

UNIVERSITÀ DEGLI STUDI DI MILANO-BICOCCA
DIPARTIMENTO DI INFORMATICA, SISTEMISTICA E COMUNICAZIONE



Large Scale Data Management
**Architettura per l'analisi di dati
provenienti da Reddit**

FRANCESCO PORTO 816042
MATTIA VINCENZI 860579

Anno Accademico 2020 - 2021

Indice

1	Introduzione	1
2	Possibili utilizzi	2
3	Architettura	3
4	Scryper	5
4.1	Estrazione dati da Reddit	5
4.2	Invio dei dati a Kafka	6
5	Apache Kafka	7
5.1	Event Sourcing	8
5.2	Organizzazione topic	9
5.3	Eventi pubblicati	10
6	Speed Layer	11
6.1	Spring consumer	11
6.1.1	Sentiment analysis	12
6.2	Elasticsearch	13
6.3	Kibana	15
6.3.1	Kibana Dashboard	16
7	Batch Layer	21
7.1	Batch consumer	21
7.2	MongoDB	22
7.3	Tableau	22
8	Deployment	31
9	Scalabilità	33
9.1	Obiettivi dell'analisi di scalabilità	33
9.2	Comparazione tra subreddit - Quantità dati	33
9.3	Metodologia di raccolta dati	36
9.4	Eperimento 1 - Capire i limiti dell'applicazione	37
9.5	Eperimento 2 - Capire la scalabilità dell'applicazione	38
9.5.1	Setup sperimentale	38
9.5.2	Confronto tra quantità monitorate	40

9.5.3	Richiamo sull'organizzazione delle partizioni	42
9.5.4	AVG Queue Time su Threads - Speed Layer	43
9.5.5	Queue Time e Processing Time su Threads - Batch Layer	44
9.6	Riepilogo dei bottleneck incontrati e possibili soluzioni	48
10	Conclusioni	50
	Bibliografia	51

1 Introduzione

Scopo di questo progetto è di realizzare una pipeline di analisi Big Data, avendo in tal modo la possibilità di testare le tecnologie presentate durante il corso di **Large Scale Data Management**.

Come sorgente Big Data si è deciso di utilizzare **Reddit** [1], ovvero un sito di social news, intrattenimento e forum, dove gli utenti registrati (chiamati **redditors**) possono pubblicare contenuti sotto forma di post (**threads**). Ad ogni thread può inoltre essere associato un **flair**, ovvero un tag che ne descrive la categoria di appartenenza (ad esempio il flair *Media* va ad indicare che il thread riguarda un video/immagine).

Gli utenti, inoltre, possono attribuire una valutazione (comunemente chiamati "**upvote**" e "**downvote**"), ai contenuti pubblicati: tali valutazioni determinano la posizione e visibilità dei vari contenuti. I contenuti del sito sono organizzati in aree di interesse chiamate **subreddit**.

E' stata implementata una pipeline di monitoraggio dati provenienti da un subreddit, in modo da avere sia una visione real-time dell'andamento di alcune metriche su uno specifico argomento di interesse (tra cui ad esempio: andamento di utenti attivi, sentiment dei threads, redditor con più upvotes ecc.) sia la possibilità di memorizzare dati per eseguirne un'analisi a posteriori.

Reddit fornisce delle **API** [2] che consentono di accedere ai propri dati. Questi ultimi in periodi temporali di lunga durata possono raggiungere volumi consistenti ed al contempo essere prodotti ad una velocità elevata vista la numerosità di utenti (o meglio redditor) utilizzatori di questa piattaforma. Per rendere l'analisi fattibile e sensata ci si è limitati alla raccolta dei dati provenienti da un subreddit.

In particolare si è preso come esempio */r/destinythegame* [3], ovvero un subreddit dedicato al noto gioco sparatutto online in prima persona prodotto dalla Bungie. Questo subreddit è stato scelto in quanto entrambi gli autori vi hanno familiarità, oltre al fatto che vengono forniti dati facoltativi (come ad esempio i flairs) che permettono un'analisi più approfondita. La pipeline può comunque essere riutilizzata per un generico subreddit, a patto di apportare alcuni piccoli cambiamenti.

Il codice è open source e disponibile all'indirizzo: <https://github.com/HopedWall/subreddit-analyzer>.

2 Possibili utilizzi

Durante la progettazione di questa applicazione si è cercato di andare oltre i semplici fini didattici e cercare di contestualizzarla in uno scenario realistico e concreto.

Si è immaginata una situazione in cui un’azienda metta in commercio un determinato prodotto e, contestualmente, crei un subreddit dedicato a raccogliere discussioni di qualsiasi genere relative a tale prodotto, spingendo gli utilizzatori a condividere opinioni su questo subreddit, divulgandone il link, ad esempio, per mezzo dei canali social.

Con l’apertura del subreddit viene anche deployata l’applicazione che implementa la pipeline di analisi dati. In questo modo l’azienda è in grado di monitorare real-time l’andamento dei threads/post creati nel subreddit durante l’intero arco delle settimane. Al contempo è possibile, ad esempio, con cadenza settimanale analizzare i dati memorizzati in maniera persistente per trarre insight e report che possano contribuire alla presa di decisioni in ottica futura. In tal modo è anche possibile confrontare l’andamento in periodi differenti, per poter confrontare l’effetto delle decisioni prese.

Questo esempio è di più facile concezione se si pensa ad un prodotto software, come potrebbe essere un videogioco (da qui la decisione del subreddit monitorato), che dopo l’uscita monitora l’opinione dei clienti. Dopo un primo breve periodo di monitoraggio se l’andamento del sentiment e il numero degli utenti attivi non sono soddisfacenti si può ricercare il problema tra i post e commenti e realizzare una patch/aggiornamento, e vedere se la situazione è migliorata.

3 Architettura

L'applicazione è basata su un'architettura lambda, in modo tale da perseguire un duplice scopo: da un lato il monitoraggio dell'andamento del Subreddit scelto, dall'altro la memorizzazione persistente dei dati per analisi future.

La **lambda architecture** è un pattern architetturale per la gestione di dati veloci e di grandi quantità, come ad esempio quelli provenienti da sensori o da social network. Questa soluzione viene adottata quando si ha la necessità di lavorare con dati real-time e di catturare insight, fare predizioni o ottenere subito dei risultati da essi ed allo stesso tempo di memorizzarli per fare batch processing.

In questa applicazione la parte superiore, comune, della lambda è costituita da Kafka, che rende poi disponibili ai due layer gli eventi per l'elaborazione; graficamente costituiti dalle due linee inferiori del simbolo (Figura 1).

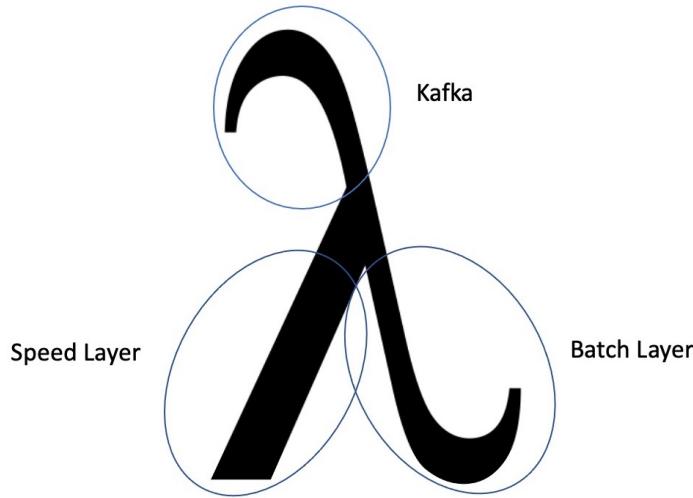


Figura 1: Rappresentazione grafica lambda.

Procedendo con ordine le componenti utilizzate nella realizzazione dell'applicazione sono quelle mostrate in figura Figura 2:

- **Scryper.** Applicazione scritta in Python che si occupa di reperire i dati da Reddit e di pusharli sotto forma di eventi su Kafka.

- **Apache Kafka.** Piattaforma distribuita per gestire streaming di eventi. Costituita da message broker che si basano su pattern di comunicazione asincrona publish-subscribe.
- **Speed Layer.** Layer costituito da un consumer che si occupa di ripetere dati da kafka ed eseguire semplici trasformazioni su di essi in aggiunta ad una predizione del sentiment. Successivamente i dati vengono caricati su **Elasticsearch** in modo che siano visualizzabili mediante una dashboard realizzata con **Kibana**.
- **Batch Layer.** Layer dedicato a memorizzare i dati in maniera persistente, in modo che possano essere analizzati a posteriori. Un consumer una volta ricevuti i dati da Kafka li salverà su **Mongo**. Infine, per eseguire analisi sulle informazioni in esso memorizzate è stato utilizzato **Tableau** (per analisi OLAP).

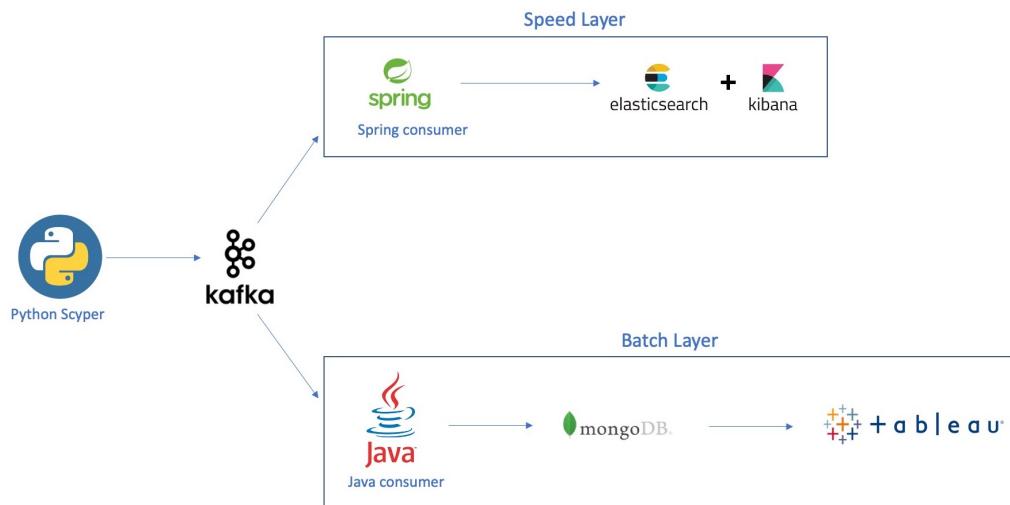


Figura 2: Overview dell'architettura.

4 Scryper

Lo **Scryper** è uno script Python che svolge sostanzialmente due funzioni:

1. Estrazione dei dati da Reddit
2. Invio dei dati su Kafka

4.1 Estrazione dati da Reddit

Per quanto riguarda l'estrazione dei dati, Reddit offre delle API REST [2], attraverso le quali è possibile ottenere diversi tipi di informazioni relative ai suoi contenuti (come ad esempio titolo e autore di un thread, numero e contenuto dei commenti di un redditor, ecc.).

Generalmente si preferisce evitare di utilizzare queste API direttamente, in favore di wrapper che ne semplificano l'utilizzo, sia in termini di leggibilità del codice che in termini di usabilità, offrendo diverse astrazioni. In Python il wrapper più famoso è sicuramente **PRAW (Python Reddit API Wrapper)** [4]. PRAW rende completamente trasparente al programmatore l'utilizzo delle API di Reddit (non è infatti necessario scrivere manualmente le richieste GET/POST) e utilizza un modello di astrazione che ricalca i concetti base di Reddit stesso, permettendo di trattare subreddit, threads e commenti come normali oggetti Python.

Per utilizzare PRAW è innanzitutto necessario registrare la app nella sezione "preferenze" del proprio profilo Reddit, per ottenere il secret da utilizzare in fase di connessione. Una volta fatto questo è sufficiente inserire questo secret nel file **praw.ini**, che dovrà poi essere posizionato nella root del progetto: sarà così possibile utilizzare tutte le potenzialità di PRAW.

Lo scryper utilizza PRAW per estrarre i dati relativi ai 10 thread più hot¹ del subreddit */r/destinythegame*. Per ognuno di questi thread vengono memorizzati:

¹Su Reddit, per *hot* si intende il rapporto tra il tempo trascorso dalla creazione di un thread e il numero di upvotes che ha ricevuto.

- i dati relativi al **thread** (id, autore, upvotes ecc.)
- i dati relativi ad ogni **commento** nel thread (id, autore, testo, ecc.)
- i dati relativi ad ogni **utente** che posta almeno un commento nel thread (id, nome utente, numero complessivo di upvotes ecc.)

Vengono inoltre memorizzati dati relativi al numero di **utenti attivi** e **subscribers** relativi al subreddit.

Ovviamente per monitorare un subreddit non è sufficiente estrarre i dati una sola volta, ma deve esserne effettuato un **aggiornamento periodico**. A tal scopo è stato necessario creare delle classi per la memorizzazione dei vari oggetti (threads, commenti e utenti) in locale, in modo da poter interrogare PRAW ad intervalli regolari per ricevere il dato più aggiornato.

Per garantire la reciprocità tra creazione e aggiornamento dei dati è stata utilizzata un'architettura **multi-threaded**. Più nello specifico sono stati utilizzati 3 thread²:

- **PostDownloader** che scarica i dati relativi ai 10 thread più hot.
- **PostUpdater** che aggiorna i dati relativi ai thread mantenuti in memoria, oltre che a quelli relativi a commenti e utenti.
- **SubredditUpdater** che scarica i dati relativi ad utenti attivi e iscritti.

La cooperazione tra questi thread permette sostanzialmente di avere in memoria lo stato corrente del subreddit. Il passo successivo è quello di inviare questi dati a Kafka, per permettere ai due layer di accedervi.

4.2 Invio dei dati a Kafka

Per interfacciare lo Scryper a Kafka è stata utilizzata la libreria **kafka-python** [5]. Questa libreria consente di definire oggetti **KafkaProducer** che permettono di scrivere su uno stream Kafka in maniera immediata. Come formato per la comunicazione è stato scelto **JSON** in quanto immediato da ottenere a partire da dizionari Python.

