Implementazione di un'Architettura Lambda

Davide Massuda, Giulio Raponi

July 2021

1 Abstract

Con il rapido sviluppo del web, delle app mobili e dei dispositivi IoT, un enorme volume di dati viene creato ogni giorno. Questi dati non sono solo massivi, ma vengono anche generati rapidamente e con una varietà di formati diversi. Molte aziende hanno interesse nell'elaborare questi "big data" in tempo reale o quasi. All'interno di diversi domini applicativi, alcuni dati devono essere elaborati real time, mentre per altri è possibile utilizzare un'elaborazione batch offline. L'Architettura Lambda è una soluzione che unisce sia i vantaggi delle tecniche di elaborazione batch che quelli streaming. In questo lavoro, verrà approfondita la realizzazione di un'architettura Lambda per l'analisi e il monitoraggio di dati nell'ambito delle criptovalute utilizzando diversi software open-source.

2 Introduzione

Con l'enorme crescita delle quantità di dati, ci sono state diverse innovazioni sia nella memorizzazione che nell'elaborazione dei big data. A causa di sorgenti come sensori IoT, interazioni nei Social Network, log delle applicazioni e i cambiamenti di stato dei database, il numero di dati da tenere in considerazione è cresciuto esponenzialmente. Questi stream rappresentano flussi di dati continui e senza limiti che richiedono molto spesso un'elaborazione quasi in tempo reale. Il campo dell'analisi dei dati sta crescendo immensamente per trarre preziose intuizioni da grandi quantità di dati grezzi (raw data). Al fine di ricavare informazioni in un sistema di dati, i sistemi di elaborazione risultano essere essenziali, come la necessità di disaccoppiare il calcolo dalla memorizzazione. I frameworks di elaborazione possono essere classificati in tre categorie: elaborazione batch, elaborazione dei dati stream e sistemi di elaborazione dei dati ibridi. L'elaborazione tradizionale dei dati batch dà buoni risultati ma con un'alta latenza. Al fine di ottenere risultati in tempo reale con bassa latenza, una buona soluzione è quella di utilizzare strumenti come, ad esempio, Apache Kafka accoppiato con Apache Spark o con altri sistemi di elaborazione (near) real time. Questi tipi di modelli streaming risultano essere ottimi in termini di alta disponibilità e bassa latenza, ma potrebbero non garantire un'elevata precisione. Nella maggior parte degli scenari, i casi d'uso richiedono sia risultati

veloci che un'elaborazione approfondita dei dati. Questo progetto si focalizza su quella che viene chiamata Architettura Lambda, in grado di unificare sia l'elaborazione batch che quella quella streaming. Nello specifico la Lambda Architecture si compone di tre strati principali:

- Batch Layer
- Serving Layer
- Speed Layer

In questa relazione viene presentata la realizzazione di un'architettura Lambda, attraverso la sperimentazione di una o più tecnologie che combinano Stream Analysis e Batch Analysis. In particolare, si vuole realizzare un sistema in grado di estrarre dati da una fonte esterna e analizzarli in modo differente, attraverso l'utilizzo di strumenti che processino i dati in tempi differenti. A seguito della fase di data ingestion, il processo viene diviso in due rami: da una parte, i dati arrivano in flussi e vengono elaborati man mano che arrivano, nell'altro invece, i dati vengono memorizzati "as is" in un dataset persistente e vengono poi applicate delle elaborazioni. Successivamente questi vengono mostrati ed eventualmente consumati.

