tipo di analisi: Apache Mahout, per esempio, è una piattaforma su cui poter fare analisi complesse interrogando il database distribuito.
Uno strumento ideale per fare analisi più semplici, invece, è Pig, il quale mette a disposizione un linguaggio, Pig Lating, per interrogare HDFS.
Se si vuole interrogare il database distribuito, esistono linguaggi appositi creati per i database, molto spesso si tratta di linguaggi SQL-like. Infine si cita anche R, un software di calcolo statistico dotato di un linguaggio proprio. Con questo linguaggio attraverso l'integrazione con appositi framework è possibile analizzare i dati prelevandoli dal database distribuito o dal file system.

1.3 Il Cloud Computing

Con il termine Cloud Computing si intende la distribuzione, di risorse hardware e software come servizio remoto, da parte di un fornitore nei confronti di un cliente. L'ente fornitore del servizio cloud si chiama Internet Service Provider (ISP), ed un cliente può essere un ente giuridico qualsiasi.

Esistono 3 tipologie di servizi concessi. Viene fatta una classificazione dall'alto verso il basso fra tre tipologie di servizi:

SaaS - Software as a Service;

PaaS - Platform as a Service;

IaaS - Infrastructure as a Service;

Con il servizio SaaS, è concesso al cliente la possibilità di poter utilizzare un software via remoto. Il software non risiede nel computer cliente ma è installato su terze parti e vi si può accere via web. Normalmente il cliente paga il servizio con un abbonamento mensile e vi accede tramite log-in., in questo modo non deve nemmeno preoccuparsi della manutenzione del programma.

PaaS è il servizio tramite cui l' ISP mette a disposizione del cliente un ambiente di sviluppo software. Il cliente paga per poter usare una piattaforma che contiene diversi servizi, come per esempio librerie, necessari al cliente per poter sviluppare il software.

Il servizio di Infrastructure as a Service consiste nel fornire al cliente un infrastruttura fatta da risorse hardware virtuali: per sempio cpu, capacità di rete, sistemi di memoria, Firewall, memoria Ram e memoria di archiviazione persistente. Viene dato al cliente la possibilità di poter creare istanze di macchine virtuali e di configurare l'intera infrastruttura partendo da zero.
I vantaggi del cloud sono:

1. Non occorre esegure operazioni di manutenzione hardware o software

2. Non occorre sostenere costi iniziali elevati per l'impatto della soluzione.

Normalmente il metodo di pagamento più usato è pay-per-use.

3. La scalabilità. Ovvero la capacità di un sistema di poter aumentare o diminuire di scala in funzione delle proprie necessità di utilizzo

Gli svantaggi sono :

1. I dati vengono affidati a terzi per cui occorre prendere le precauzioni necessarie.

2. Spesso il porting, (cioè il passaggio dall'ambiente di storage locale a quello su cloud) di grosse quantità di dati comporta tempi lunghissimi a causa della congestione ottenuta dalla rete e occorre adottare altre soluzioni. Per esempio alcuni ISP offrono servizi di trasporto di harddisk fisici

1.4 NoSQL - not only SQL

Con il termine NoSQL si intendono tutti quei sistemi software dove la persistenza dei dati è caratterizzata dall'uso di soluzioni diverse da quelle del modello relazionale tradizionale o anche detto RDBMS.
Questi sistemi permettono la distribuzione dei dati su più macchine e sono detti schemaless (cioè senza schema) perché non si basano sul concetto matematico di relazione (o tabella,) come fanno invece i database relazionali.
I database not only sql, basano le loro soluzioni su l'utilizzo di comodity hardware e sono per la maggior parte sistemi open source. Sistemi di questo tipo inoltre presentano api semplici e non implementano le proprietà ACID delle transazioni ma sono in grado di scalare orizzontalmente.

1.4.1 Teorema Cap

Cenni sul calcolo distribuito ed alcune definizioni

Nella fase di immagazzinamento del ciclo di vita dei Big Data, si è discusso del calcolo distribuito e delle tecnologie che ne sfruttano le proprietà. Nel linguaggio informatico, il calcolo distribuito è la disciplina che studia i sistemi distribuiti, ovvero quel tipo di sistemi formati da più computer autonomi che comunicano tra loro attraverso lo scambio di messaggi e con un fine comune.
Il calcolo distribuito studia il modo in cui usare un sistema distribuito per risolvere problemi computazionali; l'approccio consiste nel dividere il problema in molti compiti, ognuno dei quali risolto da un singolo computer.
Nel sistema di gestione dell'archiviazione di valori di dati e nel recupero di essi in modo efficiente si ricorre all'uso del database distribuito (detto anche cluster), ossia un tipo di database che ha i propri archivi di dati memorizzati su più elaboratori(o nodi del cluster).

Il Teorema

Ci sono tre proprietà che un sistema distribuito dovrebbe garantire :

- Consistenza dei dati in tutti i nodi nello stesso momento
- Disponibilità, del sistema a dare una risposta, sia positiva che negativa, per ogni richiesta ricevuta.
- Tolleranza di partizione, ovvero che il sistemi continui a funzionare anche con la perdita di alcuni messaggi

Secondo quanto asserito dal teorema CAP se si distribuisce un sistema qualsiasi, prendiamo in esempio il database, si creerà un trade off fra consistenza e disponibilità. Viceversa l'unico modo per evitare la nascita di questo trade-off è quello di non partizionare il database. Date consistenza, disponibilità e partizionamento, possiamo avere due proprietà insieme, ma mai tutte e tre.
Un database può essere:

- CA : Consistente nei dati e sempre disponibile. Ma ciò implica, che il sistema non è tollerante ai guasti nella partizione o non c'è una partizione; se c'è una partizione e la si vuole rendere tollerabile, allora nasce pure un trade off fra queste due proprietà, perché una esclude l'altra.

- AP: Altamente disponibile e tollerante ai guasti, se si vuole rendere il servizio sempre disponibile alle richieste che arrivano, allora si deve poter scrivere anche se un dato non è stato precedentemente inserito. Questo provoca inconsistenza nei dati.

- CP: Consistente e tollerante ai guasti, se si vuole attendere la consistenza dei dati, e dunque che almeno un numero arbitrario di nodi abbiano replicato, allora non puà esserci disponibilità nei confronti delle richieste del client se la risposta da parte dei nodi non avviene.

Ogni sistema offre un proprio servizio: per esempio Cassandra è AP mentre MongoDB invece è CP.
Proprio a causa del trade-off appena citato non esiste un database unico.Ogni database è progettato per una particolare classe di applicazioni o per ottenere una combinazione specifica di proprietà di sistema desiderabili.

1.4.2 ACID vs BASE

Una transazione è un esecuzione di letture e scritture o di operazioni singole in un flusso di lavoro.
Per sempio: esistono due valori all'interno di un database che rappresentano rispettivamente il saldo del c/c dell'utente A ed il saldo del c/cdell'utente B. Se A effettua un bonifico a beneficio di B, Il conto di A dovrà essere stornato della stessa quota da sommare al conto di B. Se non accadesse questo si avrebbe un guasto nei dati. Si vuole che:

(D) una volta cambiati, i dati aggiornati non andranno persi.

(A) che le operazioni di somma su c/cA e differenza su c/cB siano eseguiti insieme. Se non vanno a buon fine si deve ritornare allo stato

precedente dove nessuna delle due era stata ancora eseguita.

(C) alla fine il database non deve essere compromesso.

(I) nessuna transazione che sta eseguendo nello stesso istante possa provocare calcoli sbagliati

Una transazione che gode delle proprietà ACID è in grado di assicurare la coerenza dei dati dentro un database : Atomicity - Consistency - Isolation - Durability.

Atomicità

La transazione deve essere vista come un lavoro unico anche se fatta da più "jobs". Questi lavori devono essere eseguiti tutti insieme oppure non deve restare alcun effetto sui dati in caso di fallimento.

Consistenza:

questa proprietà assicura che ogni transazione porti il database da una situazione di consistenza ad un altra sempre di consistenza, cioè dopo che una transazione viene completata il db è strutturalmente valido.

Isolamento:

Una transazione non deve subire gli effetti di altre transazioni che stanno eseguento nello stesso momento sulla stessa area di memoria (cioè concorrenti). L'esecuzione deve produrre gli stessi risultati che si avrebbero dalla produzione in serie di questi database.

Persistenza:

questa proprietà assicura che il cambiamento dei dati, dopo che è avvenuto il comando di "salvataggio" (commit), durerà nel tempo anche a seguito di un errore o una anomalia nel sistema. I dati potranno essere modificati solo a seguito di un altra transazione che va a buon fine.

I sistemi NoSQL sono sistemi distribuiti e, come è stato spiegato al paragrafo 1.4.1, essi non possono rispettare tutte le proprietà sopra. Ne segue

che: Cassandra è AID, MongoDB è CID.

Per contro, le transazioni nel mondo NoSql usano l'acronimo BA.S.E. (Basically Available, Soft state, Eventual consistency).
Un applicazione funzionerà sempre (BA), tuttavia non deve garantire la consistenza ad ogni istante(E), ma l'importante è che alla fine sia garantita.(S)
Ci sono 3 modalità di funzionamento per garantire le proprietà BASE:

Casual Consistency: per poter effettuare una modifica l'applicazione manda un messaggio alle altre sessioni contente i dati da aggiornare. Alla ricezione le altre sessioni vedranno il dato aggiornato.

Read your own writes : la sessione che riceve la proposta di modifica esegue il cambiamento su se stessa, mentre le altre potranno vedere il cambiamento inviato precedentemente con un leggero ritardo.
Monotonic consistency : viene assicurato che una sessione non vedrà mai dati più vecchi dei propri, potrà vedere i dati non ancora aggiornati degli altri ma mai quelli più vecchi.