²In questo caso, la parola *thread* si riferisce al mondo dei sistemi operativi, e non a Reddit.

5 Apache Kafka

Apache Kafka è una piattaforma per il data streaming distribuito che permette di archiviare ed elaborare grandi flussi di dati in tempo reale. Più nello specifico permette di:

- **Pubblicare** e **leggere** stream di eventi, realizzando la comunicazione all'interno di applicazioni complesse o tra applicazioni diverse
- **Memorizzare** stream di eventi in maniera affidabile e duratura
- **Processare** eventi sia in **real-time** che in **batch**

Un **evento** può essere considerato come una "registrazione" di qualcosa che è accaduto. In Kafka, ad ogni evento è associata una **chiave**, un **valore** e un **timestamp**. Gli eventi sono organizzati in **topic**, e ad ogni topic sono associate una o più **partizioni**, che permettono a più **producer** e **consumer** di scrivere (leggere) allo stesso tempo.

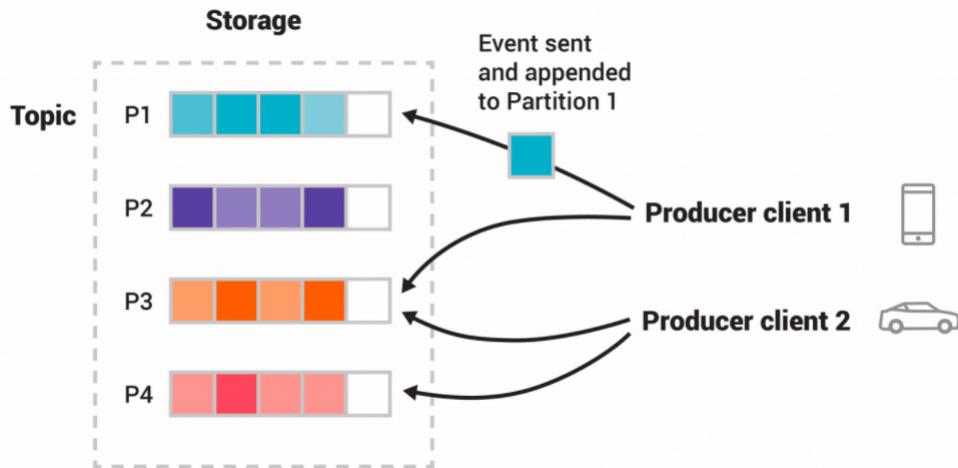


Figura 3: Funzionamento di Apache Kafka.

5.1 Event Sourcing

I dbms più comuni (sia relazionali che non-relazionali) tendono a memorizzare lo **stato corrente** di un sistema, in quanto è generalmente quello che interessa ad un'azienda per le sue attività. Questo approccio ha però il problema di rendere molto complessa la ricostruzione della **storia**, soprattutto per quanto riguarda possibili analisi a posteriori.

Per ovviare a questo problema sono stati creati alcuni *approcci alternativi*, per permettere di ricostruire la storia in maniera semplice e intuitiva. Uno dei più famosi è sicuramente quello ad **Event Sourcing** [6]. L'idea principale dell'Event Sourcing è quella di memorizzare stream di **eventi**, ad ognuno dei quali è associato un **timestamp** relativo alla sua emissione. Ognuno di questi eventi corrisponde ad un **cambiamento nello stato** di un sistema, un **fatto immutabile**: avendo un **evento iniziale** di creazione, e una serie di cambiamenti in ordine cronologico, diventa possibile sapere lo stato corrente del sistema in **ogni** istante temporale.

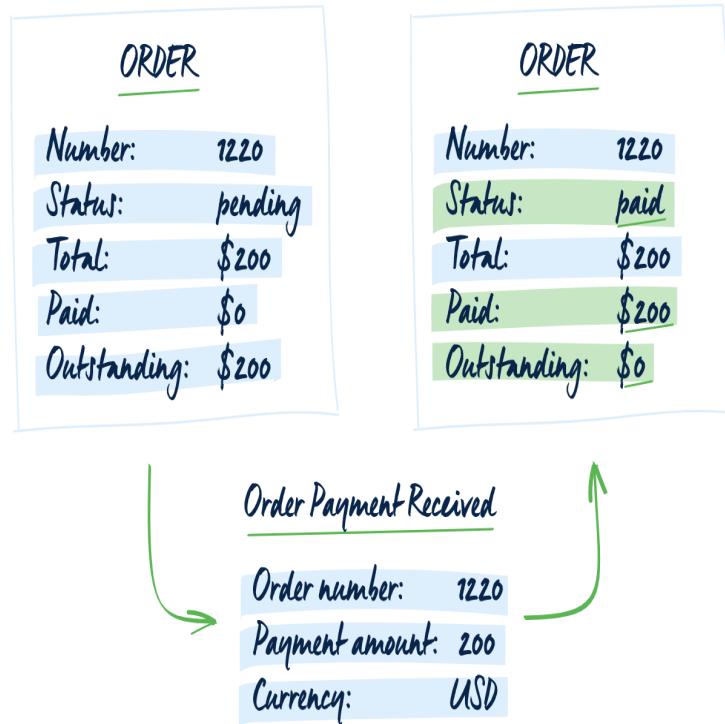


Figura 4: Esempio di Event Sourcing.

5.2 Organizzazione topic

Sono state considerate diverse strategie per l'organizzazione dei topic prendendo spunto da [6]. E' stata subito scartata la strategia **single topic**, in quanto sicuramente inadeguata per gestire eventi relativi ad oggetti diversi. Anche l'organizzazione **topic-per-entity** è stata scartata, in quanto creare un topic per ogni thread, commento e utente avrebbe portato a migliaia di topic diversi, che sarebbero impossibili da gestire nel pratico. Si è quindi adottata una strategia molto simile a quella **topic-per-entity-type**, e più nello specifico sono stati utilizzati 3 topic:

1. **threads**, che contiene eventi relativi ai thread e commenti (creazione e aggiornamento).
2. **users**, che contiene eventi relativi ai redditors (creazione e aggiornamento).
3. **subreddit-data**, che contiene i dati relativi agli utenti attivi e subscribed.

Ad ogni messaggio nel topic **threads** è associato come chiave il **thread-id**: questo permette di ricostruire l'evoluzione temporale del thread "percorrendo" lo stream da sinistra verso destra. Lo stesso ragionamento è stato applicato per il topic **users** usando lo **user-id** come chiave.

Si noti che non è stato creato un topic specifico per i commenti, ma si è scelto di inserirli nel topic **threads**, con chiave **thread-id** (e **non comment-id**): questo perché, come afferma Martin Kleppmann [7], nell'Event-Sourcing la priorità è **ordinamento** dei dati, e non la separazione logica tra topic diversi. Visto che ogni thread e i commenti ad esso relativi hanno la stessa chiave, diventa più facile avere un'idea chiara sulla progressione temporale di entrambi.

Per quanto riguarda il terzo topic sono state usate come chiave le stringhe "**subscribers**" e "**active-users**" per indicare il tipo di messaggio; ad ogni modo la scelta della chiave non è particolarmente rilevante in quanto tutti gli eventi si riferiscono allo stesso subreddit (*r/destinythegame*) e il timestamp è sufficiente per ricostruire la storia.

5.3 Eventi pubblicati

Per poter ricostruire la storia in maniera efficace, si è deciso di distinguere gli eventi in tipi diversi, ad ognuno dei quali è associato un valore del campo **type** nel documento JSON che rappresenta il messaggio. I possibili valori di questo campo sono:

1. **Thread-create** e **Thread-update**, per segnalare rispettivamente la creazione e l'aggiornamento di un thread.
2. **Comment-create** e **Comment-update**, per segnalare rispettivamente la creazione e l'aggiornamento di un commento.
3. **User-create** e **User-update**, per segnalare rispettivamente la creazione e l'aggiornamento di uno user
4. **Subscribers** e **Active users**, per segnalare rispettivamente il numero di subscribers e il numero di active-users

Si noti che i tipi di messaggio sono specifici per ogni topic, in quanto topic diversi rappresentano concetti differenti.

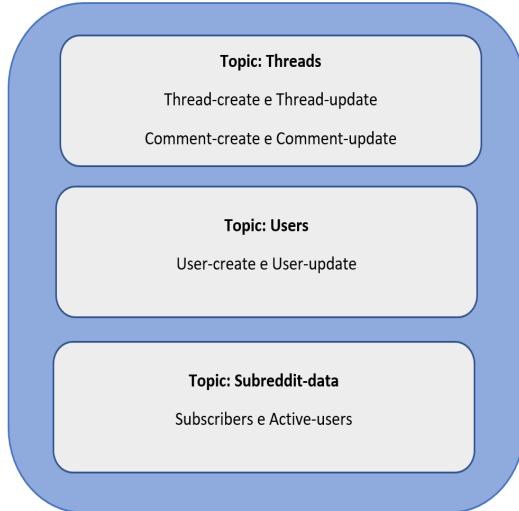


Figura 5: Topic utilizzati e i rispettivi tipi di messaggio.

6 Speed Layer

Lo **Speed Layer** si occupa di ricevere costantemente gli eventi pubblicati sui topic di kafka, di elaborarli, eseguendo una semplice operazione di predizione del sentimento, per poi depositarli su Elasticsearch in modo che questi siano reperibili e visualizzabili con Kibana. Il tutto deve essere fatto in maniera real-time in modo da favorire il monitoring dello stato attuale del subreddit analizzato.

Questo layer è costituito da una serie di componenti: uno **Spring consumer**, **Elasticsearch** ed infine **Kibana**, ognuno dei quali sarà descritto più nel dettaglio.

6.1 Spring consumer

Sfruttando il noto framework java open-source **Spring** si sono realizzati dei servizi "consumatori" dei messaggi pubblicati sui tre topic di Kafka. Per prima cosa, una volta ricevuti, ad ognuno di essi viene aggiunto come campo al file JSON il timestamp che kafka attribuisce come metadato al momento della ricezione di ogni evento. Successivamente ogni messaggio viene passato ad un handler differente a seconda del topic da cui è stato ricevuto per una rapida elaborazione e adattamento di formato. A parte qualche piccola differenza nel trattamento dei campi interi, o di conversione, tutti i messaggi vengono poi inseriti nei diversi indici di Elasticsearch.

Per quanto riguarda il thread **subreddit-data** gli eventi gestiti sono:

- **active-users**. Tale evento denota il report degli utenti attivi in un determinato istante temporale.
- **subscribers**. Tale evento denota il report degli utenti iscritti al subreddit monitorato in un determinato istante.

In entrambi i casi i messaggi vengono caricati in un indice di **Elasticsearch**, in modo da poter ricostruire in base al timestamp una serie temporale. In entrambi i messaggi è presente un campo **type** che discrimina se l'aggiornamento è relativo agli utenti attivi o agli utenti iscritti.

Per quanto riguarda il thread **Threads** gli eventi gestiti sono:

- **post-create**. Tale evento denota la creazione di un nuovo post. Una volta aver ottenuto il messaggio, a questo viene estratto il campo testuale, in quanto non interessante al fine di monitoring. Tale campo

viene però preprocessato e su di esso viene eseguita l'**analisi del Sentiment** (Sezione 6.1.1), in modo da arricchire il messaggio con la polarità del post.

- **comment-create.** Tale evento rappresenta la creazione di un commento in un determinato post/thread. Nello Speed Layer questo evento viene utilizzato per aggiornare il numero di commenti relativi ad un post.
- **post-update.** Tale evento rappresenta una modifica nel numero di **up-votes** associati ad un post (sia in positivo che in negativo). Dall'evento viene estratto l'id del post a cui l'aggiornamento fa riferimento per utilizzarlo come chiave di ricerca ed aggiornare il campo con il nuovo numero di voti.

Per quanto riguarda il thread **Users** gli eventi gestiti sono:

- **user-create.** Tale evento rappresenta la creazione di un utente/redditor. Viene inserito nel corrispondente indice il file JSON rappresentante dell'utente.
- **user-update.** Tale evento rappresenta l'aggiornamento del numero di **upvotes** totalmente raccolti da ogni utente, ossia dai post e dai commenti da lui creati.

6.1.1 Sentiment analysis

L'analisi dell'orientamento dei post è una pratica molto diffusa e utilizzata in tutti i campi di analisi testuale. Per eseguire questa tecnica si è optato per l'utilizzo di un modello preaddestrato che prende il nome di **VADER** [8] (Valence Aware Dictionary and sEntiment Reasoner). VADER è un modello rule-based basato su un dizionario (*lexicon*) che associa ad ogni parola un punteggio.

E' stato utilizzato VADER in quanto è un modello adatto all'analisi di testi brevi, provenienti dai social network, in quanto nel lexicon comprende anche emoji, abbreviazioni e slang.

Per ogni frase passata a questo modello viene ritornato un dizionario con quattro punteggi: positivo, negativo, neutro e un compound (un punteggio riassuntivo della polarità della frase).

Prima di utilizzare VADER sono state eseguite una serie di semplici operazioni di preprocessing sul testo utilizzando la libreria java CoreNLP [9], pubblicata da un gruppo di ricercatori della Stanford. Per quanto riguarda l'analisi del sentimento, invece, è stata utilizzata una libreria java [10] che funge da wrapper a VADER implementato nella libreria python NLTK.

I passi effettuati per arrivare ad eseguire la sentiment analysis sono quelli mostrati in Algoritmo 1.