3 Architettura

La realizzazione di questa architettura ha compreso l'utilizzo di un cluster composto da 6 nodi. Il tutto è stato possibile grazie a quattro nodi Raspberry pi3 e due personal computer. Di seguito riportiamo le specifiche tecniche:

- 4x Raspberry pi3: 1.2 GHz ARM Cortex-A53 CPU
- 1x Asus VivoBook S15: Intel Core i7-8565U Processor 1.8 GHz, 16GB di RAM, GPU GeForce MX150
- 1x MacBook Air: 1,1 GHz Quad-Core Intel Core i5, 8 GB di RAM, Intel Iris Plus Graphics 1536 MB

3.1 Scelta dei Framework

In questa sezione vengono descritte le tecnologie utilizzate per la realizzazione del progetto. Nello specifico, per quanto riguarda l'ingestion dei dati, si è deciso di fare affidamento su Apache Kafka, un sistema di messaggistica Publish-Subscribe in grado di gestire code di flussi provenienti da varie sorgenti. In Kafka è possibile individuare due attori principali: Publisher che producono contenuti memorizzati in Topic e Subscriber, che si iscrivono a questi contenuti e ne usufruiscono nell'ordine e nella velocità che preferiscono. Il ramo relativo all'analisi (near) real time (ovvero lo Speed Layer) è stato realizzato attraverso l'utilizzo di Apache Flink, un framework open source in grado di elaborare processi streaming con alto throughput e bassa latenza. Parlando invece della parte

relativa al Batch Layer, il flusso di dati proveniente da Kafka è stato memorizzato in MongoDB. Questo rappresenta un sistema di storage NoSQL aggregate-oriented di tipo document store, basato su paradigma Master-Slave. I dati così memorizzati sono stati poi analizzati attraverso l'utilizzo di Apache Spark, un sistema in grado di elaborare grandi moli di dati, basato sul paradigma di programmazione map-reduce e costruzione di DAG (*directed acyclic graph*). La memorizzazione degli output del livello Batch e del livello Speed, è stata affidata a ElasticSearch, un sistema open source scalabile, che permette di memorizzare, cercare e analizzare grandi volumi di dati rapidamente. Per quanto riguarda la visualizzazione delle informazioni ottenute ci si è affidati a Kibana, un'interfaccia utente open source che permette di visualizzare e navigare attraverso i dati provenienti da Elasticsearch.

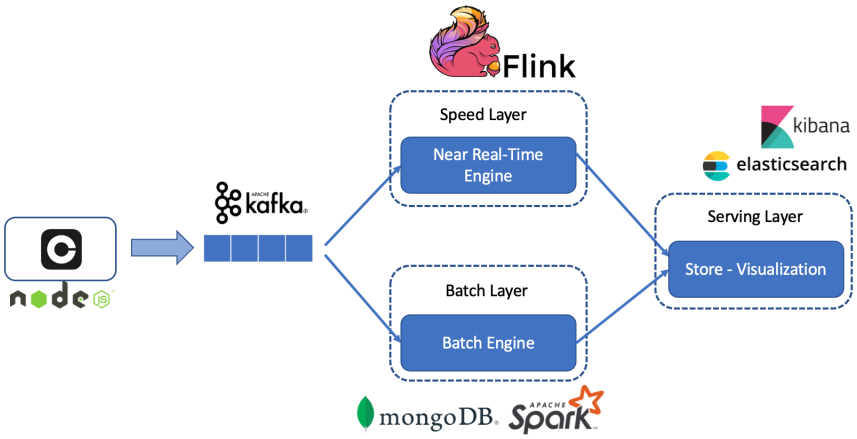


Figure 1: Lambda Architecture

4 Caso d'uso

La prima fase del progetto è stata la parte riguardante la scelta delle sorgenti informative. In particolare, si richiedeva l'utilizzo di sorgenti che garantissero un flusso continuo di dati e per questo si è scelto di utilizzare i dati provenienti dalla vendita e acquisto di BitCoin (ticker). Un exchange che mette a disposizione questo tipo di record è Coinbase Pro, una piattaforma di trading con diverse criptovalute (BTC, ETH, LTC, EOS...). I dati sono stati recuperati attraverso l'utilizzo delle API fornite dal sito stesso. Nonostante questo lavoro sia stato eseguito considerando una singola criptovaluta, il tutto potrebbe essere rieseguito analizzandone molteplici.
Si è quindi creato un producer kafka che attraverso l'utilizzo di Nodejs, recuperasse i dati provenienti da Coinbase Pro e li inserisse in un topic kafka. Dovendo gestire un'elaborazione batch e una (near) real time si è deciso di creare