1.4.3 Tipi di database NoSql

I database non relazionali si possono classificare per il loro modo di salvare e fornire l'accesso ai dati in: document-oriented, coppia chiave-valore oriented, wide-column oriented e graph oriented.

Il modello a chiave valore

viene spesso utilizzato per memorizzare informazioni che non presentano correlazioni, ad esempio per il salvataggio delle sessioni degli utenti sul web.

La struttura di conservazione logica dei dati è organizzata secondo un modello di due entità logiche: un valore ed una chiave ad esso associato che lo identifica univocamente. In gergo informatico questa struttura logica viene chiamata anche coppia ¡ chiave, valore¿.
Il valore può contenere sia dati elementari che dati avanzati.
I vantaggi di questo modello di dati risiedono nella sua semplicità e nella velocità dei dati sia in lettura che in scrittura. Le interrogazioni si effettuano sulle chiavi e da esse si ottiene il valore.
Lo svantaggio principale è che non permette di eseguire interrogazioni complesse, come il recupero di un intervallo di dati, i modo efficiente. A causa della natura di questa struttura, questo genere di database supporta solo quattro operazioni:

- (add) aggiuta di un elemento al database;
- (remove) eliminazione di un dato;
- (modify) modifica di un valore esistente;
- (get) recupero di un valore tramite la chiave.

Sono db key-value oriented : BerkelleyDB, Project Voldemort

Il modello document-oriented

il concetto di documento si riferisce ad insiemi di coppie chiave valore organizzati in formati semi strutturati: per esempio in formato JSON o XML. La struttura di archiviazione è costituita da una collezione di modelli a chiave-valore, e questo permette di ottenere query aggregate, permette di recuperare l'intero documento tramite la sua chiave o solo parti di esso. È anche permesso fare ricerche full-text, cioè ricerche per trovare i valori corrispondenti a parole o frasi.

Possono mappare perfettamente gli oggetti del modello OOP.
Ad esempio, supponendo che un documento abbia due nomi, un indirizzo e un elenco di età degli occupanti di una casa, un secondo documento invece, potrebbe avere quattro nomi, due indirizzi e nessuna informazione sull'età. Un database orientato ai documenti prenderà i dati in entrambi e li memorizzerà in base ad un valore univoco, per esempio il tipo del documento. In questo modo sarà in grado di gestire set di dati di lunghezza non fissa.
MongoDB, CouchDB sono un esempio di database orientato al documento.

Il modello wide-column

In un database relazionale, le tabelle memorizzano i record per riga. In questo modello invece l'organizzazione dei dati avviene per colonne. Ad ogni riga può corrispondere un insieme di colonne diverso.
Per evitare la presenta di valori di tipo null dentro un database, non esiste una chiave di colonna senza un valore corrispondente. Una memorizzazione a colonne larghe può essere rappresentata come una mappa multilivello ordinata. Cioè ¡K1,¡K2,V¿ ¿.
Le chiavi di primo livello (K1) identificano le righe composte da coppie chiave-valore e sono chiamati chiavi di riga, le chiavi di secondo livello (K2) sono chiamati chiavi di colonna e servono per identificare il valore.
Un utilizzo tipico che se ne fa è per l'indicizzazione di pagine web;
L'insieme di tutte le colonne è suddiviso in famiglie di colonne. Su disco, gli archivi wide-column non collocano tutti i dati di ogni riga, ma invece i valori della stessa famiglia di colonne e della stessa riga. Quindi, una riga o un singolo valore può essere recuperata solo se nella query ci sono tutte le chiavi necessarie ad identificarla.
Bigtable e HBase, ha aperto la strada al modello a colonne larghe e in parte da esso vi deriva Cassandra. Oggetto di studio nel prossimo capitolo.

Il modello a grafo

Il concetto di grafo, in matematica è una coppia G = (V, E) con V insieme finito di elementi, detti nodi (o vertici) ed E insieme finito di coppie ordinate di nodi detti archi. (o spigoli)
In informatica il grafo è una struttura dati costituita da un insieme finito di coppie ordinate (gli archi) di oggetti (i nodi) che possono essere parte della struttura oppure riferimenti a oggetti esterni.
Questo tipo di database memorizza i dati come un grafo e sono particolarmente adatti a rappresentare dati fortemente interconnessi tra loro, ad esempio quelli dei social network.
Hanno il vantaggio di poter eseguire in maniera efficiente operazioni come il calcolo del percorso minimo tra due nodi, oppure l'individuazione dei nodi direttamente o indirettamente collegati ad un dato nodo di partenza.
Esempi di utilizzo di queste basi di dati sono riscontrabili nel campo della gestione delle reti, nella gestione dei dati geospaziali, nelle biotecnologie e come prima affermato, nei social network. Neo4J è un esempio di grap database open source.

1.5 Cassandra

Di seguito vengono introdotti nuovi concetti chiave che rappresentano come Cassandra salva e gestisce i propri dati.

Il keyspace

il concetto di Keyspace è simile a uno schema nel mondo RDBMS. Un keyspace è un contenitore per tutti i dati dell'applicazione. Quando si definisce uno spazio per le chiavi, è necessario specificare un fattore di replica per il keyspace, questi determineranno il numero di repliche dei dati all'interno di un cluster.

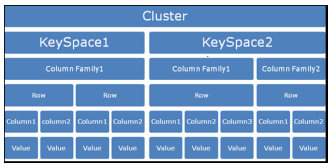


Figura 1.9: Modello di archiviazione in Cassandra

Column Family

una "famiglia di colonne" è una collezione di coppie chiave valore dove il valore è a sua volta una collezione ordinata di altre coppie chiave valore. Cassandra è dotato di un proprio linguaggio per operazioni di definizione, manipolazione, interrogazione e controllo dei dati molto simile ad SQL, si chiama CQL (Cassadra Query Language). Si noti che nel linguaggio CQL una famiglia di colonne viene definita tabella.
Da questo momento quando si parlerà di tabelle ci si riferirà al significato dato al termine dal Cassandra Query Language.

Row key

Una row key è anche nota come chiave di partizione e ha un numero di colonne ad essa associato, ovvero una collezione ordinata, come è stato detto sopra. La chiave di riga è responsabile della determinazione della distribuzione dei dati in un cluster.

Il Cluster

è la struttura più esterna ed è di fatto un contenitore di keyspaces che vivono su uno o più nodi. Un Cluster è detta anche ring perché Cassandra assegna i dati ai nodi nel cluster disponendoli a forma di anello.

Cassandra salva e gestisce i propri dati sotto forma di tabelle. Una ta-

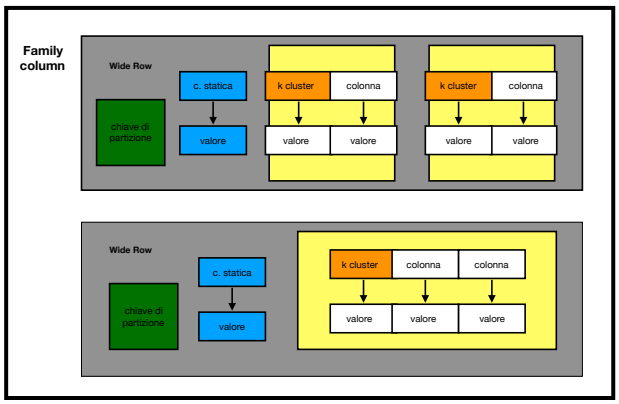


Figura 1.10: Esempio di architettura a due datacenter

bella è una logica divisione che associa dati simili. In ogni tabella possiamo aggiungere una o più righe, e per ogni riga le colonne che vogliamo. Per rappresentare meglio le righe, dette anche partizioni, Cassandra usa chiavi composte.
Una chiave composta è formata da:

- una chiave di partizione, o chiave di riga, che è usata per determinare i nodi su cui sono salvati le righe
- una chiave di clustering, che definisce ogni colonna.

Entrambe, vengono usate, dall'utente per controllare la modalità di ordinamento dei dati per la memorizzazione all'interno di una partizione: con la chiave di partizione viene assicurato che essa conterrà una collezione ordinata di elementi dentro un singolo nodo, mentre la chiave di clustering determina l'ordine fisico di memorizzazione dei dati dentro le partizioni. Oltre a colonne normali, Cassandra introduce la possibilità di avere colonne statiche e queste contengono dati condivisi da ogni riga in una partizione.
Per dare al lettore una più chiara idea nell'immagine sottostante, viene mostrata l'architetture appena descritta nelle figure 1.9 e 1.10

Riassumendo:
Cluster è un container di Keyspaces;

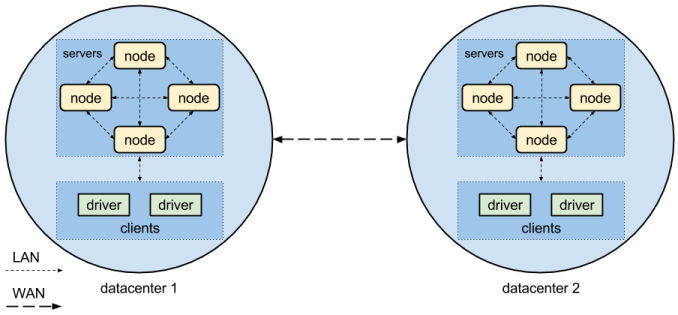


Figura 1.11: Modello di archiviazione in Cassandra

Keyspace è un container di tabelle (o column families);
Table è un container di righe;
Row è un container di colonne referenziate a una primary key;
Una Colonna è una coppia chiave-valore.