Algorithm 1 Sentiment analysis

```

1: procedure PREPROCESSTEXTANDSENTIMENT(post)
2:   post_compound  $\leftarrow$  0
3:   for sentence in post do:
4:     cleaned  $\leftarrow$  remove_stopwords(stopword_list, sentence)
5:     sentence_compound  $\leftarrow$  vader(cleaned)
6:     post_compound  $\leftarrow$  post_compound + sentence_compound
7:   if post_compound  $>$  0 then return pos
return neg

```

Dal momento che VADER è adatto per analizzare frasi brevi ed i post possono essere piuttosto lunghi questi, per prima cosa, vengono scomposti nelle singole frasi (line 3 - **Splitting**). Successivamente vengono rimosse le **stopwords**, ovvero l'insieme delle parole che contribuiscono in piccola parte al significato della frase (solitamente sono circa il 30%). Per eseguire questa rimozione viene prima scomposta la frase nelle singole parole/token (**tokenization**) e ognuna di queste viene rimossa se è presente in una lista di stopwords reperita su GitHub (line 4 - **cleaning**). Ognuna delle frasi ottenute dalla fase di cleaning viene data in pasto a VADER (line 5 - **sentiment**) e dei punteggi restituiti viene sommato quello riassuntivo di compound al punteggio globale del post.

Infine, una volta ripetuto l'algoritmo per ogni frase del post e sommati i vari punteggi di compound, se il punteggio globale di compound del post è maggiore di 0 allora viene restituita l'etichetta "**pos**" (che rappresenta polarità positiva), altrimenti "**neg**" (polarità negativa) (line 7).

6.2 Elasticsearch

Elasticsearch è un motore di ricerca e di analytics distribuito e gratuito per qualunque tipo di dati: testuali, numerici, geospatiali, strutturati e non strutturati.

turati. E' il componente principale di **Elastick Stack**, ovvero un set di strumenti gratuiti per l'acquisizione, l'arricchimento, l'archiviazione, l'analisi e la visualizzazione dei dati [11].

Questo strumento è noto per la sua natura distribuita, la velocità, la scalabilità e le sue semplici API REST. Mediante queste API è possibile infatti modificare ed interrogare i dati indicizzati in Elasticsearch, al cui interno le informazioni sono gestite come documenti JSON.

Per memorizzare i dati in un cluster Elasticsearch vengono utilizzati gli indici, che sono definiti come una raccolta organizzata di documenti JSON. Ogni documento a sua volta è una raccolta di campi, ognuno dei quali rappresenta una coppia chiave valore contenente i dati [12].

Ad alto livello è possibile vedere un indice come un database in un database relazionale, ovvero un meccanismo di organizzazione dei dati [13].

Visti gli aggiornamenti apportati in Elasticsearch 7.0 e i suggerimenti riportati in [14] allora si è optato per la creazione di un indice per ogni tipo di documento. Facendo riferimento ai topic da cui vengono ricevuti gli eventi in Kafka, descritti in Sezione 5.2, allora sono stati creati tre indici:

- **Subreddit-data-json.** Memorizza gli eventi di aggiornamento del numero di utenti iscritti e attivi. In ognuno di questi eventi è presente un campo per la discriminazione del tipo di evento ed uno relativo al timestamp di creazione, in modo che sia possibile da essi riscostuire gli andamenti temporali.
- **Threads-data-json.** Memorizza tutti gli eventi relativi ai post e ai commenti ad essi correlati. Ai soli fini di monitoring non è stato necessario memorizzare il testo dei post e dei commenti. Infatti, su questo indice è possibile trovare memorizzati i post e per ognuno di essi il numero di commenti ed il sentiment ad esso correlati.
- **Users-data-json.** Memorizza tutti gli utenti che hanno creato post o commenti con i relativi dati e il numero di **upvotes** complessivamente ricevuti.

In tutti e tre i casi lo **Spring Consumer** descritto nella sezione precedente si occuperà di andare a caricare i dati nei corretti indici a seconda del topic da cui sono stati ricevuti.

6.3 Kibana

Kibana è un'applicazione frontend open source che lavora in cima all'Elas-tic Stack, consentendo la ricerca e la visualizzazione dei dati indicizzati in **Elasticsearch**. Inoltre, Kibana funge anche da interfaccia utente per il monitoraggio e la gestione di un cluster Elastick Stack [15].

Per realizzare la maggior parte dei grafici è stata utilizzata una funzio-nalità di Kibana chiamata **Lens**, che fornisce un'interfaccia intuitiva per la visualizzazione dati [16]. **Lens** permette di realizzare grafici mediante drag-and-drop, consentendo di trarre insights da enormi quantità di dati senza possedere conoscenze a priori sull'utilizzo di Kibana.

In questa applicazione Kibana è stata utilizzata per creare grafici di vario tipo per il monitoraggio del Subreddit di interesse. I grafici realizzati sono di vario genere: da histogrammi per il conteggio degli upvotes e dei commenti per post, a diagrammi a torta per la percentuale di utilizzo dei flair.

Per lavorare con Kibana è stato necessario accedere all'applicazione web reperibile su localhost alla porta 5601: **localhost:5601**.

Una volta configurata Kibana indicando dove è possibile reperire i dati di Elasticsearch (mediante variabili d'ambiente all'avvio del container), è stato necessario configurare gli **index pattern**.

Un **index pattern** identifica uno o più indici di Elasticsearch che si vogliono esplorare con Kibana. Nella definizione di questi pattern è possibile utilizzare dei wildcards, in modo che Kibana possa ricercare dati su tutti gli indici di Elasticsearch che mettano quel pattern [12] [17].

Per quanto riguarda la dashboard di questa applicazione non si aveva la necessità di ricercare dati tra indici differenti con un singolo index pattern. Gli index-pattern definiti sono quindi tre, che permettono la ricerca e visualizzazione dei dati nei corrispondenti tre indici di Elasticsearch:

- **Subreddit-data-json***.
- **Threads-data-json***.
- **Users-data-json***.

6.3.1 Kibana Dashboard

La dashboard realizzata è quella che si può vedere in figura Figura 6. I dati visualizzati sono quelli raccolti su un arco temporale di 16 ore, in modo da avere a disposizione una quantità di dati rilevante.

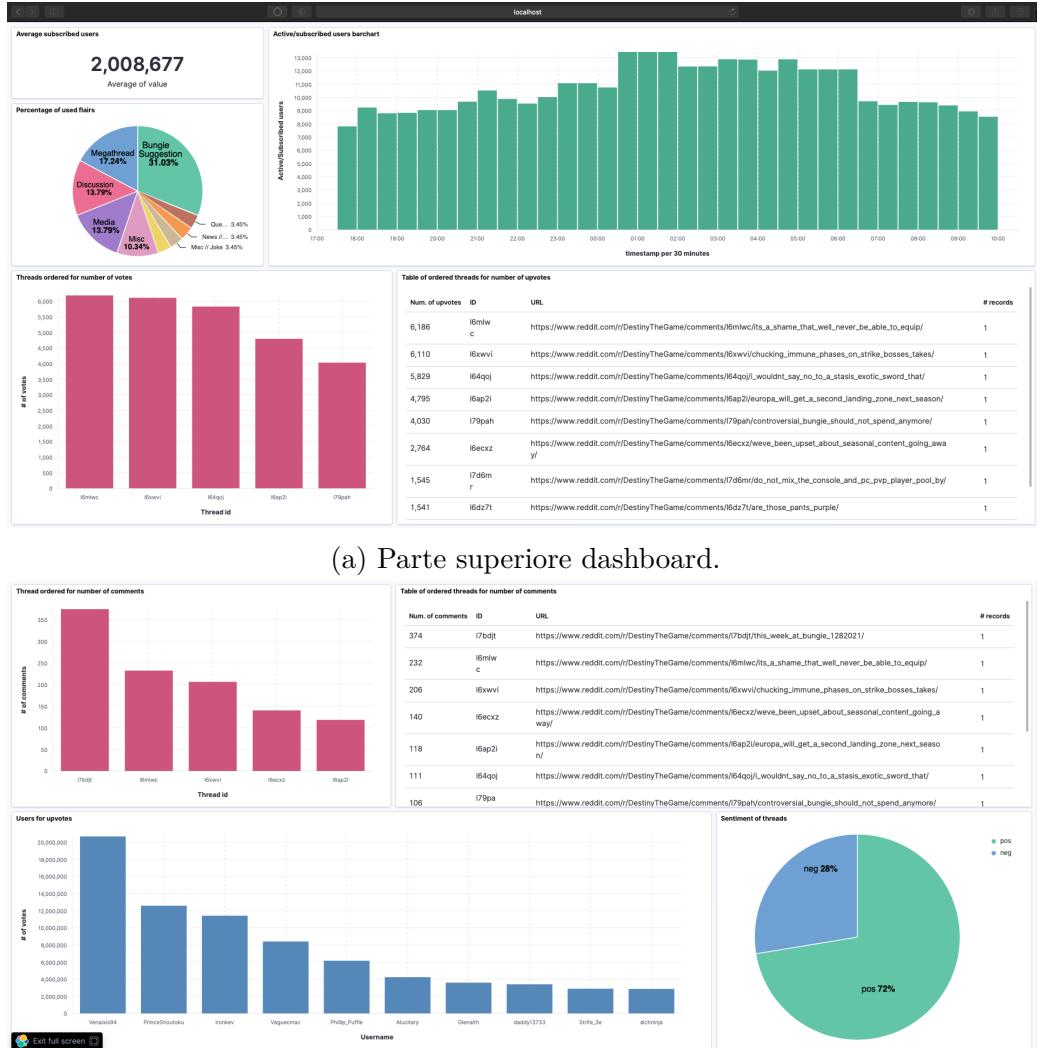


Figura 6: Dashboard completa.

Andando più nel dettaglio i singoli grafici che è possibile trovare all'interno della dashboard sono i seguenti:

- In Figura 7 viene riportato il grafico che mostra l'andamento temporale del numero di utenti attivi, sull'intero arco di monitoraggio delle 16 ore.

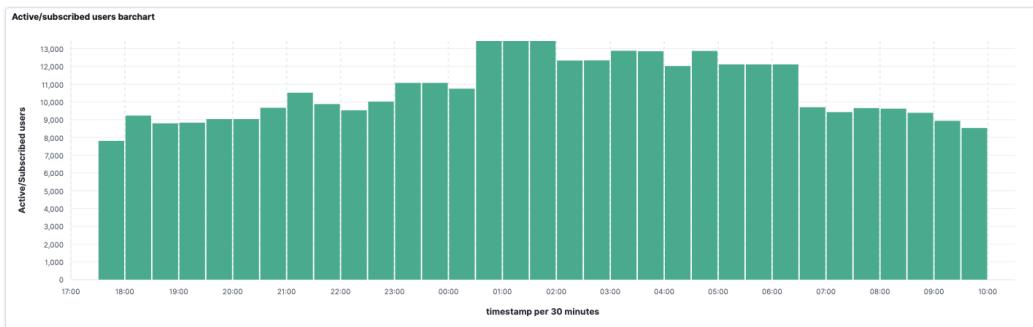


Figura 7: Serie temporale rappresentante gli utenti attivi nell'arco del periodo monitorato.

- In Figura 8 viene riportato il numero medio di utenti iscritti nell'arco temporale monitorato.



Figura 8: Numero medio di utenti iscritti nell'arco temporale monitorato.

- In Figura 9 viene riportata la distribuzione di probabilità sui **flair** utilizzati nei post tracciati.

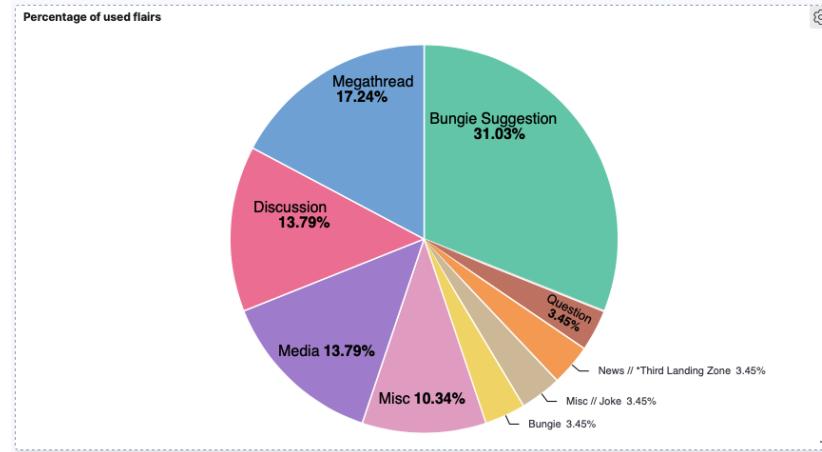


Figura 9: Percentuale di utilizzo dei flair.

- In Figura 10 viene riportato sulla sinistra l'istogramma dei post ordinati per numero di **upvotes** ricevuti, con indicato sulle X il corrispondente ID. Per consentire una più semplice lettura, sulla destra, viene riportato un menù che ad ogni ID associa il numero di commenti e l'URL a cui poter reperire il post.



Figura 10: Post ordinati per **upvotes** e tabella consultabile per ottenere il corrispondente URL.

- In Figura 11 viene riportato sulla sinistra l'istogramma dei post ordinati per numero di **commenti** ricevuti. Anche in questo caso è presente una tabella, identica a quella del punto precedente.



Figura 11: Post ordinati per **numero di commenti** e tabella consultabile per ottenere il corrispondente URL.

- In Figura 12 viene riportato l'istogramma degli utenti ordinati per numero di **upvotes** complessivamente ricevuti.

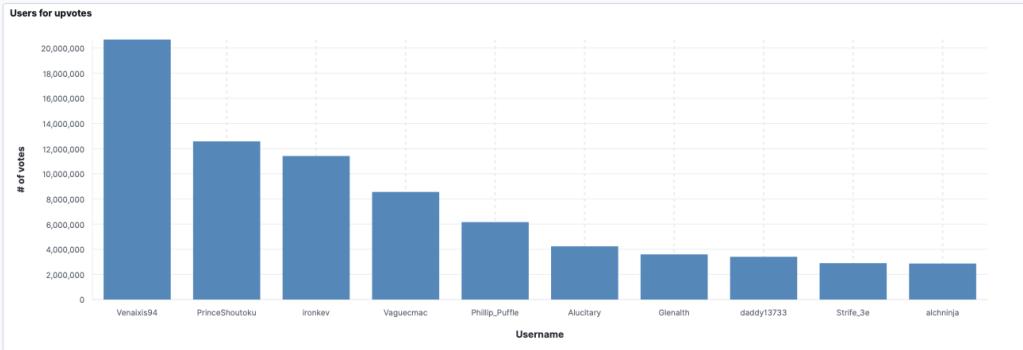


Figura 12: Utenti ordinati per numero di **upvotes** ricevuti complessivamente tra post e commenti pubblicati.