due consumer in modo tale da disaccoppiare le velocità di processamento dei dati. Di sotto si riporta un esempio di struttura ottenuta attraverso il Producer kafka.


```
{"type":"ticker","sequence":12037843028,"product_id":"BTC-USD",
"price":"26504.67","open_24h":"26960.72",
"volume_24h":"941.58122430","low_24h":"26371.64",
"high_24h":"27223.71","volume_30d":"42454.67276770",
"best_bid":"26501.30","best_ask":"26504.67","side":"buy",
"time":"2021-07-16T10:52:24.828693Z","trade_id":46746298,
"last_size":"0.00058888"}
```


4.1 Speed Layer

Per questo livello si è deciso di utilizzare Apache Flink come framework di elaborazione, in quanto mette a disposizione numerosi strumenti per analisi (near) real-time. Questo rappresenta uno dei principali competitor di Apache Spark Streaming, framework approfondito durante il corso di Big Data. Si è scelto dunque di sperimentare un sistema non affrontato a lezione. Apache Flink fornisce un "connector" che gli permette di collegarsi al topic Kafka e consumare quindi i dati come subscriber. I dati ripresi da Kakfa, recuperati in formato Json, sono stati prima "parsati" per ottenere una struttura leggibile e poi attraverso un aggregazione con una finestra temporale di un minuto, è stato creato un modello OHLC estraendo i valori di *open*, *close*, *high*, *low* per ogni minuto (open rappresenta il prezzo di apertura dell'intervallo, close l'ultimo prezzo riportato, high il più alto ed infine low quello più basso). Questi dati sono stati infine inviati ad Elastichsearch per una visualizzazione tramite Kibana. Di seguito viene riportato un esempio di dato inviato da Flink ad Elastiseach:


```
{high=31738.22, low=31685.25, close=31730.19, open=31691.14,
timestamp=1.626653039675E12}
```


4.2 Batch Layer

Per quanto riguarda invece l'analisi batch, si è creato, attraverso l'utilizzo di Nodejs, un kafka consumer che riuscisse a recuperare i dati del topic e, attraverso l'instaurazione di una connessione con database MongoDB, li salvasse "as is" nel sistema di storage. Questi record sono stati in un secondo momento recuperati da Apache Spark SQL ed inviati, dopo varie elaborazioni, ad Elasticsearch. In particolare, le elaborazioni svolte sono state: recupero del prezzo minimo e massimo, calcolo della variazione percentuale, calcolo della distribuzione cumulativa, conteggio dei record salvati nel database. L'utilizzo di Spark SQL ha permesso di interrogare la collezione utilizzando una sintassi SQL. Questo servizio risulta molto intuitivo da utilizzare e configurare e grazie a questo è stato possibile creare un flusso che collegasse diverse applicazioni.

4.3 Serving Layer

L'ultimo step di questa pipeline ha riguardato l'interrogazione dei dati e la creazione di un'interfaccia grafica che permettesse la visualizzazione delle informazioni. A tal proposito si è deciso di utilizzare Elasticsearch e Kibana, framework che permettono di analizzare grandi volumi di dati, grazie alla loro scalabilità, offrendo una rappresentazione ottimale delle informazioni. In particolare, è possibile navigare tra i dati ed ottenere così informazioni riguardanti singoli intervalli temporali. Kibana implementa inoltre una dashboard interattiva che permette di organizzare, ridimensionare e modificare il contenuto di questa, senza la necessità di conoscere linguaggi di programmazione come CSS o HTML. E' possibile inoltre selezionare un intervallo temporale nel quale visualizzare i dati.