1.5.1 Architettura

Cassandra distribuisce i dati in un cluster di nodi che può essere visualizzato come un anello. Un cluster di nodi, è anche detto datacenter, e quest'ultimo può duplicare i dati all'interno di altri suoi simili.
Tutti i nodi in Cassandra sono peer e la richiesta di lettura o scrittura di un client può essere inviata a qualsiasi nodo nel cluster, indipendentemente dal fatto che tale nodo contenga effettivamente i dati richiesti. Non esiste un concetto di master o slave e i nodi imparano in modo dinamico l'uno sull'altro, scambiandosi informazioni in merito allo stato e la salute degli altri nodi. Questo avviene attraverso un protocollo chiamato di gossip.
Quando un nodo che riceve una query dal client viene definito coordinatore per l'operazione; esso facilita la comunicazione tra tutti i nodi di replica responsabili della query (aspettando la risposta di almeno n nodi di replica per soddisfare il livello di coerenza), prepara e restituisce un risultato al client. Un esempio di flusso di scrittura è osservabile in figura 1.12.
Ogni aggiornamento o aggiunta di dati contiene una chiave di riga univoca

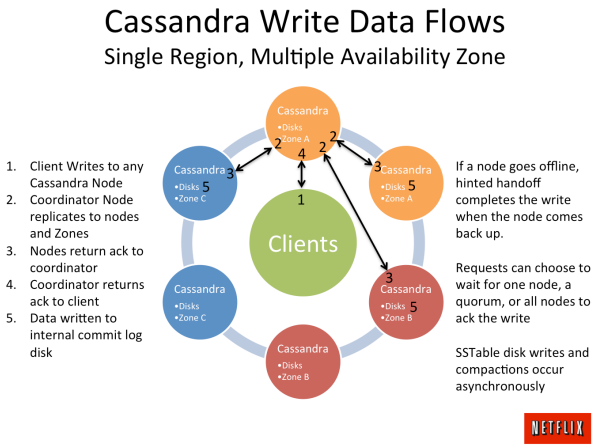


Figura 1.12: Esempio del flusso di scrittura di un dato

(nota anche come chiave primaria). La chiave primaria viene sottoposta a hash per determinare una replica (o nodo) responsabile di essa. I dati vengono inoltre, memorizzati nel cluster tante volte quanto è impostato il fattore di replica del keyspace o, in altre parole, una volta su ciascuna replica responsabile della chiave di riga di una determinata query.
Al paragrafo 2.3.2.1 questi concetti verrano usati per introdurre l'attività di modellazione di una base di dati in cassandra.

1.5.2 Livello di consistenza e fattore di replica

Il fattore di replica è un parametro specificato alla creazione di un keyspace ed indica il numero di repliche di un dato all'interno del cluster.
Il livello di consistenza è un parametro che può essere specificato all'inizio di una sessione o per ogni interrogazione o attività di manipolazione dei dati nel keyspaces.
Cassandra mette a disposizione diversi tipi di livelli di cosistenza visibili alla tabella N. Definendo il livello di consistenza si definisce il numero di risposte da attendere, da parte dei nodi che contengono le repliche, prima di rispondere ad una richiesta del client.

Combinando appositamente i due parametri, è possibile definire il grado di consistenza e di disponibilità di cassandra a livello di query.
Dagli studi fatti da Datastax, azienda che con Cassandra lavora da anni, si è trovata una regola da seguire per poter ottenere la consistenza in lettura : W+R>RF
La somma del livello di consistenza delle letture più la somma del livello di consistenza delle scritture deve essere maggiore del fattore di replica altrimenti è assicurata soltanto la consistenza finale.

Esempi di configurazione

Quando in un sistema distribuito un nodo muore e il cluster è in grado di continuare a funzionare, si dice che il sistema è tollerante ai guasti (in gergo tecnico fault-tollerant).Il sistema deve inoltre essere in grado di riportare allo stato di consistenza il nodo ripristinato.
Se, invece, per qualsiasi motivo, uno o più nodi il sistema non è più attivo si dice che c'è un punto di rottura nel sistema (single point of failure).
Alla figura 1.13 vengono fatti alcuni esempi con diversi livelli di consistenza e viene riportato per ogni scenario il punto di rottura ed il limite di tolleranza ai guasti.
Durante lo sviluppo della base di dati solo due livelli sono stati coinvolti: One e Quorum.
Il primo consente una risposta ad un client non appena abbia replicato la replica più vicina, mentre il valore di Quorum, è dato dalla formula N = (q / 2) + 1. dove q è la somma dei fattori di replica di ogni datacenter in cui è distribuito il keyspace.
Questo livello consente la risposta al client, dopo aver sentito almeno N repliche.
Anticipando il caso d'uso al capitolo 2, il fattore di replica era 3, quindi il livello di quorum corrispondeva a 2.

RF	LC (Scrittura e Lettura)	Consistenza scrittura + consistenza lettura	Punto di rottura	limite di tolleranza ai guasti
3	2	4	2 repliche	1 replica
4	2	4	3 repliche	2 repliche
6	1	2	5 repliche	4 repliche
3	1	2	2 repliche	1 replica

Figura 1.13: Esempi con diversi LC e RF

Capitolo 2

La Progettazione

2.1 Caso di studio : gestione dell'archiviazione di dati temporali di temperatura ed umidità

E' stata progettata un architettura in grado di gestire l'archiviazione persistente ed il recupero di dati provenienti da qualsiasi oggetto IoT.
L'architettura è composta da 3 software ed ognuno di loro svolge un singolare ruolo:
Il database : contiene i dati ed è distribuito su un cluster di nodi. Fornisce alta disponibilità a discapito di una consistenza finale.
Il broker: punto di contatto per gli oggetti intelligenti che vogliono fare inserimenti all'interno del database; riceve i dati da questi ultimi e li filtra per argomento al servizio principale.
Il servizio main: da questo software passano tutte le richieste di scrittura e lettura dei dati. E' il punto di riferimento per il broker che vuole inserire i dati nel db e per i client che vogliono leggerli. Si occupa anche di rendere il carico delle richieste bilanciato.
In figura 2.1 è possibile vedere un immagine dell'architettura; maggiori dettagli di questa verranno dati durante questo capitolo e al prossimo.

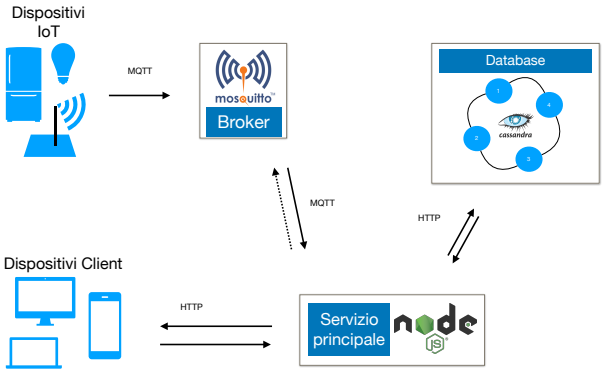


Figura 2.1: Architettura software

Lo scenario d'uso in cui l'architettura è stata utilizzata ha visto far confluire i dati provenienti da una rete di sensori domestica ZigBee all'interno del database, allo scopo di monitorare costantemente la temperatura e l'umidità del laboratorio Ranzani dell'università di Bologna. In questo contesto è stata sviluppata e implementata una base di dati distribuita su 4 nodi in Cassandra. Successivamente, il database è stata testato per diversi livelli di consistenza dei dati e su diverse quantità di dati. I risultati sono riportati al capito 4.
Lo scenario d'uso in cui l'architettura è stata utilizzata ha visto far confluire i dati provenienti da una rete di sensori domestica ZigBee all'interno del database, allo scopo di voler monitorare la temperatura e l'umidità del laboratorio Ranzani dell'università di Bologna. In questo contesto è stata sviluppata e implementata una base di dati distribuita su 4 nodi in Cassandra. Successivamente, il database è stata testato per diversi livelli di consistenza dei dati e su diverse quantità di dati. I risultati sono riportati al capito 4.

2.1.1 Esempio di utilizzo

Le richieste di scrittura e lettura vanno fatte tutte sullo stesso indirizzo ip : 90.147.188.131 Per conservare i dati dentro il database della piattaforma occorre inviare richieste MQTT all'indirizzo ip e alla porta 1883. E' possi-

bile leggere i dati attraverso api publiche effettuando richieste AJAX Http all'indirizzo sopra specificato. Viene fornita pure un interfaccia web raggiungibile via browser. I dettagli della rete ZigBee sono disponibili al report Performance analysis of ZigBee sensor networks[4]

2.2 Analisi

Questo processo è stato diviso in due fasi : Inizialmente si sono raccolti i requisiti necessari per comprendere l'obbiettivo da realizzare. Dopo, questi ultimi sono stati analizzati in dettaglio. In questo paragrafo si vedranno in dettaglio i dati necessari per poter fare la progettazione dell' infrastruttura di archiviazione.

2.2.1 Raccolta dei requisiti

Si vuole progettare una base di dati distribuita su più server in Cassandra che sia in grado di archiviare in modo scalabile le misurazioni di temperatura ed umidità proveniente da una rete di sensori. Si vuole che l'applicazione sia sempre disponibile e che sia garantita la consistenza finale. Il numero di sensori nel contesto varia da 1 a 5 ma in futuro potrebbe variare. La frequenza di scrittura dei sensori dipende dal tipo di misurazione. Inoltre deve essere sempre possibile ottenere i seguenti dati sia per temperatura che per umidità : ultimo valore misurato, il valore massimo giornaliero, il valore minimo giornaliero, la media ed il numero di rilevazioni. Si deve poter anche, essere in grado di raggruppare questi risultati per ogni singolo sensore in modo da poter controllare le misurazioni e le metriche del sensore in questione. Si vuole infine mantenere lo storico delle metriche di ogni sensore.
La base di dati verrà usata per monitorare la temperatura e l'umidità del Laboratorio di informatica Ranzani per l'università di Bologna.