- In Figura 13 viene mostrata la distribuzione del **sentiment** dei post monitorati.

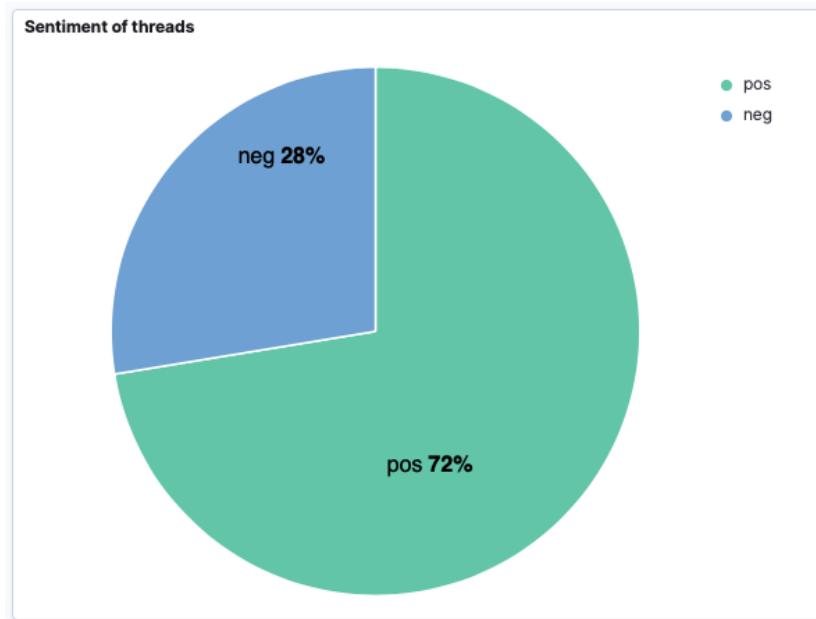


Figura 13: Distribuzione del sentiment dei post monitorati.

7 Batch Layer

Il **Batch Layer** si occupa dell’analisi dei dati a posteriori, offrendo all’utente la possibilità di effettuare sessioni **OLAP** (On-Line Analytical Processing) per analizzare particolari aspetti ritenuti utili per scopi di business. Più nello specifico, questo layer è costituito da 3 componenti:

1. **Batch consumer**, che si occupa di consumare gli eventi pubblicati sui topic di Kafka e di memorizzarli in maniera persistente.
2. **MongoDB**, un database non-relazionale di tipo documentale che viene utilizzato per memorizzare in maniera persistente i dati rilevanti provienti da Kafka.
3. **Tableau**, un software di analisi e business intelligence che permette di effettuare sessioni OLAP ai dati di MongoDB attraverso il **MongoDB BI Connector**.

7.1 Batch consumer

Il **Batch consumer**, scritto in Java, è un’applicazione **multi-threaded** che si occupa di consumare i tre stream (threads, users e subreddit-data) e di memorizzarne i dati in maniera persistente su MongoDB. Prima di essere memorizzati come documenti di MongoDB, ai vari messaggi viene aggiunto il **timestamp** e per quanto riguarda threads e commenti viene aggiunto il **sentiment**, come specificato in Sezione 6.1.1.

Il Batch consumer e lo Speed consumer³ presentano alcune differenze:

- Nello Speed consumer **non** vengono memorizzati i **testi** di thread e commenti, nel Batch consumer invece sì; questa scelta è dovuta al fatto che potrebbero essere interessanti ai fini di analisi.
- Nel Batch consumer vengono memorizzati **tutti** gli stati intermedi per ogni oggetto: questo diventa necessario in quanto per fare analisi OLAP questi dati sono necessari. Sarebbe stato interessante far connettere Tableau direttamente a Kafka, ma ad oggi un tale connettore non sembra essere disponibile.

³Nome alternativo per lo Spring consumer (descritto in Sezione 6.1).

7.2 MongoDB

MongoDB è un database NoSQL di tipo documentale, che negli ultimi anni ha acquisito un'enorme popolarità grazie alla sua flessibilità e semplicità di utilizzo. Questo database è stato scelto in quanto i documenti sono in formato JSON, lo stesso che viene utilizzato su Kafka; inoltre, grazie allo **sharding automatico**, è in grado di scalare in maniera efficiente per grandi quantità di dati. Con riferimento al *CAP theorem*, MongoDB è **Consistent** (tutti i nodi vedono gli stessi dati allo stesso momento) e **Partition Tolerant** (il sistema continua a funzionare anche in caso di perdite di messaggi o malfunzionamenti).

I dati sono stati organizzati in 3 collections:

- **post-collection**, che contiene i dati relativi ai threads e ai relativi commenti.
- **user-collection**, che contiene i dati relativi agli utenti.
- **subreddit-data-collection**, che contiene i dati relativi agli utenti attivi e subscribed.

E' stata usata la tecnica dell'**embedding** per modellare la relazione tra thread e relativi commenti: questo perché si tratta di entità fortemente accoppiate. Per quanto riguarda la relazione autore-thread e autore-commento è stato invece usato il **referencing**, in quanto sarebbe stato eccessivo replicare lo stesso utente (soprattutto visto che un utente può avere scritto migliaia di commenti!).

7.3 Tableau

Tableau è un software di analisi e business intelligence che permette di effettuare sessioni OLAP e fornisce connettori per molti tipi di database. E' un software commerciale, ma è possibile ottenere una licenza education sul sito ufficiale, che include il software Tableau, Tableau Prep ed il software di ETL della stessa compagnia.

Tableau non supporta nativamente un connettore per MongoDB, ma è possibile scaricare il connettore **MongoDB BI Connector** al seguente indirizzo: <https://docs.mongodb.com/bi-connector/v2.0/installation/>.

Una volta installato, è necessario eseguire il seguente comando:

Listing 1: Generazione dello schema

```
mongodrdl --host localhost -d reddit_data -o schema.drdl
```

Viene così generato uno schema per il database (o la collection) selezionato nel file **schema.drdl**. A questo punto con il comando:

Listing 2: Esposizione del servizio

```
mongosqld --schema schema.drdl --mongo-uri localhost
```

Tableau sarà in grado di raggiungere le collections di MongoDB connettendosi a **localhost:3307**. L’interfaccia di Tableau è molto intuitiva, e permette all’utente di creare visualizzazioni semplicemente trascinando **dimensioni** e **misure** sullo schermo.

Sono state create alcune visualizzazioni d’esempio, il cui obiettivo è quello di effettuare un’analisi su dati storicizzati per estrarre informazioni aggiuntive rispetto a quelle presentate sulla dashboard. Tableau fornisce inoltre la possibilità di esportare i dati su Excel e di creare presentazioni personalizzate.

Più nello specifico, Tableau fornisce 3 tipi di visualizzazioni:

- **Worksheets**, che permettono di visualizzare un singolo grafico.
- **Dashboards**⁴, che permettono di raggruppare più worksheets.
- **Stories**, che permettono di visualizzare l’andamento delle dashboard nel tempo.

Di seguito vengono presentate alcune delle visualizzazioni che sono state create.

⁴In questo caso le dashboard di Tableau non sono state utilizzate come quella di Kiabana per monitoraggio real-time, ma bensì solamente per raccogliere più worksheet in un’unica visualizzazione.

<Numero di thread a cui un utente ha partecipato>

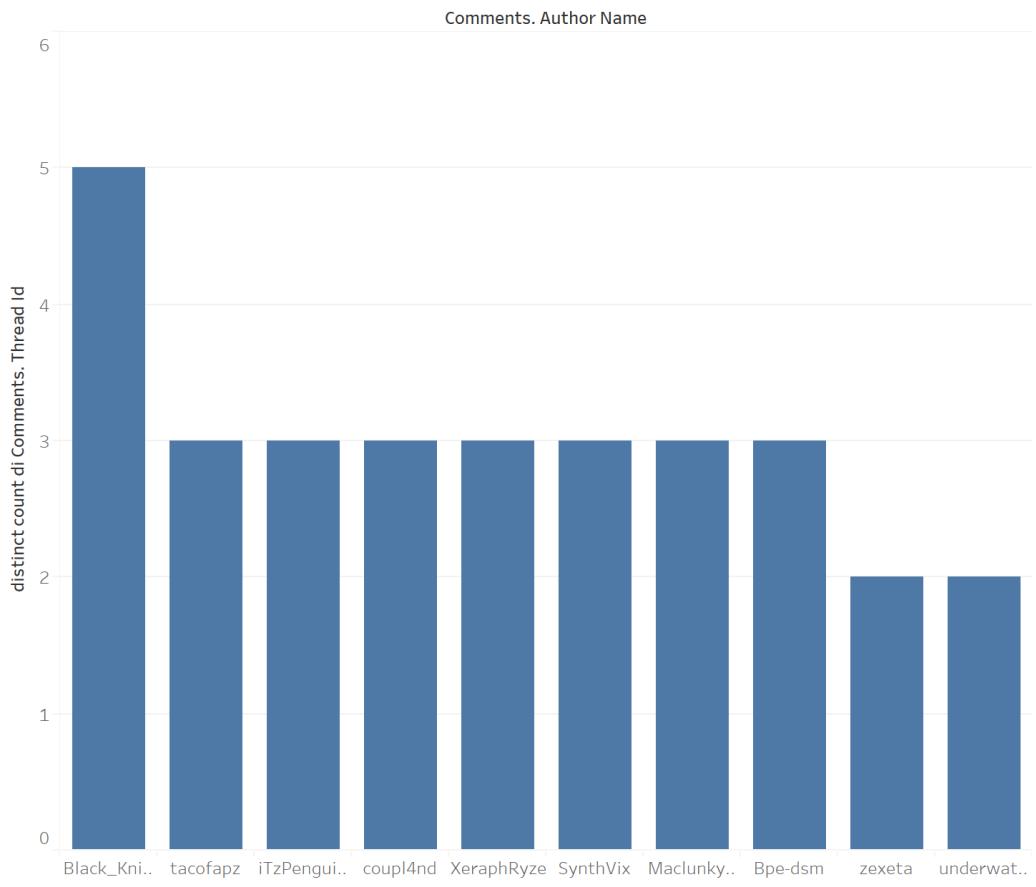


Figura 14: Primi 10 utenti per partecipazione a thread hot.

<Differenza percentuale rispetto
all'ora precedente di utenti attivi e
subscribers>

Hour of Tim..	User type	
	active_users	subscribers
9		
10	2,901%	0,001%
11	-1,892%	0,001%
12	-3,711%	0,001%

% Difference in Avg. Value broken down by User type vs.
Timestamp Hour.

Figura 15: Variazione percentuale di subscribers e utenti attivi, per ora.

<Polarity dei 10 commenti con più upvotes>

Comments. Text	Comments. Author Name	Comments.Polarity
You might say that mnk users *recoil* from that change.	TheDecafDude	
Edit: thanks for awards and updots. So bad for my dad joke disorder lol.		
I was using an SMG today and realized it effectively has zero recoil on PC.	BaconIsntThatGood	
For real. I love how they nerfed swords strictly because of usage.	TheWagn	
All other heavies are kiiinda trash bar heavy GLs...		
If you play crucible a lot put on the ghost mod that gives you a chance to get an enhancement prism after a precision kill. I've already gotten like 30 just from week or so of play, which I then bought ascendant shards with.	lordkinsanity	
r/DestinyTheGame... overreacting??? WHAT???	Nyoomfist	
Wait. They are removing gunsmith weeklys ? How will we get upgrade modules without buying them or using season pass ?	Siberia_Rite_Epok	
While "buff everything else" is a fun sentiment -- and I don't disagree that those identified legendary weapons need buffs -- Swords were an objective outlier that needed a reduction...	Menirz	
It is once again time to jump off the cliff 10 games in a row each week the armor is the engram. Less than 3 min per game that way and you never see the other team. In and out, no salt.	I_am_Rezix	
Ngl good riddance. Hoarding bounties really made streamers who spent all their time on the game powerful.	I_LIKE_THE_COLD	
..		
Absolutely agree with you but reddit isn't the place for common sense and patience	ShatteredMemories_	

Comments.Polarity (color) broken down by Comments. Text and Comments. Author Name. The view is filtered on Comments. Text, which keeps 10 members.

Figura 16: Polarity dei primi 10 commenti in base al numero di upvotes.



Figura 17: Dashboard sulla polarità dei thread hot.