Dove possibile, sono state utilizzate immagini Docker per la creazione di container che potessero essere installate, eseguite e rimosse senza dover impostare ogni volta l'ambiente (ecosistema) di lavoro. Nello specifico sono state create *docker images* per Kafka e Zookeeper, Kafka producer, Kafka consumer, Elasticsearch e Kibana.
Di seguito si riporta un'immagine che rappresenta come i vari nodi siano stati utilizzati all'interno del progetto:

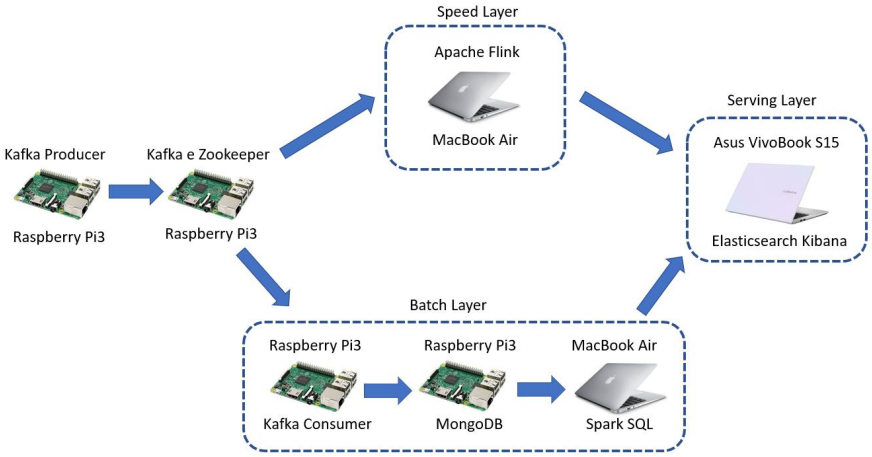


Figure 2: Cluster e Architettura

- Raspberry Pi3: Kafka Producer
- Raspberry Pi3: Kafka e Zookeeper
- Raspberry Pi3: Kafka Consumer
- Raspberry Pi3: Database MongoDB

- MacBook Air: Apache Flink e Spark SQL
- Asus VivoBook S15: ElasticSearch e Kibana

Nonostante il progetto sia stato eseguito su cluster, in ambiente distribuito, è presente sulla pagina github del progetto una repository che permetta di avviare il tutto localmente.

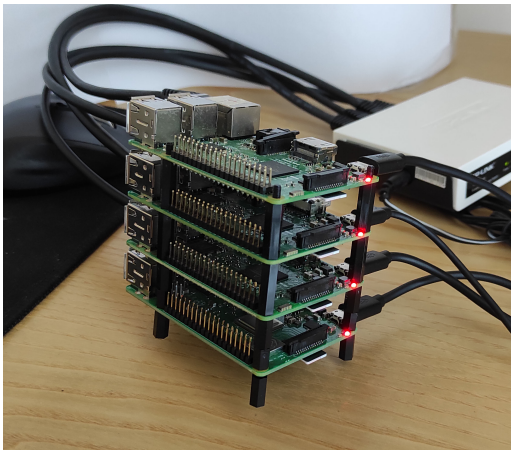
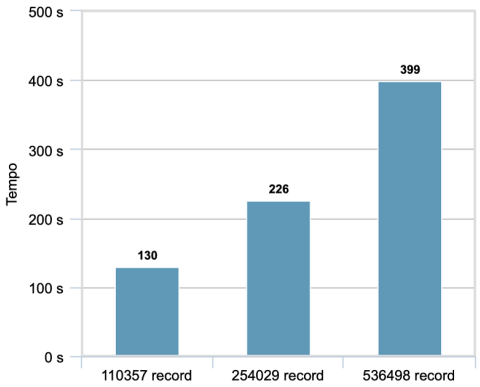


Figure 3: Quattro nodi del Cluster

5 Risultati

Una volta terminata la progettazione dell'architettura, il sistema è rimasto operativo per una durata di 66 ore, processando in totale 536.500 record circa.
Questi dati sono stati analizzati giornalmente, al fine di ottenere statistiche sui tempi di esecuzione di Apache Spark SQL, al crescere delle dimensioni dell'input.
Di seguito vengono riportati una tabella e un grafico con i tempi e numero di record processati: I tempi sopra riportati comprendo l'instaurazione di una connessione MongoDB, il recupero dei dati al suo interno, l'elaborazione e la scrittura su ElasticSearch.
Per quanto riguarda l'elaborazione near real time, i dati inviati a Flink sono stati aggregati in intervalli temporali di un minuto, ottenendo così 30519 record ed estraendo un modello OHLC.
La visualizzazione dei dati è stata affidata a Kibana, attraverso il quale è stato possibile creare una Dashboard interattiva che permettesse di navigare tra i dati e creare grafici e tabelle con diverse caratteristiche.
Alcuni esempi di grafici implementati sono riportati di seguito:



# Record	Tempo Esecuzione
110357	130 sec
254029	226 sec
536498	399 sec

Figure 4: Tempi Esecuzione

Variazione Percentuale

Prima Quotazione	Ultima Quotazione	Variazione Percentuale
Sat Jul 17 2021 18:25	Tue Jul 20 2021 13:08	-6.595

Figure 5: Variazione percentuale rispetto ai dati raccolti

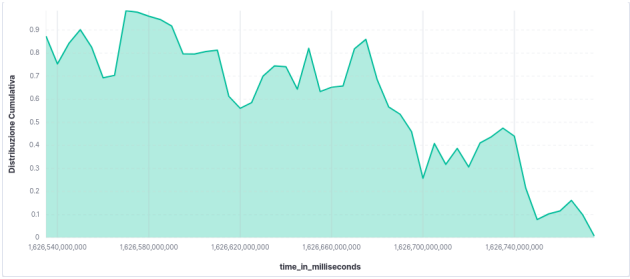


Figure 6: Distribuzione cumulativa

6 Conclusioni e sviluppi futuri

In questo progetto è stata presentata la realizzazione di un'Architettura Lambda che consentisse il monitoraggio e l'analisi di dati provenienti dalla criptovaluta Bitcoin. In particolare, la pipeline ha riguardato da una parte un'elaborazione batch (eseguita attraverso l'utilizzo di database MongoDB e framework Apache Spark SQL), dall'altra un'elaborazione real time (Apache Flink). Il tutto è stato poi reso disponibile attraverso sistemi come Elasticsearch e Kibana. Il primo per lo storage dei dati, il secondo per la visualizzazione. Nonostante i dati raccolti siano inerenti ad un intervallo temporale di soli tre giorni, è stato possibile eseguire statistiche ed analisi che riuscissero ad indagare l'andamento

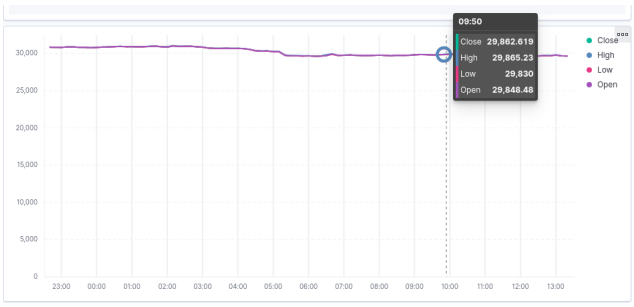


Figure 7: Andamento del bitcoin in tempo reale

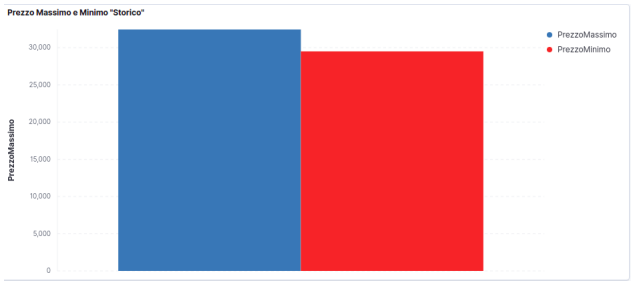


Figure 8: Prezzo minimo e massimo rispetto ai dati raccolti

del bitcoin. Con una quantità maggiore di dati a disposizione, uno sviluppo futuro potrebbe riguardare l'utilizzo di tecniche di Machine Learning in grado di prevedere l'evoluzione della criptovaluta nel tempo e fornire un migliore supporto alle decisioni.