2.2.2 Analisi dei requisiti

Il lavoro è stato suddiviso in altre due sotto-fasi: la fase uno raccoglie e definisce le specifiche sui dati, mentre la fase due svolge lo stesso lavoro ma sulle operazioni.

Fase 1 - Specifiche sui dati Sono dati di tipo time series. Nel database i dati arrivano in modo costante e continuo, con una certa frequenza, per entrambe le misurazioni la frequenza è uguale ma questo è solo un caso. Se l'argomento di misurazione cambiasse la frequenza potrebbe variare anche di molto. La frequenza di temperatura ed umidità per ogni sensore è di circa 1 al minuto. Per ogni tipo di misurazione il messaggio del sensore contiene le seguenti informazioni:

- un parametro id che contiene il nome con cui il sensore si mostra alla rete esterna;
- un timestamp che indica il momento della rilevazione ed è in formato unix;
- il valore della misurazione;
- una lista di ip con tutti i broker a cui il sensore ha mandato il dato aggiornato.

Fase 2 - Specifiche sulle operazioni
si vuole poter leggere l'ultimo valore arrivato per entrambe le misurazioni, inoltre si deve poter conservare statistiche sulle misurazioni che si aggiornano ad ogni nuova rilevazione. Le statistiche sulle metriche comprendono: il numero di rilevazioni effettuate, la media, il valore max, ed il valore minimo e vengono calcolate su un certo periodo di tempo, per esempio l'arco di un giorno. Si deve essere in grado, inoltre, di ottenere le stesse rilevazioni raggruppandole per ogni singolo sensore. Infine è necessario considerare che a leggere i dati saranno algoritmi di Analysis e per questo i dati devono essere preparati per essere letti a gruppi. Un esempio è l'estrazione delle rilevazioni giornaliere di tutti i sensori.

In conclusione: il numero di letture è molto minore rispetto al numero di scritture e tutte le letture avvengono su grandi set di dati.

2.3 Progettazione

Il lavoro riguardante la progettazione si è svolto per iterazioni ed ogni volta si è definito sempre meglio ed in dettaglio il problema e come progettare la soluzione. La sua discussione è stata divisa in 3 parti.
Per prima cosa è stato progettata l'infrastruttura fisica su cui costruire la base di dati, dopo si è costruito il keyspaces e in conclusione è stato ideato il software a contorno di tutto ciò per poter svolgere tutte le funzionalità richieste.

2.3.1 Progettazione dell'infrastruttura

Riportando quanto affermato nella documentazione di Datastax, in Cassandra quanti più nodi fanno parte di un cluster di macchine su cui creare il database distribuito, quanto migliori saranno le prestazioni del cluster. La documetazione dice anche di non creare nodi troppo piccoli dove il troughput necessario per soddisfare una richiesta potrebbe superare la capacità del nodo.
Considerato questi due aspetti si è scelto, per la fase di sperimentazione, di creare quattro macchine virtuali dove installare il database distribuito ed una vm da utilizzare come getway dell'infrastruttura.
Il motivo - come si vedrà al paragrafo "Progettazione dell'architettura software" - è di poter eseguire le entità software più importanti in ambienti separati e completamente indipendenti gli uni dagli altri. Ciò ha facilitato di molto lo sviluppo del software.
Si parlerà meglio di questo argomento al paragrafo 3.2.3

2.3.2 Modellazione dei dati in Cassandra

In Cassandra la progettazione del database non prevede uno schema concettuale per l'organizzazione dei dati in quanto la creazione di tabelle basate sulle entità e sulle relazioni non sfrutterebbe al meglio le potenzialità di Cassandra e per contro porterebbe ad un approccio sbagliato.
Come consigliato da Datastax, azienda che con Cassandra lavora ormai da anni, ci sono 5 consigli utili da seguire per svolgere una buona progettazione delle basi di dati in Cassandra. Questi 5 consigli mettono in risalto i punti di forza di una base di dati performante.
Riassumendo quanto letto sulla documentazione di Datastax :

Su Cassandra, le scritture non sono costose e Cassandra non supporta join, group by, clausole OR, aggregazioni, ecc. Quindi, bisogna archiviare i dati in modo che siano completamente recuperabili. Di seguito alcune regole che conviene seguire durante la modellazione dei dati in Cassandra.

1. Non minimizzare il numero di scritture. Cassandra è ottimizzato per elevate prestazioni di scrittura. Quindi, ottimizza le prestazioni di lettura dei dati aumentando il numero di scritture di dati. Ricorda che c'è comunque un trade-off tra la scrittura dei dati e la lettura.

2. Non minimizzare la duplicazione dei dati. In Cassandra è normale avere tabelle denormalizzate e duplicate. Lo spazio su disco non è più costoso della memoria, dell'elaborazione della CPU e del funzionamento delle operazioni di I/O. Poiché Cassandra è un database distribuito, la duplicazione dei dati garantisce la disponibilità immediata dei dati nel sistema qualora andasse in down un nodo.

3. Distribuisci i dati in modo uniforme attorno al cluster. Usa la chiave di partizione per fare questo. Quindi, scegli correttamente la chiave primaria per diffondere i dati in modo uniforme attorno al cluster.

4. Riduci al minimo il numero di partizioni letto durante l'interrogazione dei dati. In ogni partizione vi è un gruppo di record con la stessa chiave di partizione. Quando viene ricevuta una query di lettura, Cassandra raccoglie i dati che servono. Questi dati possono trovarsi in tante partizioni. Maggiore è il numero di partizioni lette maggiore è il tempo di lettura totale della query. Ciò non significa che le partizioni non dovrebbero essere create. Per dati molto grandi, non puoi mantenere un enorme quantità di dati su una singola partizione. La singola partizione verrà rallentata. Quindi prova a scegliere un numero bilanciato di partizioni.

5. Modella guardando alle queries per minimizzare le partizioni lette. Il modo per ridurre al minimo le letture delle partizioni consiste nel modellare i dati per adattarli alle queries.

   (a) Primo step: determinare le queries da supportare. Cerca di determinare esattamente quali queries devi supportare. Questo può includere molte considerazioni a cui potresti non pensare all'inizio.

   (b) Step due: provare a creare una tabella in cui è possibile soddisfare la queries leggendo (approssimativamente) una partizione. In pratica, questo in genere significa che userete approssimativamente una tabella per modello di query. Se è necessario supportare più modelli di query, in genere è necessario più di una tabella.

2.3.3 Progettazione del database

In linea con queste regole la modellazione delle tabelle è rappresentata in figura 2.2. Si è scelto dividere il lavoro per argomento e quindi creare una tabella per archiviare i dati delle rilevazioni sulla temperature ed una per archiviare i dati sull'umidità.
Per una serie di motivi che verranno elencati più avanti,per distribuire uniformemente i dati all'interno del cluster, come chiave di partizione si è sfruttato il timestamp di arrivo dei dati troncato a 12 ore. Con lo scorrere del tempo,

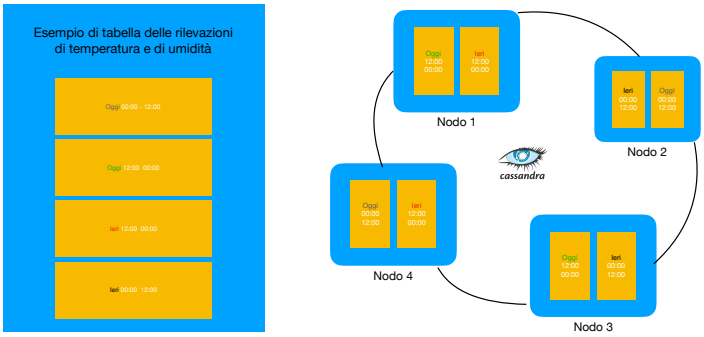


Figura 2.2: Architettura tabelle

si avrà un nuovo timestamp della chiave di partizione ed una nuova chiave verrà creata. Tutto ciò implica anche che i nodi gestori delle repliche dei dati, rimarranno uguali per 12 ore.
In altri termini una partizione contiene tutti i valori delle rilevazioni consecutive, avvenute in un arco di 12 ore. Trascorse le 12 ore, la chiave di partizione cambierà ed i dati verranno salvati su un altri nodi, e sempre sugli stessi, per le successive 12 ore.

I motivi per cui si è scelto di fare un cambio di partizione ogni 12 ore sono i seguenti :

1. la latenza introdotta dalla rete del cloud è veramente alta e la qualità delle macchine virtuali è molto bassa;

2. se si volesse aumentare il numero di sensori senza dover cambiare subito la logica del client si avrebbe un numero di righe maggiori.

Al capitolo dei test è possibile trovare i tempi di latenza in lettura e la quantità di memoria stimata per partizione.