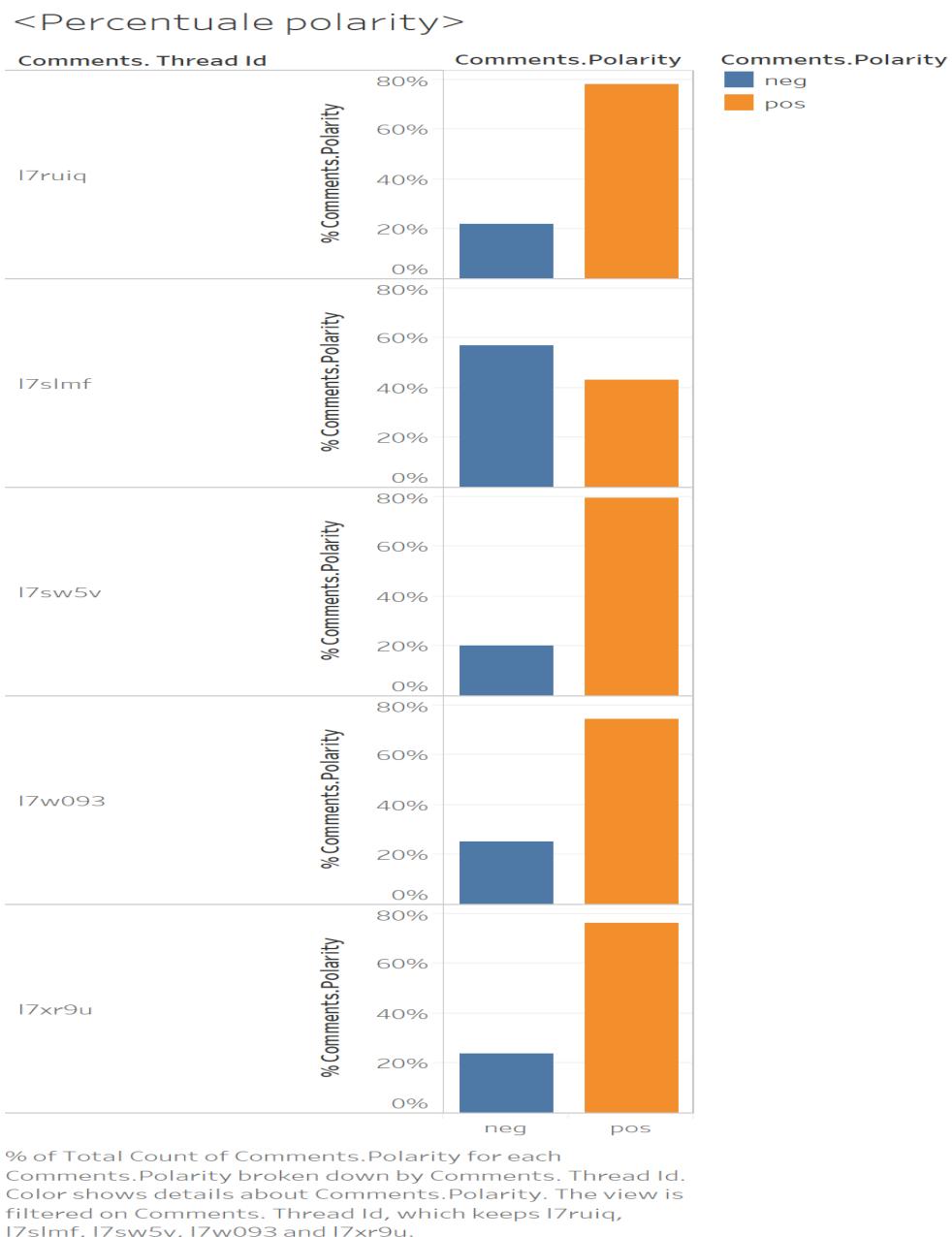


Figura 18: Distribuzione della polarità dei commenti per i primi 5 thread, ordinati sulla base del numero di commenti.

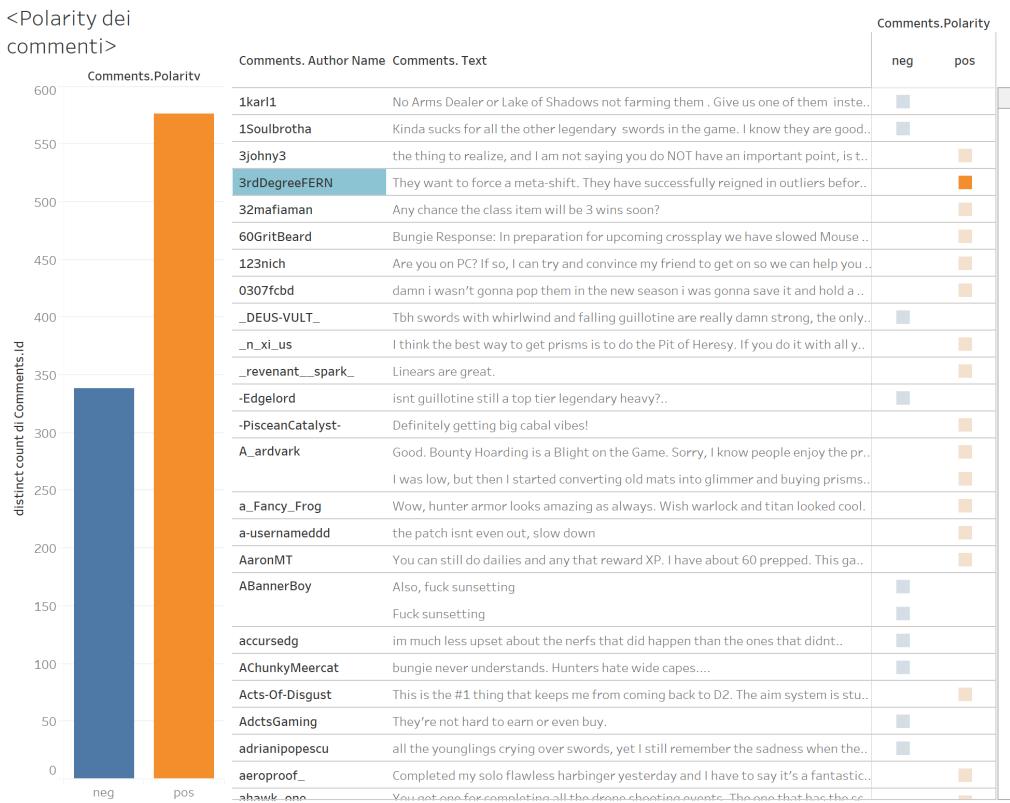
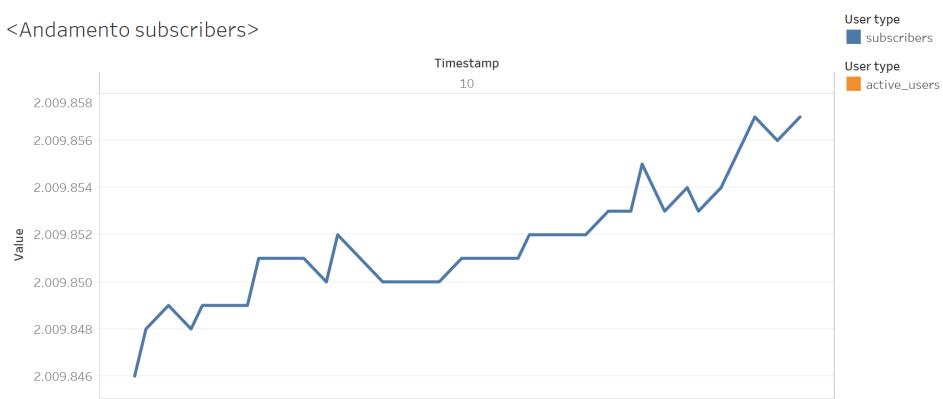


Figura 19: Dashboard sulla polarità dei commenti.

<Evoluzione per ora>



<Andamento subscribers>



<Andamento active users>

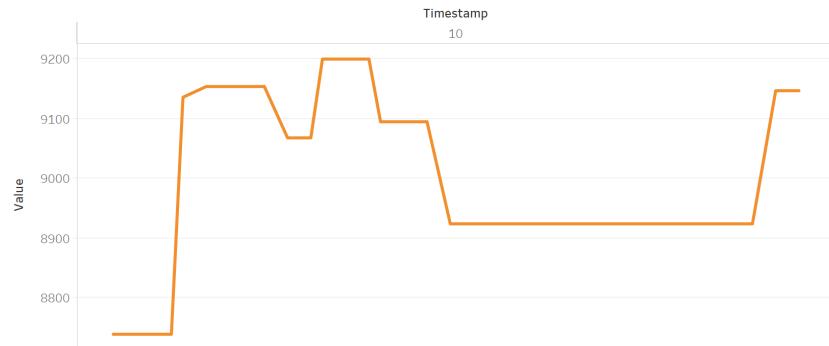


Figura 20: Storia per la visualizzazione dell'andamento di subscribers ed utenti attivi.

8 Deployment

Per rendere l'applicazione di facile utilizzo e distribuzione si è optato per l'esecuzione dei database e dei servizi di terze parti in container docker. Sin dalle prime fasi di sviluppo dell'applicazione Kafka, Kibana e i database quali Mongo ed Elasticsearch sono stati eseguiti scaricando le immagini dal Docker Hub, in modo da evitare l'installazione esplicita di tali tool e favorire la creazione di un ambiente di testing in maniera rapida.

Una volta sviluppati e testati i due consumer e lo scryper, sono stati anch'essi dockerizzati. Per ognuno di questi componenti è stato infatti scritto un **Dockerfile** per la creazione delle rispettive immagini.

Infine, per evitare l'esecuzione manuale dei container e avere un metodo idempotente per eseguire e deployare tutti i servizi si è utilizzato il tool **Docker Compose**. A tale scopo si è scritto uno YAML file che consente di definire in maniera dichiarativa quella che è la configurazione dei servizi (volumi, variabili d'ambiente, network ecc.). Docker compose tradurrà tale file nei rispettivi comandi docker in maniera trasparente all'utente.

Accedendo alla repository di GitHub è possibile trovare nella root directory il file **docker-compose.yml**. Per deployare localmente l'intera applicazione sarà necessario eseguire semplicemente il comando:

Listing 3: Comando per il deployment dell'applicazione

```
docker-compose up -d
```

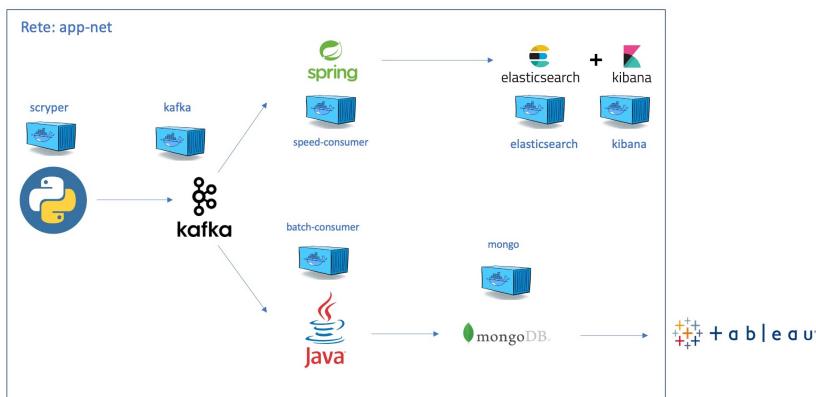


Figura 21: Piano di deployment, ad ogni componente è affiancato il nome del servizio nella docker net.

Per dockerizzare l'intera applicazione è stato necessario andare a modificare in tutti i file le stringhe di connessione, sostituendo **localhost** con i nomi dei container connessi alla stessa docker net (passabili come variabili d'ambiente). In questo modo il servizio di DNS distribuito del Docker Engine può eseguire la traduzione dei nomi nei corrispondenti indirizzi IP. In Figura 21 è possibile vedere i nomi utilizzati per i container.

9 Scalabilità

In seguito al feedback ricevuto nell'appello di Febbraio, il progetto è stato modificato in modo da permettere un'analisi accurata in termini di scalabilità. Nelle sezioni successive saranno discusse nel dettaglio le modifiche apportate, la metodologia utilizzata per il monitoraggio e i risultati ottenuti. I dati e il notebook utilizzato per l'analisi sono disponibili a [questo indirizzo](#).

9.1 Obiettivi dell'analisi di scalabilità

L'analisi di scalabilità si propone due obiettivi principali:

1. **Capire la quantità di dati gestiti dell'applicazione in uno scenario verosimile.** Per fare questo il subreddit monitorato è stato modificato da `destinythegame` a `wallstreetbets`, recentemente reso popolare dal caso GameStop. Questi subreddit sono stati quindi confrontati per avere un'idea chiara delle differenze tra i due in termini di quantità dati.
2. **Esaminare come l'applicazione sia in grado di scalare con carichi applicativi più o meno grandi.** Per prima cosa il codice è stato modificato per permettere la visualizzazione di diverse metriche relative a tempi e ritardi. In seguito si sono analizzate e confrontate queste metriche per capire come esse variano con carichi applicativi di diversa grandezza. Queste analisi sono state condotte al fine di individuare possibili **bottleneck** e capire come poterli risolvere.

9.2 Comparazione tra subreddit - Quantità dati

Nella precedente discussione dell'applicazione era stato preso come riferimento il subreddit `destinythegame`, in quanto era familiare ad entrambi i componenti del gruppo. Per quanto riguarda questa seconda analisi, si è scelto di utilizzare `wallstreetbets`, diventato popolare per il caso GameStop. Come si può vedere in Figura 22, `wallstreetbets` ha circa 4.5 volte il numero di utenti di `destinythegame`. Un comportamento simile si può notare anche per il numero di commenti e utenti monitorati.

	destinythegame	wallstreetbets
Subscribed users as of 27/03/2021	2'038'100	9'670'987
Monitored threads (updates) ~ 6 hour	210 (16883)	196 (24042)
Monitored comments (updates) ~ 6 hour	2516 (18214)	11933 (38792)
Monitored users (updates) ~ 6 hour	2084 (72176)	8631 (97134)

Figura 22: Tabella riepilogativa del confronto tra subreddit.

Nei grafici riportati in Figura 23, Figura 24, Figura 25 e Figura 26 è possibile vedere l’andamento del numero di thread, utenti e commenti tracciati dal sistema messi a confronto per i due subreddit.

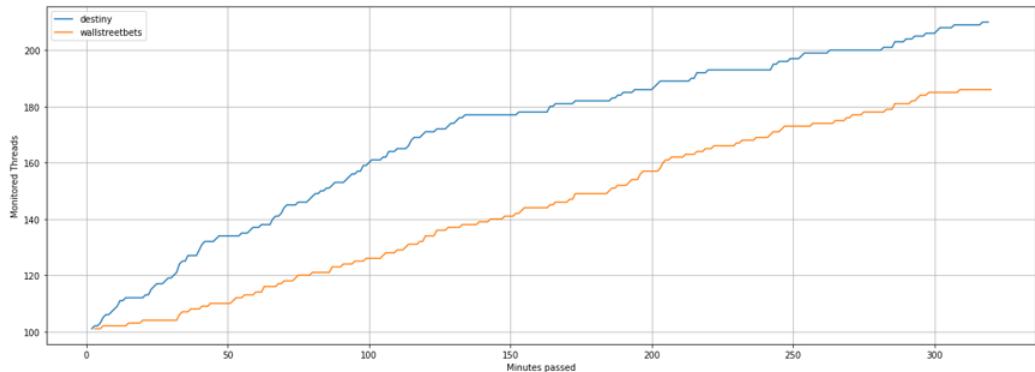


Figura 23: Confronto tra numero di thread tracciati nei due subreddit.

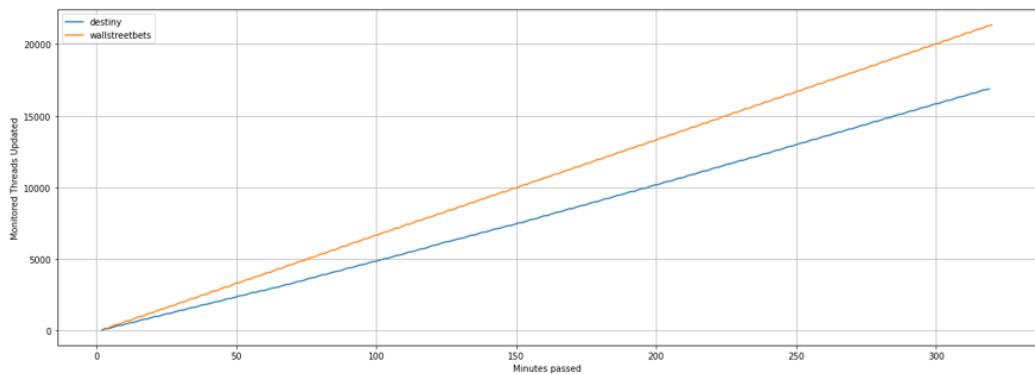


Figura 24: Confronto tra numero di thread aggiornati nei due subreddit.

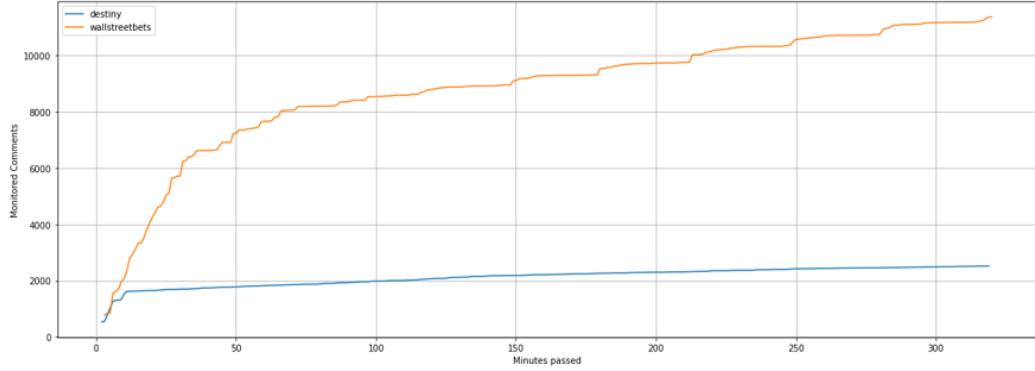


Figura 25: Confronto tra numero di commenti tracciati nei due subreddit.

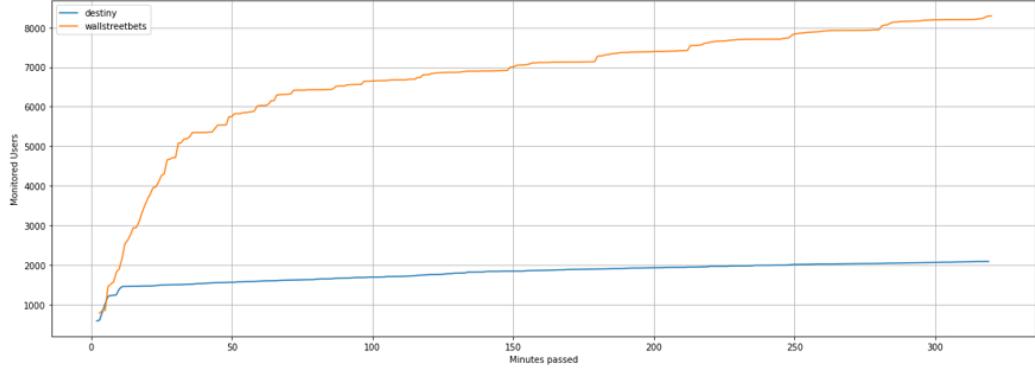


Figura 26: Confronto tra numero di utenti tracciati nei due subreddit.

L'unica eccezione è il numero di thread monitorati, come si può vedere anche in Figura 23, che è leggermente superiore in destinythegame. Per spiegare questa differenza è necessario ricordare che ad ogni iterazione lo Scryper riceve dalle API i dati relativi ai **primi 100** thread/post hot. Un numero di thread creati maggiore sta semplicemente ad indicare una maggiore **variabilità** nei thread hot; detto in un altro modo, un thread rimane hot per meno tempo. Per i thread ha quindi più senso considerare il dato relativo agli aggiornamenti, e come si vede questo è molto maggiore rispetto a quanto visto in destinythegame.

9.3 Metodologia di raccolta dati

Per raccogliere i dati necessari ad effettuare le analisi di scalabilità è stato necessario apportare diverse modifiche al codice pre-esistente. L'obiettivo di queste modifiche era quello di avere a disposizione, per ciascuno dei componenti dell'architettura, diverse metriche relative alla **quantità di dati** gestiti e ai tempi impiegati per elaborarli.

Per quanto riguarda lo **Scryper**, vengono stampati, una volta al minuto, i dati relativi al numero di thread, commenti e utenti **creati e aggiornati** fino a quell'istante. Per quanto riguarda **Speed** e **Batch Consumer** vengono visualizzate diverse informazioni relative ad ogni messaggio ricevuto:

- **Message type:** il tipo di messaggio.
- **Sent time:** il timestamp che rappresenta il momento in cui il messaggio è stato inviato a Kafka dallo Scryper.
- **Received time:** il timestamp che rappresenta il momento in cui il messaggio è stato ricevuto dal consumer.
- **End Consumer Processing Time:** il timestamp che rappresenta il momento in cui il messaggio è pronto ad essere memorizzato sul database.
- **End Database Operation Time:** il timestamp che rappresenta il momento in cui il messaggio è stato processato dal database.

Nella tabella riportata in Figura 27 è possibile vedere un esempio di quelli che sono i dati di analisi raccolti dai consumer per ogni messaggio ricevuto.

Message Type	Sent Time	Received Time	End Consumer Processing Time	End Db Operation Time
post-update	1616001154911	1616001154928	1616001154942	1616001154963
comment-create	1616001154934	1616001154978	1616001154984	1616001155009
post-update	1616001155086	1616001155332	1616001155342	1616001155381
comment-create	1616001154967	1616001155384	1616001155396	1616001155429

Figura 27: Esempio di dati di analisi di un Consumer.

Sulla base di questi dati, sono state calcolate delle metriche ritenute rilevanti ai fini dell'analisi:

- **Queue Time**: tempo per il quale il messaggio rimane in coda. Ottenuto dalla differenza tra Received Time e Sent Time.
- **Consumer Processing Time**: tempo impiegato dal consumer per preparare il messaggio all'inserimento sul database. Ottenuto dalla differenza tra End Consumer Processing Time e Received Time.
- **Database Operation Time**: tempo impiegato dal database per memorizzare il messaggio. Ottenuto dalla differenza tra End Database Operation Time e End Consumer Processing Time.

9.4 Esperimento 1 - Capire i limiti dell'applicazione

Ai fini di capire i limiti dell'applicazione si è optato per l'utilizzo di una macchina virtuale con caratteristiche simili a quelle di un computer di uso comune, e più nello specifico:

- **CPU**: Intel Xeon E5-2660 4x2.195 GHz.
- **RAM**: 8 GB.

Come numero di thread monitorati (iniziali) è stato usato il valore 100. Dopo circa 4 ore la VM ha iniziato a diventare molto lenta, per poi crashare completamente verso le **5 ore**. Analizzando l'utilizzo di memoria, riportato nella parte superiore di Figura 28, si può vedere che questi fenomeni avvengono in corrispondenza dell'azzeramento della RAM libera. Sempre allo stesso istante temporale, nella parte inferiore, si può inoltre notare un aumento considerevole dell'utilizzo della CPU, probabilmente dovuto al fenomeno dello **swapping**. Difatti, nel momento in cui gli 8 Gb di RAM terminano, l'*iowait* della CPU aumenta in maniera consistente.

Si può quindi concludere che **8 GB di RAM non sono sufficienti** per far funzionare l'applicazione per periodi di monitoraggio prolungati, sarebbe quindi più opportuno testare l'applicativo su sistemi con un quantitativo di RAM maggiore.

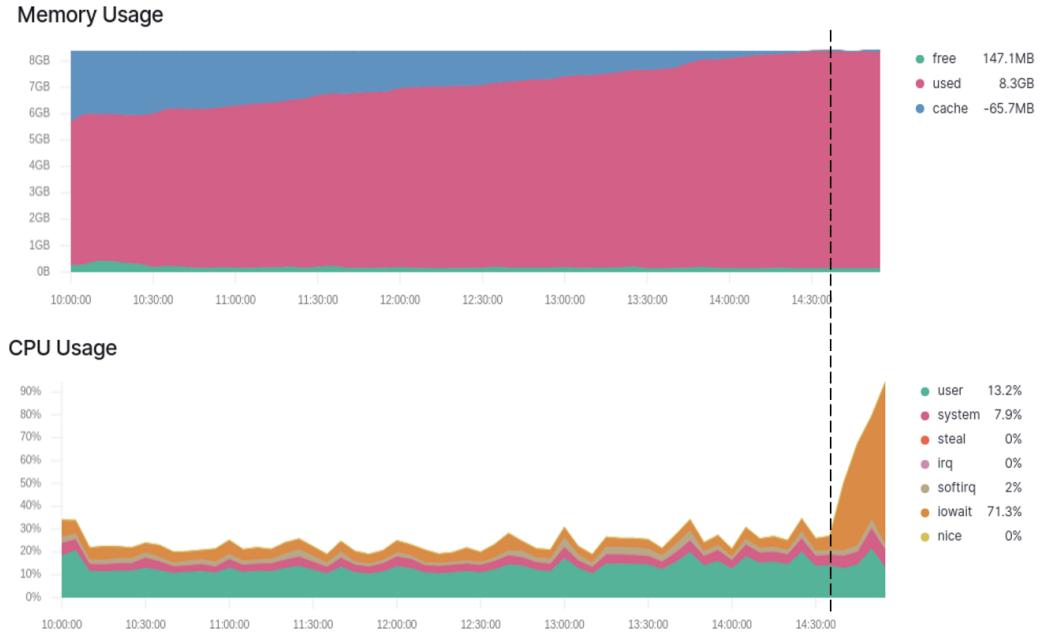


Figura 28: Occupazione RAM e CPU nell'esperimento 1.

9.5 Esperimento 2 - Capire la scalabilità dell'applicazione

9.5.1 Setup sperimentale

Ai fini di testare la scalabilità dell'applicazione si è optato per l'utilizzo di una macchina virtuale più performante rispetto alla precedente, in modo tale da condurre l'esperimento senza intoppi o crash inaspettati.

Le caratteristiche rilevanti della macchina virtuale sono le seguenti:

- **CPU:** Xeon Platinum 8171M 4x2.095 GHz.
- **RAM:** 16 GB.

Tali caratteristiche, in particolare la RAM maggiore, hanno consentito di realizzare esperimenti di durata consistente senza sperimentare comportamenti inaspettati e consentendo la ricerca agevole dei *bottleneck* dell'applicazione.

Con l'obiettivo di analizzare la scalabilità dell'applicazione si è testata la stessa sottponendola a diversi carichi di utilizzo. L'approccio mantenuto

come riferimento, e leggermente adattato allo specifico problema, è stato quello di eseguire diversi test con carichi man mano crescenti e ottenere i tempi trascorsi dai messaggi/eventi nei vari step della pipeline.

L'applicazione è stata quindi sottoposta a carichi crescenti: prima il 25%, poi il 50%, 75% ed infine 100% dei dati. Sono quindi stati ripetuti 4 test separati della stessa, con carichi di partenza differenti. Le differenti percentuali di carico fanno riferimento al carico iniziale di lavoro e quindi al numero di post monitorati in partenza, i quali però durante il tempo di monitoring tenderanno inevitabilmente ad aumentare. Questo è dovuto al fatto che l'applicazione, sfruttando le API di Reddit, mantiene la lista aggiornata dei post più hot, che nel tempo per forza di cose tenderà ad evolvere. Il tutto implica che al termine del tempo di monitoraggio, il numero di post risultati come tracciati dal sistema, sarà maggiore rispetto a quello di partenza.

Supponendo un carico massimo pari ai 100 post più hot monitorati, gli esperimenti effettuati sono i seguenti:

- 25% - 25 post di partenza.
- 50% - 50 post di partenza.
- 75% - 75 post di partenza.
- 100% - 100 post di partenza.

Ognuno dei 4 esperimenti è stato condotto per una durata di circa 15 ore; tale intervallo è stato predisposto considerando le caratteristiche della VM a disposizione, al fine di evitare saturazione delle risorse come nel primo caso di studio riportato.

Avendo a disposizione due macchine virtuali, con le medesime risorse, i 4 esperimenti sono stati eseguiti in parallelo a coppie di due. Si è ritenuto rilevante anche considerare due aspetti che inizialmente si erano lasciati in secondo piano, ma che attraverso varie esecuzioni sono risultati di rilievo:

- L'intervallo temporale in cui sono stati eseguiti gli esperimenti, seppur su due giornate diverse, è stato sempre lo stesso.
- Si è scelto un intervallo temporale significativo considerando il fuso orario degli USA. In particolare:
 - Orario Italiano: 18:00-8:00.