Il motivo della scelta di utilizzare il timestamp come chiave di riga è il seguente:
L'ideale generale sarebbe avere una partizione in grado di scalare dinamicamente e dipendentemente dalla frequenza di inserimento. Se la frequenza aumentasse la partizione scalerebbe più velocemente e viceversa.
Per fornire questo requisito, però, è necessario leggere un parametro indicativo della frequenza di scrittura prima di ogni inserimento. Il paramentro, per esempio potrebbe essere il numero di righe di una partizione oppure il timestamp dell' ultimo valore arrivato.
L'obbiettivo però, è di portare questa logica a livello di Database, in modo che sia indipendente dal client.
Ricordando SQL si potrebbe pensare ad una store procedure che svolge le azioni di lettura prima e scrittura dopo in modo atomico. Cassandra, data la sua natura distribuita, non permette di farlo e rifiuta questo modo di pensare. In Cassandra non esiste il concetto di lettura e scrittura atomica, per contro l'atomicità e garantita fra le scritture e solo a livello di partizione.
Il problema è stato risolto in questo modo.
Concettualmente : se si conoscesse la frequenza di scrittura e se essa non cambiasse, come in questo caso, si potrebbe scalare bene su una partizione generando ogni rowkey, dai valori in arrivo che devono essere archiviati. Poiché nel nostro caso la frequenza è conosciuta a priori e non cambia, non è più necessario leggere un parametro esterno prima di scrivere i dati su una tabella.
Essendo poi un ambiente sicuro e sperimentale non sarebbe necessario nemmeno preoccuparsi di sensori impazziti che scrivono a velocità assurde o peggio ancora di attacchi DDos. Comunque anche per questo si potrebbero adottare soluzioni diverse che permetterebbero di risolvere questo problema. Un broker, per esempio può aggregare più risultati insieme prima di inviarli ad un subscriber.
Ma sono tutte soluzioni lasciate alla fase di implementazione.

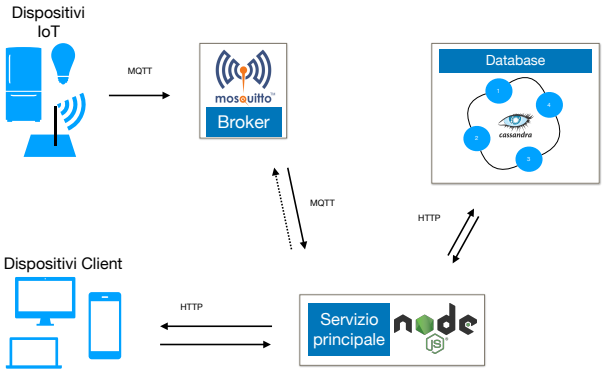


Figura 2.3: Architettura software

2.3.4 Progettazione dell'architettura software

L'infrastruttura è il frutto dell'unione dell'architettura publisher subscriber e di quella client server.
A seguito della progettazione del database, sfruttando le compatibilità delle due architetture sopra citate, è stata creata l'architettura in figura 2.3.
La rete di sensori, tramite il protocollo mqtt è in grado di contattare il servizio principale e può farlo a diversi gradi di qualità come visto in 1.2
A questa si aggiungono due architetture di tipo client- server: la prima è fra il servizio principale e il database, mentre la seconda è fra: le richieste httpp provenienti dai dispositivi client ed il servizio principale.
Il servizio principale svolge 3 ruoli distinti agli occhi di chi interagisce con esso :

- per il broker, è un un sottoscrittore di due argomenti,
- per il database cassandra, è un client di una architettura client server,
- infine svolge il ruolo di server in un architettura per i dispositivi client che vi fanno richieste web http.

L'architettura client - server è implementata secondo le specifiche REST
La sequenza del flusso di scrittura di un dato sul database svolge le seguenti operazioni :

Si attivano Broker e Cassandra, ed il servizio principale,
Il servizio principale, al suo avvio effettua tre operazioni:

1. si sottoscrive al broker per i due argomenti: “temperatura“ ed “umidità“ ;

2. apre una connessione verso i nodi del database;

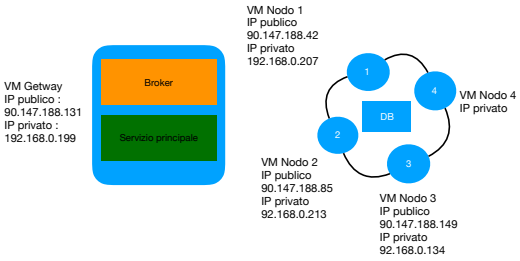
3. si mette in attesa di essere contattato dal broker o da un client che intende leggere i dati.

Quando un dispositivo iot contatta il broker con argometo di temperatura o di umidità; questo ultimo, riceverà l'informazione e la filtrerà al suo sottoscrittore, e cioè il servizio principale. Ricevuti i dati, il servizio principale si preoccupa di salvarli sul database.
Il servizio principale gode di una funzione di load balancing e quando riceve le richieste, può gestirele con una politica Round Robin sui vari nodi in modo da distribuire meglio il carico. Il database riceve le informazioni da archiviare e risponderà al servizio principale l'avvenuto successo o fallimento della scrittura sul database.
La secquenza del flusso di lettura è svolta dai seguenti passi:
un dispositivo client contatta tramite una richiesta http il servizio principale, questo ultimo a sua volta prende in carica la richiesta ed effettua una richiesta di recupero dei dati, effettua una richiesta ad un nodo del database, il quale può fornire una risposta di avvenuto successo o insuccesso. Infine il servizio principale inoltra la risposta con i dati al client.

Infine viene mostrato in figura 2.4 su quali macchie virtuali è stato installato il software creato.

2.4 Funzionalità

La base di dati permette di fornire tutti i requisiti raccolti: il cluster è P2P, distribuitio e ha single-point-of-failure configurabili.



Nome	CPUs	RAM	Capacità di archiviazione	Sistema Operativo
VM Getway	1	2GB	20GiB	Ubuntu 18.04
VM Nodo 1	2	4GB	108GiB	Ubuntu 18.04
VM Nodo 2	2	4GB	108GiB	Ubuntu 18.04
VM Nodo 3	2	4GB	108GiB	Ubuntu 18.04
VM Nodo 4	1	2GB	56GiB	Ubuntu 18.04

Figura 2.4: Architettura hardware virtuale

Inanzitutto, la modellazione fornisce una la logica per scalare i dati dentro le partizioni che è di aiuto ai tempi di latenza in lettura, essa, permette di inserire i dati scalando orizzontalmente.
In secondo luogo il trade off fra che nasce fra consistenza e disponibilità distribuendo un sistema, è configurabile a livello di query usando il fattore di replica ed il livello di consistenza. Verrà citato al capitolo successivo, ma fra le funzionalità del database vi è anche il downsamplig effettuato sulle tabelle con la tecnica “shedlock“ per ridurre il carico della base di dati.
Cassandra assicura la consistenza finale dei dati essendo BASE e AID, e fra le cose permette anche di gestire la sicurezza, dando la possibilità di definire permessi a grana fine su ogni tabella. Anche di questo si discuterà al prossimo capitolo.

Le funzionalità del dell’intera struttura, invece, sono:

- Viene usato solamente software open source che implementano i protocolli standard MQTT e HTTP.
- Rispetta tutti i requisiti dell’analisi
- Il broker può essere utilizzato anche per altri scopi indipendenti
- Bilancia le richieste di scrittura e lettura rivolte al cluster.

- Tutte le richieste sono asincrone.

Capitolo 3

Implementazione

L'implementazione vede coinvolte diverse tecnologie. Verrà spiegato cosa sono e come sono state utilizzate

3.1 GARR Cloud Platform

L'acronimo GARR vuol dire Gruppo per l'armonizzazione delle reti della ricerca. E' una rete a banda ultra larga dedicata alla comunità dell'istruzione della ricerca e della cultura.
La rete è oggi gestita dal Consorzio GARR e ha l'obbiettivo di fornire connettività ad altissime prestazioni per la comunità scientifica accademica e della cultura. Fanno parte della rete GARR i centri di ricerca CNR, ENEA,INFN, CRUI,(nonché fondatori) insieme ad altri centri di ricerca e quasi tutte le università italiane tra cui l'università di Bologna.
Come riporta wikipedia :

> ≪"alla rete GARR sono connesse circa 1000 sedi su tutto il territorio nazionale per un totale di oltre 2 milioni di utenti finali tra docenti, studenti e ricercatori. "≫

Dato che, l'università di Bologna è partecipe di questa iniziativa, è stato

possibile accedere al servizio Cloud con le credenziali di Ateneo. Il servizio Cloud è di tipo Iaas. ma fornisce anche un ambiente PaaS.

3.2 Mosquitto

Mosquitto è un broker MQTT di tipo open source. È un progetto della fondazione Eclipse ed implementa le versioni 3.1 e 3.11 dello standard MQTT. risulta efficiente soprattutto sui dispositivi con limitazioni e reti a bassa larghezza di banda, a latenza elevata o inaffidabili.
Le sue caratteristiche lo rendono perfetto per i sensori che hanno limitazioni non solo di rete ma anche di risorse.
Mosquitto, fra le varie cose che permette di fare, permette anche di creare broker distribuiti. Per questi motivi, in una ottica di estensione del progetto, Mosquitto risulta uno fra i miglior candidati.

E' stato installato Mosquitto su una macchina virtuale ed è stato messo in ascolto per ricevere le richieste mqtt.
Potenzialmente, qualora fosse necessario, altri publisher ed altri subscriber ulteriori potrebbero utilizzarlo per lo scambio di messaggi e di altri topic.
Attualmente, ovviamente, sono connessi al broker un sottoscrittore per il topic "Temperature" ed uno per il topic "Humidity", in entrambi l'ente software subscriber degli argomenti è il driver client del database.