- Orario Americano: **13:00-3:00**.

Queste due scelte sono state dettate dal fatto che Reddit ha la maggior parte dell'utenza, o comunque quella più attiva, su territorio Americano.

Per ognuno dei 4 esperimenti sono state calcolate le metriche presentate in Sezione 9.3 per ciascuno dei due layer. **Dal momento che la differenza nel numero di messaggi sui tre topic può essere consistente, le metriche sono state analizzate distintamente per ogni topic e per ogni layer, portando ad avere 18 casi di analisi differenti.**

Sin dal principio il topic **subreddit data**, contenente aggiornamenti sul numero di utenti attivi e iscritti, è stato escluso dall'analisi, in quanto il numero di messaggi in esso contenuti è esiguo rispetto agli altri.

Tutti i casi testati, con i relativi grafici, sono consultabili nel notebook. Di seguito questa trattazione si concentrerà solo su quelle che sono state le analisi che hanno consentito di trarre gli insight più significativi rispetto ai limiti dell'applicazione.

Solo una menzione viene fatta ad esempio per i tempi medi di esecuzione delle operazioni sul database nello speed layer, questi, trattandosi solo di un semplice inserimento, sono tempi pressoché costanti, con grafici che, a meno di qualche picco poco interessante, sono poco significativi. D'altro canto, alcuni tempi sono estremamente bassi, con differenze dell'ordine di neanche un millisecondo, il che ha portato ad escluderli da un'analisi approfondita per via del fatto che la metodologia utilizzata per la loro raccolta potrebbe essere poco affidabile per quelli inferiori alla decina dei millisecondi, come riportato in [18].

9.5.2 Confronto tra quantità monitorate

Per prima cosa è doveroso riportare quelli che sono i differenti carichi in termini di numero di post, commenti e utenti nei 4 esperimenti eseguiti. Di seguito i grafici riportati in Figura 29, Figura 30 e Figura 31 mostrano l'andamento nel tempo (circa 15h), delle entità tracciate dal sistema. Come ad esempio si può vedere in Figura 29 il numero di post monitorati tende ad aumentare per ognuno dei 4 esperimenti (le 4 righe presenti nel grafico), e l'aumento sarà più accentuato per quelli di carico superiore (ad esempio l'esperimento con il 100% del carico passa da 100 a circa 270 thread/post tracciati, mentre quello con il 25% da 25 a circa 55). Questo è dovuto al fatto che considerando un maggior numero di thread **hot**, alcuni di quelli

considerati avranno importanza minore e di conseguenza verranno scavalcati più frequentemente da altri nel ranking dei post durante il tempo di monitoraggio. Questo si traduce quindi in un maggior numero di post tracciati dal sistema.

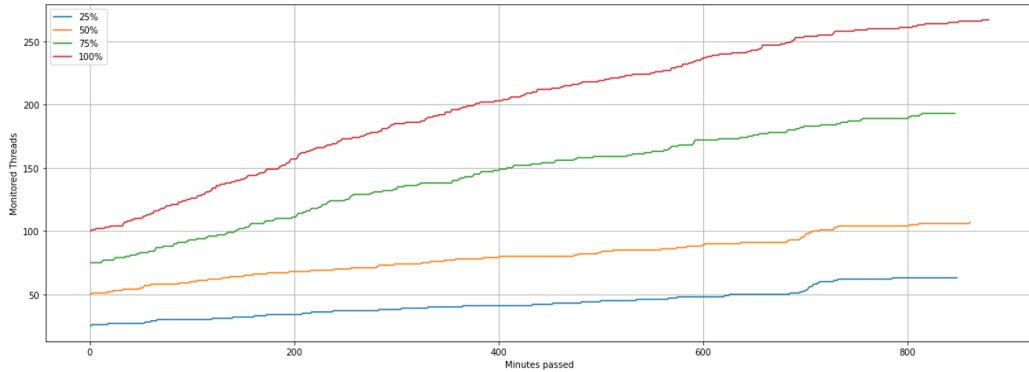


Figura 29: Andamento del numero di post/thread, tracciati durante i vari esperimenti, nel tempo.

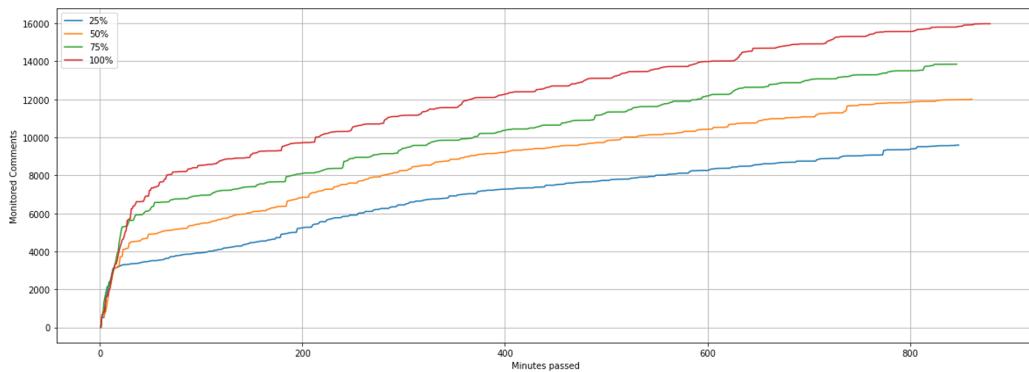


Figura 30: Andamento del numero di commenti, tracciati durante i vari esperimenti, nel tempo.

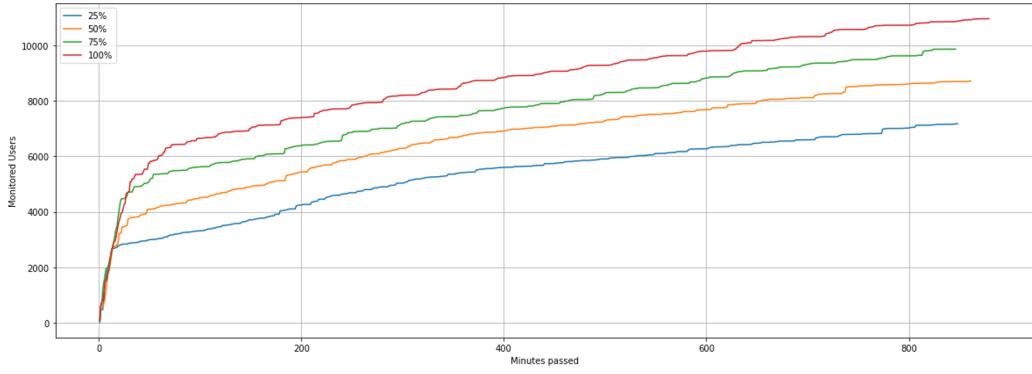


Figura 31: Andamento del numero di utenti, tracciati durante i vari esperimenti, nel tempo.

9.5.3 Richiamo sull'organizzazione delle partizioni

Di seguito verranno analizzati i tempi che hanno permesso di trarre gli insight più significativi, ma per comprendere al meglio quelle che sono state le considerazioni fatte, è importante fare un piccolo richiamo a come vengono distribuiti i messaggi sulle partizioni in Kafka.

Le **partizioni** sono gruppi logici di messaggi correlati, ed in Kafka i messaggi sono strutturati come coppie chiave valore. Questo punto è importante in quanto tutti i messaggi con la medesima chiave finiranno nella medesima partizione.

Tale meccanismo è stato utilizzato per etichettare tutti i messaggi relativi ad un post con chiave l'id del post stesso e, al contempo, anche quelli relativi ai commenti di quel post sempre con la medesima chiave. In tal modo tutti i messaggi attinenti ad un post ed ai suoi commenti verranno accodati nella stessa partizione. Scorrendo una partizione da sinistra verso destra sarà quindi possibile ricostruire l'intera storia di un post e dei suoi commenti.

Quando il numero delle chiavi supera il numero delle partizioni, messaggi con chiavi differenti inizieranno a condividere la medesima partizione. Sulla medesima partizione si raccoglieranno messaggi relativi a più post con i relativi commenti.

In questo progetto Kafka è stato configurato per avere 100 partizioni per ogni topic.

9.5.4 AVG Queue Time su Threads - Speed Layer

In questa sezione viene analizzato quello che è il **Queue Time**, ovvero il tempo trascorso in coda su Kafka da un messaggio. Ai fini dell'analisi viene calcolato quello che è il tempo medio in coda su tutti i messaggi di un determinato topic. Nello specifico, il tempo qui analizzato è quello relativo all'**AVG Queue Time** per il topic **threads** per quanto concerne lo **speed layer**.

Nella tabella riportata in Figura 32 si può vedere il tempo medio trascorso in coda dai messaggi nei vari esperimenti.

AVG Queue time - topic Threads	
25%	261.10 ms
50%	160.41 ms
75%	148.43 ms
100%	168.27 ms

Figura 32: Speed Layer - Threads topic - AVG Queue Time.

Guardando i valori sopra riportati ci si può accorgere che il tempo in coda per il primo esperimento, e quindi con un numero inferiore di thread monitorati, è superiore ai successivi, il che a primo impatto potrebbe sembrare contro intuitivo.

Nell'esperimento con il 25% dei dati verranno monitorati inizialmente i primi 25 thread più hot. Considerando che ai messaggi relativi allo stesso post e ai commenti ad esso associati viene associata come chiave l'id del post, questi finiranno nella medesima partizione. Proprio per questo ragionamento, avendo a che fare con un basso numero di post, un piccolo numero di partizioni sul totale disponibili verrà occupato, le quali però risulteranno maggiormente congestionate. In altre parole, si avrà un grande numero di messaggi concentrati tutti su un basso numero di partizioni. Oltre tutto, i 25 post tracciati sono quelli più hot, motivo per cui tenderanno a generare **maggior traffico** sulle poche partizioni utilizzate. Questo contribuirà quindi ad aumentare il tempo medio in coda.

Nei casi con il 50% e il 75% del carico, si traceranno un numero maggiore di topic ma di minore importanza. Questo porterà un numero maggiore di partizioni ad essere occupate, alcune delle quali però con un quantitativo di messaggi non molto significativo. Dal momento che ci saranno dei messaggi

che rimarranno in coda un tempo minore, questi contribuiranno ad abbassare il tempo medio di permanenza in coda.

Discorso simile avviene per l'esperimento con il 100% del carico. Dal momento che tutte le partizioni saranno occupate sin da subito, allora thread diversi inizieranno a condividere le stesse partizioni, portando quindi il tempo medio in coda ad aumentare (rispetto ai casi con il 50% e il 75% del carico). Venendo comunque tracciati un certo numero di thread poco rilevanti, e che quindi genereranno un quantitativo di messaggi molto inferiore rispetto ai primi, certe partizioni saranno meno occupate, contribuendo a mantenere il valore della media inferiore all'esperimento con il 25% del carico.

9.5.5 Queue Time e Processing Time su Threads - Batch Layer

Di seguito vengono presi in esame il **Queue Time** e il **Processing Time** per quanto riguarda il topic **Threads** sul **Batch Layer**.

Il **Processing Time** rappresenta il tempo trascorso da quando il consumer (in questo caso quello del batch layer) preleva il messaggio dalla coda di Kafka a quando lo inserisce effettivamente nel database (MongoDB). Per come sono stati presi i tempi nell'applicazione il tempo di processamento include anche quello impiegato per recuperare i vecchi valori nel caso si debba effettuare un'operazione di **update**. In Figura 33 si può vedere l'andamento dei tempi di processamento per i 4 esperimenti, durante le 15h di monitoraggio.

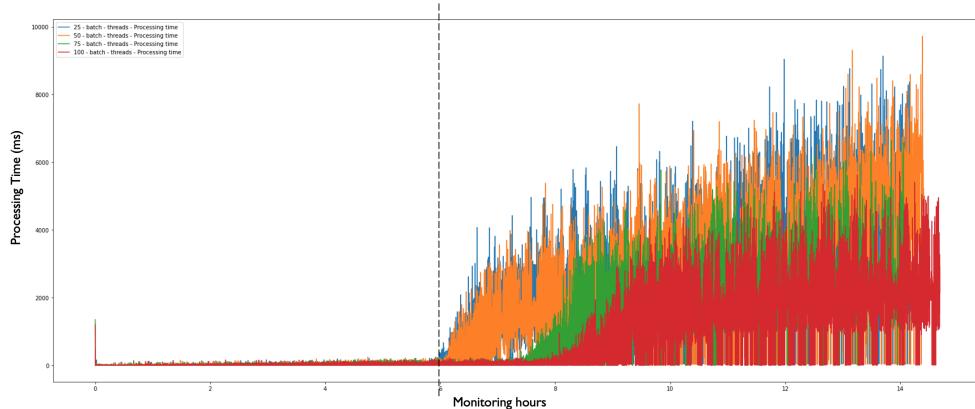


Figura 33: Batch Layer - Threads topic - Andamento del Processing Time durante le ore di monitoraggio.

Il **Queue Time** rappresenta sempre il tempo trascorso su Kafka da parte dei messaggi. In Figura 34 viene riportato l'andamento del tempo in coda dei messaggi prima di essere prelevati dal Batch consumer per i 4 esperimenti, durante le 15h di monitoraggio.

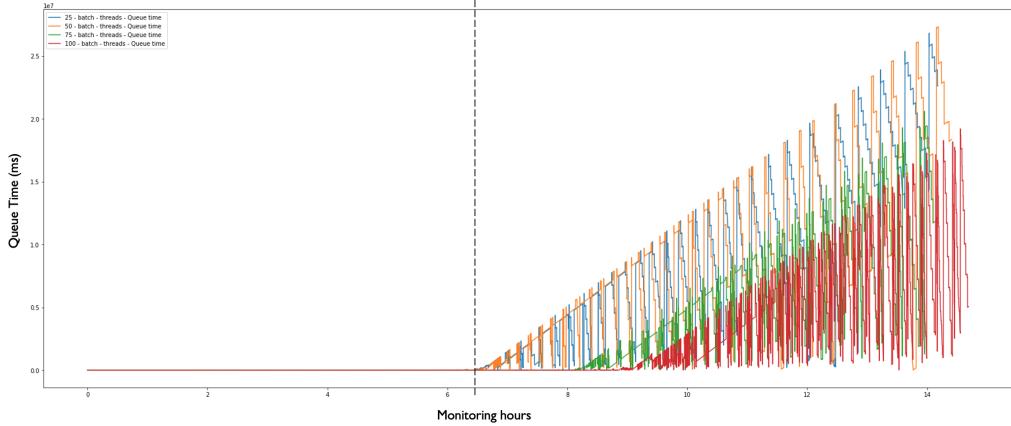


Figura 34: Batch Layer - Threads topic - Andamento del Queue Time durante le ore di monitoraggio.

Cosa molto interessante da notare è che attorno alle 6 ore trascorse dall'inizio del monitoraggio, i tempi riportati in entrambi i grafici tendono fortemente ad aumentare. Il fatto che entrambi aumentino circa nello stesso range temporale non può essere sicuramente una casualità, il che ha spinto l'analisi ad entrare maggiormente nei dettagli implementativi per trovare una risposta a questi comportamenti.

In entrambi i grafici è presente una linea verticale che coincide circa con l'inizio dell'aumento dei tempi. Prestando attenzione si può notare che il tempo di processamento (Figura 33) aumenta prima rispetto al tempo di coda (Figura 34).

Per spiegare il motivo per cui aumenta il **Processing Time** è necessario analizzare come sono stati organizzati i dati all'interno di MongoDB. Per memorizzare i post e i relativi commenti si è utilizzata la tecnica dell'**embedding** (vedi Figura 35), ovvero i commenti sono memorizzati come documenti innestati all'interno del post a cui fanno riferimento. Questo implica che ogni qual volta si presenti un messaggio di **comment-update**, allora è necessario prima ricercare il post, e poi trovare il commento nell'insieme di documenti innestati al post. La situazione si complica ulteriormente dal momento che i

thread più hot hanno associato un numero di commenti molto maggiore rispetto agli altri; oltretutto un maggior numero di messaggi di aggiornamento sarà riferito ad essi. Dal momento che il **Processing Time** include anche il tempo di ricerca del dato all'interno del database quando viene eseguita una update, questo tempo è fortemente impattato, per via della peculiarità appena spiegata.

Per dare un'idea dei numeri in gioco, durante il periodo dalle 6 alle 8h di monitoraggio, il numero medio di thread tracciati è circa 207 con un numero medio di commenti per thread di circa 60. Considerando che in questo intervallo, come riportato sulla sinistra di Figura 36, si ha un aumento di circa 15000 comment-update, per ognuna di esse si dovrà cercare su 207 thread e per ognuno di essi su circa 60 commenti (se la distribuzione dei commenti fosse uniforme sui thread). Non essendo tale si avranno per giunta i thread più hot con un numero di commenti estremamente più alto, e il numero di comment-update relativi ad essi sarà di conseguenza maggiore, spingendo i tempi di ricerca ad essere quasi sempre nel caso pessimo.

```

>   _id: ObjectId("601514252a90fd6a737150d0")
>   _flairs: "Megathread"
>   _upvotes: "14"
>   _author: "DTG_Bot"
>   _title: "[D2] Trials of Osiris Megathread [2021-01-29]"
id: "17xr9u"
type: "post-create"
_url: "https://www.reddit.com/r/DestinyTheGame/comments/17xr9u/d2_trials_of_o..."
  "#Trials of Osiris is LIVE
_text:
  This thread is for all general discussion, ..."
timestamp: "2021-01-30T09:09:07.970"
comments: Array
  > 0: Object
    _thread_id: "17xr9u"
    _author_id: "54kldr5c"
    _upvotes: "67"
    _author_name: "Friend0fCats"
    id: "g19bhta"
    type: "comment-create"
    _text: "Turned in bounty- 3 wins is helmet"
    timestamp: "2021-01-30T09:09:42.682"
    polarity: "pos"
  > 1: Object
  > 2: Object

```

Figura 35: Embedding dei commenti all'interno dei documenti dei post.

Una volta spiegato il motivo dell'aumento del tempo di processamento è stato possibile spiegare il comportamento del **Queue Time**. Difatti, come

si può notare, l'aumento dei tempi di coda è ritardato rispetto al processing time. Dal momento che i messaggi di **comment-update** impiegheranno un tempo maggiore per essere eseguiti, allora quelli in coda dovranno attendere maggiormente.

Dato che le collezioni tenderanno con il passare del tempo ad aumentare sempre più in dimensione, allora è giustificato il fatto che il **Processing Time** per via dell'esecuzione delle **comment-update**, e di conseguenza anche il **Queue Time**, tendano ad aumentare all'infinito.

Per verificare le ipotesi effettuate si è approfondita l'analisi andando a ricercare esattamente che cosa accadesse nell'intervallo tra le 6 e le 8 ore, in cui si ottiene un aumento drastico di quelli che sono i tempi di coda e di processamento.

Come si può vedere in Figura 36, nella parte di sinistra, nell'intervallo dalle 6 alle 8 ore il numero di **comment-update** aumenta da 38792 a 53338. Dall'immagine sulla destra è possibile dimostrare come un susseguirsi di messaggi di **comment-update** durante gli istanti successi alle 6h possano denotare un incremento del Queue Time e del Processing Time, espressi in millisecondi.

Incrociando quindi i dati ottenuti dai log dello scryper con quelli dei log del batch consumer è stato possibile dimostrare ed indagare più nel dettaglio quanto detto poc'anzi.

Scryper			Batch consumer				
	datetime	Comments Updated		elapsed time	messageType	Queue time	Processing time
361	2021-03-17 23:12:30	38792	75201	6.026848	comment-update	10054	63
362	2021-03-17 23:13:30	39125	75202	6.026867	comment-update	10077	63
363	2021-03-17 23:14:30	39239	75203	6.026887	comment-update	10109	62
364	2021-03-17 23:15:30	39328	75204	6.026906	comment-update	10154	59
365	2021-03-17 23:16:30	39465	75205	6.026924	comment-update	10204	58
476	2021-03-18 01:07:36	52858	75236	6.027535	comment-update	10887	59
477	2021-03-18 01:08:36	53011	75237	6.027553	comment-update	10910	59
478	2021-03-18 01:09:36	53109	75238	6.027572	comment-update	10960	84
479	2021-03-18 01:10:36	53204	75239	6.027597	comment-update	11027	58
480	2021-03-18 01:11:37	53338	75240	6.027614	comment-update	11063	68

Figura 36: Sulla sinistra è possibile vedere come aumenti il numero di comment-update nell'intervallo tra le 6 e le 8 ore; A destra si possono vedere come aumentano il Processing Time e il Queue Time (espressi in ms) a fronte delle comment-update.

9.6 Riepilogo dei bottleneck incontrati e possibili soluzioni

Attraverso il primo esperimento è stato possibile notare che la **RAM** costituisce un bottleneck significativo per l'applicazione: per poter essere eseguita senza problemi per un intervallo di tempo significativo essa necessita infatti di un quantitativo di RAM **superiore agli 8 GB**.

Una volta superato questo primo ostacolo è stato quindi possibile concentrarsi sui bottleneck relativi alle scelte implementative di Speed e Batch Consumer, e più nello specifico:

1. Nello **Speed Layer** si è notato che l' AVG Queue Time aveva comportamenti inconsistenti al variare del numero di thread monitorati. Questo ha consentito di riflettere sul modo in cui i topic Kafka erano organizzati.
2. Nel **Batch Layer** si è notato che dopo 6 ore il Processing Time e il Queue Time crescevano in maniera considerevole. Questo è dovuto all'utilizzo del **Document embedding** in MongoDB.

Per risolvere il problema 1, una delle possibili soluzioni potrebbe essere quella di **cambiare l'organizzazione dei topic** in Kafka, aggiungendo un nuovo topic dedicato esclusivamente ai Commenti, anziché inserirli nello stesso topic dei Threads. In questo nuovo topic, i messaggi utilizzerebbero come chiave l'id del commento, in modo da raggruppare tutti gli aggiornamenti relativi allo stesso commento nella stessa partizione; ovviamente l'id del thread dovrà essere aggiunto nel messaggio inviato. Questa soluzione potrebbe però portare a problemi nello Speed Layer, poiché i commenti potrebbero essere processati prima del thread a cui fanno riferimento, causando una **perdita di eventi**.

Per risolvere il problema 2, si potrebbe decidere di utilizzare la tecnica del **referencing** in MongoDB. Inizialmente si era pensato che l'embedding fosse la soluzione migliore ai fini di analisi con Tableau. Ad esempio per contare i commenti di un certo thread (per ogni thread, in modo da classificarli), sarebbe bastato accedere allo specifico thread e contare il numero di commenti ad esso associato. Utilizzando il referencing sarebbe invece stata necessaria un'operazione aggiuntiva di aggregazione, che su collezioni molto grandi potrebbe non essere indifferente. Inoltre in linea di principio l'embedding è la tecnica preferenziale per la gestione delle relazioni 1-n, come nel caso di un

thread e tutti i commenti ad esso associati. Attraverso i vari esperimenti è stato possibile rendersi conto del fatto che il trade-off tra tempi di inserimento e tempi di analisi batch sarebbe migliore nel referencing piuttosto che nell'embedding: visto che le analisi batch non hanno in genere stretti vincoli in termini temporali, non dovrebbe essere problematico se le interrogazioni richiedessero qualche minuto in più.

In conclusione scegliere la tecnica del **referencing** per la memorizzazione in MongoDB sul batch consumer sarebbe stata probabilmente la scelta migliore. In tal modo le **comment-update** non avrebbero impattato sulle prestazioni così come ora.

10 Conclusioni

In questo documento è stata presentata una pipeline per l'analisi dei dati di Reddit, basata su Apache Kafka e più nello specifico sul modello ad Event Sourcing. E' stato discusso nel dettaglio il funzionamento dello Scryper e dei due layer che compongono la Lambda Architecture: lo Speed Layer per la visualizzazione di una dashboard real-time, e il Batch Layer per effettuare analisi OLAP utilizzando l'applicativo Tableau.

Le principali difficoltà riscontrate riguardano la comprensione del funzionamento di Kafka e dei pattern per il suo corretto utilizzo, la progettazione ad alto livello dell'architettura (come ad esempio capire quali database e stack applicativi utilizzare) e ovviamente l'utilizzo dei diversi tool per la creazione di dashboard/OLAP. Anche la dockerizzazione dell'applicativo, affrontata soprattutto nelle fasi finali del progetto, non è stata semplice: è stato necessario effettuare alcune modifiche al codice pre-esistente, contribuendo però a migliorarne la qualità.

In seguito al feedback della prima discussione del progetto, si è deciso di analizzare la pipeline dal punto di vista della scalabilità. Per prima cosa è stato analizzato nel dettaglio il quantitativo dei dati gestiti in uno scenario verosimile. In seguito ne sono stati analizzati i bottleneck sia dal punto di vista dell'hardware necessario per eseguirla correttamente, che dal punto di vista delle decisioni implementative. Attraverso questa analisi è stato quindi possibile proporre possibili soluzioni e miglioramenti della pipeline.

I possibili sviluppi futuri riguardano la possibilità di utilizzare la pipeline per subreddit scelti direttamente dall'utente, oppure per monitorare più subreddit contemporaneamente. Si potrebbe inoltre pensare di addestrare un modello di Machine Learning per la rilevazione di commenti spam o offensivi, sfruttando per l'addestramento i dati memorizzati sul Batch Layer.

Il lavoro è stato diviso equamente tra i due componenti del gruppo, che sono rimasti costantemente in contatto per confrontarsi, risolvere problemi insieme, ecc.

Bibliografia

- [1] Reddit. <https://www.reddit.com>.
- [2] Reddit api. <https://www.reddit.com/dev/api/>.
- [3] Destinythegame. <https://www.reddit.com/r/DestinyTheGame/>.
- [4] Praw: The python reddit api wrapper.
<https://praw.readthedocs.io/en/latest/>.
- [5] kafka-python.
<https://kafka-python.readthedocs.io/en/master/index.html>.
- [6] Event sourcing using apache kafka. <https://www.confluent.io/blog/event-sourcing-using-apache-kafka/>.
- [7] Should you put several event types in the same kafka topic?
<https://www.confluent.io/blog/put-several-event-types-kafka-topic/>.
- [8] Eric Gilbert C.J. Hutto. Vader: A parsimonious rule-based model for sentiment analysis of social media text, 2014.
- [9] Corenlp. <https://stanfordnlp.github.io/CoreNLP/>.
- [10] Vader in java.
<https://github.com/apanimesh061/VaderSentimentJava>.
- [11] Elasticsearch: What is elasticsearch?
<https://www.elastic.co/what-is/elasticsearch>.
- [12] Elasticsearch: Glossary of terms. <https://www.elastic.co/guide/en/elasticsearch/reference/current/glossary.html#index>.
- [13] What is an elasticsearch index?
<https://www.elastic.co/blog/what-is-an-elasticsearch-index>.
- [14] Removal of mapping types. <https://www.elastic.co/guide/en/elasticsearch/reference/current/removal-of-types.html>.

- [15] Kibana: Your window into elastic stack.
https://www.elastic.co/kibana?ultron=B-Stack-Trials-EMEA-S-Exact&gambit=Elasticsearch-Kibana&blade=adwords-s&hulk=cpc&Device=c&thor=kibana&gclid=Cj0KCQiA6t6ABhDMARIIsAONIYyy803rF2q2f25Ioe1EP0U-zivxmkBWNja3lki6xa-UiInJof37ZQaAkggEALw_wcB.
- [16] Kibana-lens. <https://www.elastic.co/kibana/kibana-lens>.
- [17] Kibana: Index patterns. <https://www.elastic.co/guide/en/kibana/6.8/index-patterns.html>.
- [18] millisec-java.
<https://stackoverflow.com/questions/351565/system-currenttimemillis-vs-system-nanotime>.