3.3 NodeJs

Nodejs è un interprete di Javascript nato dall'interprete del browser Google Chrome e permette quindi di eseguire il codice del client lato server.Come cita wikipedia:

≪" Node.js ha un'architettura event-driven capace di gestire I/O asincro-

ni. Questa scelta progettuale mira a ottimizzare il Throughput e la scalabilità nelle applicazioni web che prevedono molte operazioni di input/output o per applicazioni web Real-time (ad esempio programmi di comunicazione in tempo reale o browser game). “ »

Nodejs, per la sua architettura event-driven, si comporta nel seguente modo: dopo che un programma nodejs è avviato, esso esegue il lavoro iniziale e non termina ma rimane in stato “sleep” fino a quando non accade un evento tale per cui il SO ospitante si occupa di svegliarlo attraverso una notifica. In Nodejs la programmazione del software avviene con i moduli o con i framework. Un modulo in Nodejs è un insieme di file javascript dove sono contenute funzionalità che possono essere riutilizzate in tutta l’applicazione Nodejs. Ogni modulo ha il proprio contesto e non può interferire con altri moduli o inquinare l’ambito globale. Lo standard implementato dai moduli di Nodejs è stabilito dal gruppo CommonJS. I moduli Nodejs possono essere definiti di tre tipologie : moduli core, moduli locali e moduli di terze parti. I primi includono le funzionalità minime di Nodejs. E’ necessario importarli e poi è possiblie utlizzarli nell’applicazione. Per esempio http è un modulo core che è stato usato. Esso include classi, metodi ed eventi per creare un programma in ascolto di richieste http. I moduli publici o di terze parti sono moduli che è possibile scaricare da internet. I moduli publici che sono stati usati durante il mio lavoro sono Express js, cassandra-driver, mqtt, cronos. Infine i moduli locali sono moduli creati localmente ed usati nell’applicazione lato server.

3.3.1 Server utils

Sono stati creati tre moduli locali : un modulo è stato chiamato mosquitto _module ed uno cassandra_module ed entrambi servono da supporto al modulo principale sofia che è omonimo del keyspace.
Il modulo sofia.js, è il cuore dell’applicazione. Risponde alle richieste dell’interfaccia web e svolge due operazioni: attiva il modulo mosquitto_module e sfrutta cassandra_module per rispondere alle richieste delle API per ottenere

i dati dal database.
Cassandra_module si occupa di interagire con il database.Crea una connessione e tiene conto dei nodi e del loro carico di lavoro, bilancia le richieste di lettura e di scrittura e stabilisce il livello di coerenza per ogni singola queries. Permette pure di creare pool di connessioni per nodo.
Questo modulo, oltre ad essere utilizzato dal modulo principale per il recupero dei dati, viene utilizzato anche da mosquitto_module per effettuare gli inserimenti sul database alla ricezione di nuovi messaggi.

Mosquitto_module si connette al broker e ne chiede la sottoscrizione dei due topic precedentemente citati, dopodiché rimane in attesa di ricevere i messaggi dal broker. Quando un messaggio arriva viene letto per argomento e da li inserito tramite cassandra_module sul database

Riassumendo, sono stati creati 3 moduli : il modulo principale, mosquitto e cassandra.
Ogni modulo si occupa di eseguire una parte del proprio lavoro ed è indipendente dall'altra. Ne basta cambiarne uno senza dover modificare tutti gli altri per apportare delle modifiche.

3.3.2 Clients utils

Un interfaccia web è stata creata per poter leggere i dati e visualizzarli via browser. L'interfaccia è raggiungibile all'indirizzo ip 90.147.188.131

3.4 Cassandra

3.4.1 Il Cluester

Il cluster creato è formato da 4 nodi. In figura 3.1 l'output digitando il comando nodetool status (un tool che fornisce lo status del cluster) sul nodo

```
root@nodeone:/home/ubuntu/Fiorilla/Test# nodetool status
Datacenter: dc1
===============
Status=Up/Down
|/ State=Normal/Leaving/Joining/Moving
--  Address         Load       Tokens       Owns (effective)  Host ID                               Rack
UN  192.168.0.207   956.06 KiB  256          84.3%             33703f4a-8db4-43fc-b876-648de93c1046  rack1
UN  192.168.0.226   586.96 KiB  128          54.9%             3172dc84-ec26-4890-a683-58b40713e7e6  rack1
UN  192.168.0.213   952.07 KiB  256          81.6%             df841c03-a6bf-44b8-9d58-3d6bb4922ac7  rack1
UN  192.168.0.134   961.67 KiB  256          79.2%             2b3eb08b-3047-453e-a0bb-451540cf72d2  rack1
```

Figura 3.1: Nodetool status

uno del cluster. Uno dei 4 nodi è più piccolo degli altri e ad esso sono stati associati un minor numero di tokens. Questa decisione deriva dal fatto che, il carico di lavoro su cassandra dipende dal numero di tokens in un nodo; in questo caso esso avrà la metà del carico di lavoro svolto rispetto agli altri nodi.

3.4.2 Il Keyspaces

E' composto in totale da 8 tabelle:

1. datastream_temperature_by_time : conserva i rilevamenti della temperatura globali fra tutti i sensori in ordine di arrivo (si noti il campo ts in figura 3.2)

2. datastream_temperature_by_sensor: contiene i rilevamenti della temperatura raggruppando per sensore e poi per ordine di arrivo (si noti la riga 8 in figura 3.2)

3. datastream_humidity_by_time : contiene i rilevamenti dell'umidità globali fra tutti i sensori in ordine di timestamp.

4. datastream_humidity_by_sensor: contiene i rilevamenti dell'umidità aggregati per sensore e dopo timestamp.

5. metrics_temperature_by_sensor: contengono metriche statistiche, di ogni singolo sensore, calcolate sulle rilevazioni giornaliere di temperatura. I dati in questa tabella vengono aggiunti dai processi demoni che svolgono l'attività di downsampling. Saranno queste tabelle che conserveranno i dati nel lungo periodo.

```
CREATE TABLE datastream_temperature_by_time(
    shardid timestamp,
    sensorid text,
    ts timestamp,
    value double,
    broker text,
    primary key((shardid),ts, sensorid)
)WITH CLUSTERING ORDER BY(ts DESC, sensorid ASC)
    AND COMPACTION = {
        'class': 'TimeWindowCompactionStrategy',
        'compaction_window_unit': 'DAYS',
        'compaction_window_size': 1};
```

```
CREATE TABLE datastream_temperature_by_sensor(
    shardid timestamp,
    sensorid text,
    ts timestamp,
    value double,
    broker text,
    primary key((shardid), sensorid, ts)
)WITH CLUSTERING ORDER BY(sensorid ASC, ts DESC)
    AND COMPACTION = {
        'class': 'TimeWindowCompactionStrategy',
        'compaction_window_unit': 'DAYS',
        'compaction_window_size': 1};
```

Figura 3.2: Esempio di modellazione tabelle

6. metrics_humidity_by_sensor : contengono statistiche, aggregate per sensore, calcolate sulle rilevazioni giornaliere di umidità. E' identica alla tabella metrics_temperature ma contiene dati sull'umidità

7. lock_temperature: contiene i dati riguardanti l'esecuzione del lavoro dei processi di downsampling. Se ne parlerà meglio al paragrafo 3.4.3.

8. lock_humidity : uguale a lock_temperature ma per i processi di downsampling sui dati di umidità

In figura 3.2 è possibile osservare due esempi di tabelle utilizzate per lo storage. Seguendo le indicazioni delle regole di modellazione al paragrafo 2.3.2, sono state create due diverse tabelle per rispondere a query di lettura più velocemente. Questo ha costo in memoria ma in conpenso si recupera molto tempo per le richieste di lettura.
Per agirare questo problema è stata trovata la soluzione del downsampling di cui si parlerà nei paragrafi più avanti.

Come è stato precedentemente affermato qualche paragrafo prima, nelle colonne di clustering i dati vengono ordinati seguendo un ordine che può essere ascendente o discendente. In Cassandra, è solito sfruttare l'ordinamento per velocizzare le scritture.
Come visibile in figura 3.2 si è scelto di utilizzare un ordine di timestamp e sensore favorevole alle letture, e questo ha permesso di velocizzarle.

3.4.3 Il Downsampling

Il downsamplig è una tecnica che permette di ridurre la quantità di memoria della base di dati. Consiste in un processo che calcola statistiche sui dati già archiviati allo scopo mantenere il valore delle statistiche e poter cancellare i dati su cui si è lavorato.
Per eseguire questo processo nel database, è stato implementato un un servizio in background; demon.js. Esso, è attivo su ogni nodo del cluster e si trova in stato di sleep. La concorrenza è gestita grazie alla tecnica dello shedlock, cioè una tecnica che, in un contesto di processi paralleli e concorrenti, permette di far eseguire il lavoro su un area critica di memoria una ed una sola volta.
Dopo che i dati sono stati inseriti dal client nel database con un time to live di quattro giorni e mezzo, ogni giorno a ad una certa ora, il processo demone viene attivato da un altro processo demone; Cronos di linux. A seguito dell'attivazione, demon.js prova ad ottenere il permesso per eseguire il downsampling. Se questo permesso viene ottenuto allora a quel punto, riesce a leggere dal database i dati delle rilevazioni vecchie di due giorni prima, successivamente vi calcola per ogni sensore: media, valore massimo, valore minimo e numero di rilevazioni. I dati calcolati vengono salvati su una nuova tabella. Per quanto riguarda i dati vecchi verranno eliminati da Cassandra dopo che il loro ttl scadrà.

Nella pratica: sono state create sul database due tabelle che sono state chiamate lock_temperature e lock_humidity. Queste due tabelle sono formate da 3 colonne, la prima si chiama id ed è chiave, la seconda state e la terza status.
Scrivere su questa tabella equivale a prendere un lock. Concorrentemente più demoni nello stesso momento verranno richiamati allo scoccare del clock di Cronos, il demone linux. Una volta attivi si connetteranno a Cassandra e proveranno a scrivere inserendo solamente l'id. Visto che il comando insert permette la clausula "if not exists", solo il primo nodo che riuscirà a scrivere

```
CREATE TABLE IF NOT EXISTS lock_temperature (
    id timestamp PRIMARY KEY,
    error text,
    state int
)

CREATE TABLE IF NOT EXISTS lock_humidity (
    id timestamp PRIMARY KEY,
    error text,
    state int
)


INSERT INTO lock_humidity (id )
VALUES(toUnixTimestamp(maxTimeuuid ( ? ) ))
IF NOT EXISTS USING TTL 3600

UPDATE lock_humidity SET state = ?
USING TTL 0 , error = ?
WHERE id = toUnixTimestamp(maxTimeuuid ( ? ) )
```

Figura 3.3: Shedlock

sulla tabella vedrà come risultato della query applied uguale a true, tutti gli altri che proveranno a prendere il lock vedranno come risultato applied uguale a false.
Il client che riuscirà a prendere il lock eseguirà le azioni discusse sopra in merito al downsampling. Completato il lavoro, il demone aggiornerà i paramentri del lock riportando alla colonna status l'ip del nodo che ha eseguito insieme ad eventuali messaggi e un valore compreso fra 0 e 4 alla colonna state.
Anche il lock viene inserito con il ttl di un ora perchè se un nodo cade durante l'esecuizione del suo lavoro e dunque non lo completa il lock tornerà disponibile ed altri potranno successivamente riprovare a prendere il lock ed eseguire il lavoro. Difatti, per questo motivo, i tentativi di acquisizione del permesso sono impostate una ogni ora dalle 00 alle 4 di mattina ai minuti 25 e 50. Ora locale dei nodi server.

Capitolo 4

I Test

4.1 Risultati sperimentali

Di seguito vengono presentati i risultati dei test sperimentali svolti sulla base di dati creata.
E' stato calcolato il tempo di latenza in lettura espresso in millisecondi e ogni test si differenzia per tipologia di query. Queste ultime sono state eseguite da un programma creato appositamente per 10 000 volte con una frequenza di emissione di 10 ms.

4.2 Analisi della memoria

Si stima che, in una riga un valore colonna ed un valore chiave all'incirca pesino entrambi 31 bytes per parametro. Considerato che : in una famiglia di colonne ci sono 3 valori chiave e 2 valori colonna, il numero di bytes per riga è di circa 155 bytes. In figura 4.1 vengono messe a confronto, seguendo la logica del modello di archiviazione del paragrafo 2.3.2.2, le grandezze che si avrebbero se si distribuisse una nuova partizione ogni 24 ore oppure ogni 12 ore, a fronte di un aumento del numero di sensori e considerando che, ogni sensore non vari la sua velocità di rilevazione, ovvero di 1 volta al minuto.

tabella 1 - peso partizioni stimato in 12 ore vs 24 ore					
24 ore			12 ore		
numero di sensori	numero di righe dopo 24 ore	peso stimato di una partizione in bytes	numero di sensori	numero di righe dopo 12 ore	peso stimato di una partizione in bytes
1	1440	223200	1	720	875
2	2880	446400	2	1440	223200
3	4320	669600	3	2880	446400
4	5760	892800	4	3600	558000
5	7200	1116000	5	4320	669600
6	8640	1339200	6	5040	781200

Figura 4.1: Analisi memoria stimata

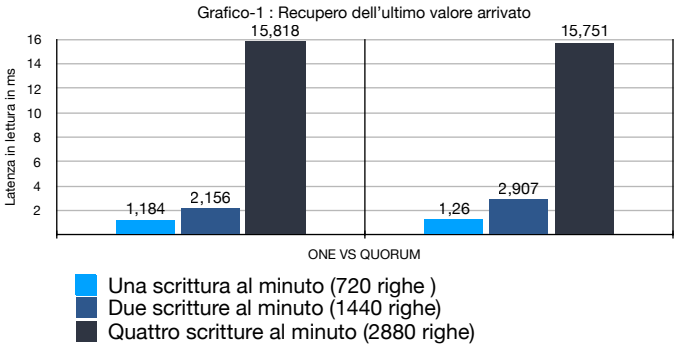


Figura 4.2: Grafico 1: risultati primo test

4.3 Primo test : estrazione dell'ultimo valore di temperatura rilevato

In questo primo test la query di lettura effettua l'estrazione dell'ultimo valore di temperatura rilevato e salvato sul database. Il test è stato ripetuto su diverse quantità di bytes, e precisamente, è stato eseguito prima su una partizione di 720 righe, poi su una partizione di 1440 ed infine, l'ultimo test è stato effettuato su una partizione di 2880 righe.
In figura 4.2 vengono messi a confronto i tempi medi di latenza delle letture espressi in millisecondi per i livelli di consistenza Quorum e One.
Invece, i risultati dettagliati del test con e senza transienti vengono riportati in figura 4.3

Tabella - risultati primo test		
partizione di 720 righe		
Livello di consistenza in lettura	media dei tempi di latenza delle letture su 720 righe con transiente	media dei tempi di latenza delle letture su 720 righe senza transiente
One	1,18398679735947 ms	1,18398679735947 ms
Quorum	1,26039207841568 ms	1,19573888689036 ms
partizione di 1440 righe		
Livello di consistenza in lettura	media dei tempi di latenza delle letture su 1440 righe con transiente	media dei tempi di latenza delle letture su 1440 righe senza transiente
One	2,15567113422685 ms	2,15506464361737 ms
Quorum	2,90668633726745 ms	2,65217392656024 ms
partizione grande 2880 righe		
Livello di consistenza in lettura	media dei tempi di latenza delle letture su 2880 righe con transiente	media dei tempi di latenza delle letture su 2880 righe senza transiente
One	15,8176415982931 ms	15,652124106128 ms
Quorum	15,751275255051 ms	15,6242524965543 ms

Figura 4.3: Risultati primo test

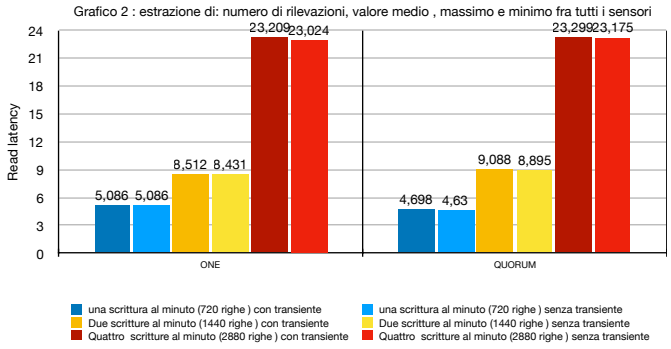


Figura 4.4: Grafico 2: secondo test

4.4 Secondo test : estrazione di metriche calcolate su tutti i sensori

In questo test la query di lettura calcola 4 metriche di misura diverse su tutte le rilevazioni dei sensori avvenute in un arco di tempo di 12 ore, ovvero una partizione (paragrafo 2.3.2.). I valori ottenuti dai calcoli vengono restituiti tutti insieme, e comprendono: il numero di rilevazioni archiviate (corrisponde al numero di riga), il valore medio, il valore massimo ed il valore minimo della temperatura in quell'arco di tempo. In figura 4.4 vengo mostrati i tempi medi di latenza delle letture per i livelli di consistenza Quorum e One. Di seguito, in figura 4.5 la tabella con i risultati dei test non troncati.

Tabella 3 - risultati secondo test		
partizione di 720 righe		
Livello di consistenza in lettura	media dei tempi di latenza delle letture su 720 righe con transiente	media dei tempi di latenza delle letture su 720 righe senza transiente
One	5,08592218443689 ms	5,08592218443689 ms
Quorum	4,69784456891378 ms	4,62958010526448 ms
partizione di 1440 righe		
Livello di consistenza in lettura	media dei tempi di latenza delle letture su 1440 righe con transiente	media dei tempi di latenza delle letture su 1440 righe senza transiente
One	8,5120024004801 ms	8,43073713287565 ms
Quorum	9,0882826565313 ms	8,89532540506484 ms
partizione grande 2880 righe		
Livello di consistenza in lettura	media dei tempi di latenza delle letture su 2880 righe con transiente	media dei tempi di latenza delle letture su 2880 righe senza transiente
One	23,2093115464296 ms	23,0238925951141 ms
Quorum	23,298674734947 ms	23,1751249978911 ms

Figura 4.5: Risultati secondo test

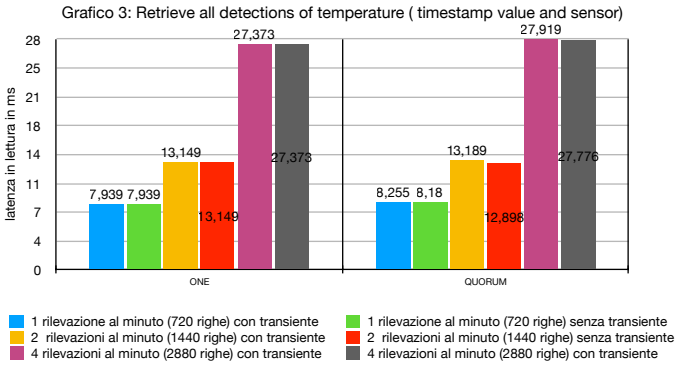


Figura 4.6: Grafico 2: terzo test

4.5 Terzo test : recupero delle rilevazioni sulla temperatura effettuati durante 12 ore

In questo test vengono recuperati da una partizione tutti i valori di temperatura ed il loro timestamp di arrivo. Nella tabella in figura 4.7 vengono rappresentati i risultati delle medie dei tempi di latenza non arrotondati, invece nel grafico in figura 4.6 vengo illustrati i tempi medi di latenza delle letture, espressi in millisecondi, mettendo a confronto i livelli di consistenza Quorum e One.

Tabella 4 - risultati terzo test		
partizione di 720 righe		
Livello di consistenza in lettura	media dei tempi di latenza delle letture su 720 righe con transiente	media dei tempi di latenza delle letture su 720 righe senza transiente
One	7,93904780956191	7,93904780956191
Quorum	8,25483596719344	8,17985955251681
partizione di 1440 righe		
Livello di consistenza in lettura	media dei tempi di latenza delle letture su 1440 righe con transiente	media dei tempi di latenza delle letture su 1440 righe senza transiente
One	13,1494653101872	13,1494653101872
Quorum	13,1890078015603	12,8983934858301
partizione grande 2880 righe		
Livello di consistenza in lettura	media dei tempi di latenza delle letture su 2880 righe con transiente	media dei tempi di latenza delle letture su 2880 righe senza transiente
One	27,3729389971985	27,3729389971985
Quorum	27,9186587317463	27,7764202032579

Figura 4.7: Risultati terzo test

4.6 Quarto test : vengono recuperati tutti i rilevamenti di un sensore

Nel grafico in figura 4.8, sono stati messi a confronto i tempi medi di latenza di lettura, sul test svolto, per i livelli di consistenza Quorum e One, mentre alla tabella in figura 4.9, sempre in merito al test, vengono riportati i valori dei millisecondi senza troncamento.
In questo test si è interrogato il database recuperando: il numero di rilevazioni effettuate, il valore medio, minimo e massimo delle rilevazioni di ogni sensore in una partizione.

4.7 Considerazioni finali sui test

In merito al test uno, è possibile notare che il livello di consistenza Quorum non ha dei tempi di latenza tanto diversi dal livello One; esso ottiene i tempi migliori nella partizioni di 720 e 2880 righe; dove, però, si registra un tempo maggiore se si eliminano i transienti. Si nota anche l'andamento esponenziale del tempo di lettura a fronte di un incremento lineare della quantità della partizione.

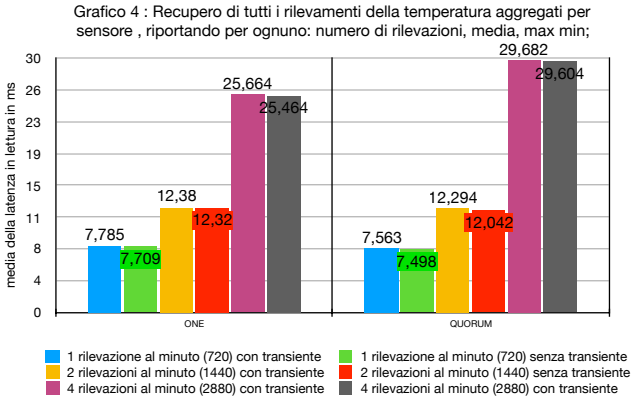


Figura 4.8: Grafico 2: quarto test

Tabella 4 - risultati terzo test		
partizione di 720 righe		
Livello di consistenza in lettura	media dei tempi di latenza delle letture su 720 righe con transiente	media dei tempi di latenza delle letture su 720 righe senza transiente
One	7,78540708141628	7,70904220031161
Quorum	7,5625525105021	7,49796459693303
partizione di 1440 righe		
Livello di consistenza in lettura	media dei tempi di latenza delle letture su 1440 righe con transiente	media dei tempi di latenza delle letture su 1440 righe senza transiente
One	12,37950090018	12,3195166092928
Quorum	12,2943938787758	12,0423947778364
partizione grande 2880 righe		
Livello di consistenza in lettura	media dei tempi di latenza delle letture su 2880 righe con transiente	media dei tempi di latenza delle letture su 2880 righe senza transiente
One	25,6640562648961	25,4643640870208
Quorum	29,6816423919038	29,6036865021733

Figura 4.9: Risultati quarto test

Il test due vede registrare tempi migliori ancora per il Quorum alla prima grandezza di partizione, sia con che senza transienti. Successivamente il livello di consistenza One registra tempi più veloci in tutti gli altri scenari di questo test. Anche qui è possibile vedere tempi di lettura crescenti esponenzialmente a fronte di un aumento lineare.

Il terzo test è il test che estrae dal db il maggior numero di dati però senza doverli elaborare. Sono visibili tempi crescenti in modo più o meno lineare e quindi d'accordo con l'aumento della quantità dei dati.
Il livello One, in questo test ha tempi migliori in tutti gli scenari.

Infine, per il test quattro, si evince un andamento migliore del livello di consistenza Quorum per le prime due partizioni. Successivamente, all'aumentare della quantità, ovvero 2880 righe, One registra tempi migliori. Il tempo di lettura in questo test cresce linearmente come la quantità.

Per quanto riguarda il confronto fra i due livelli di consistenza Quorum e One, non si evince un notevole rallentamento con il livello di consistenza Quorum - anche se il livello di consistenza One prevale 2 volte su 3 - non c'è una netta differenza di tempi fra i due. Il motivo è dovuto alle condizioni della rete usata e probabilmente al basso carico di memoria.
Casca all'occhio, inoltre, che nel primo e nel secondo test, a fronte di un aumento lineare della quantità di dati letta, si ottiene un aumento esponenziale del tempo impiegato per leggere su una partizione. Mentre, come è stato visto nelle analisi singolari, ai test 3 e 4 si registrano incrementi di tempo lineari.
Il motivo è dovuto alle operazioni di recupero e di calcolo, che Cassandra effettua dentro ai server, per generare i risultati delle query. Queste operazioni a differenza della logica di modellazione non sono state ottimizzate. Difatti nel test 3, dove non vengono effettuate operazioni computazionali dentro al

server, il tempo in latenza risulta lineare.

L'andamento dimostra che la modellazione dei dati è di aiuto ai tempi di latenza per le operazioni di lettura, quindi, alla luce di questi risultati è stato ritenuto opportuno non cambiare la logica di archiviazione dei dati.
Se si aumentasse la grandezza di una partizione potrebbe anche andare bene ma, in un‘ ottica di incremento del numero di sensori, la distribuzione dei dati risulterebbe efficiente ma con tempi troppo lunghi.

Capitolo 5

Conclusioni

Si è presentata un architettura in grado di gestire l'archiviazione ed il recupero di dati provenienti da oggetti intelligenti. L'architettura implementa tre entità software: un broker mqtt, un client ed un database distribuito facilmente utilizzabili in numerosi contesti hi-tech. Al paragrafo 2.1. ne sono state illustrate le funzionalità, mentre al paragrafo successivo è stato illustrato un esempio di utilizzo. Successivamente si è poi analizzato in dettaglio uno scenario d'uso che ha visto coinvolta l'architettura in un contesto reale: sono stati archiviati i dati provenienti da una rete di sensori ZigBee per uso domestico per monitorare la temperatura del laboratorio Ranzani dell'università di Bologna sito in Via Ranzani 14 c a Bologna. Al paragrafo 2.3, si è discusso anche sulla progettazione della base di dati in Cassandra creata per il contesto, descrivendo i problemi affrontati, le soluzioni adottate per risolverli ed i motivi che hanno portato a prendere le decisioni. Al capitolo 3 sono stati visti gli aspetti caratterizzanti dell'implementazione svolta, infine al capitolo 4 sono stati riportati i risultati dei test sperimentali svolti sulla base di dati. Da questi ultimi si evincono tre cose: In primo luogo si evince come un aumento del livello di consistenza dei dati non influenzi le prestazioni di lettura, in secondo luogo che i tempi di lettura medi crescono esponenzialmente a fronte di un aumento lineare dei dati ed infine che la modellazione della base di dati è di aiuto all'hardware.

5.1 Sviluppi futuri

Il progetto è solo all'inizio di un lavoro più ampio che prevede l'aggiunta di funzioni verticali e orizzontali alla base di dati:

- Spostare l'architettura virtualizzando con i containers Docker ed integrare questi con Kubernets : un contenitore è un'unità standard di software che racchiude il codice e tutte le sue dipendenze, e permette di far funzionare l'applicazione rapidamente e in modo affidabile indipendentemente dall'ambiente di elaborazione. Kubernetes, invece, è uno strumento open source di orchestrazione e gestione di container.

- Espandere il cluster : sia come numero di nodi sia creando un altro datacenter in un altro luogo, per poter delocalizzare i dati ed aumentarne la disponibilità.

- Rilasciare il codice prodotto open source.

- Applicare l'architettura in un contesto industriale. Innumerevoli sono le possibilità di mercato che possono nascere con l'iot ed i numeri riguardo l'uso di quest'ultimo e gli investimenti che riceve, presentati al capitolo 1, sono di supporto a questa decisione.

- Aumentare la sicurezza: Essendo un ambiente sperimentale non sono stati usati i certificati e dunque necessario passare ai protocolli https e mqtts.

- Integrare la base di dati con i nuovi strumenti di visualizzazione ed Analysis per BigData e time series.

Bibliografia

[1] https://www.businessinsider.com/intelligence/research-store?IR=T#!/The-IoT-Forecast-Book/p/108427443/category=11987294?IR=T&utm_source=businessinsider&utm_medium=content_marketing&utm_term=content_marketing_subscription_text_link_iot-forecast-book-2018-7&utm_content=IoT_Forecast_Book_2018&utm_campaign=report_teaser&vertical=iot

[2] https://www.internet4things.it/iot-library/le-previsioni-di-idcsulla-spesa-iot-nel-2019/

[3] link_statistica:https://www.emc.com/leadership/digital-universe/2012iview/executive-summary-a-universe-of.html

[4] https://www.researchgate.net/publication/331357184_Performance_analysis_of_ZigBee_sensor_networks

Ringraziamenti

Ringrazio chi mi è stato sempre vicino durante questi anni.

Un ringraziamento sentito e particolare va a mia sorella Sofia.

Ringrazio il mio relatore di tesi, il prof. Di Felice per il sostegno datomi durante questi mesi di lavoro .