



ArMeetup: Explore the Meetup Social Network in ArangoDB

Dario Bertazioli¹, Fabrizio D'Intinosante¹, Massimiliano Perletti^{1*}

Sommario

Meetup è una piattaforma che nasce con lo scopo di mettere in contatto le persone nella vita reale. Dopo la registrazione un utente può selezionare i propri interessi, iscriversi a gruppi locali, e partecipare ad eventi. In questo elaborato si è cercato di rispondere a diverse domande di ricerca tra le quali realizzare alcune misure quantitative per ispezionare i dati, verificare quale sia il momento migliore della settimana e della giornata in cui organizzare un meetup, cercare di individuare quanto ampio sia il raggio di un evento in termini di attrazione degli utenti, e cercare di valutare l'efficacia del sistema di raccomandazione della piattaforma, ricavando i dati appunto da Meetup tramite streaming RSVP. Per fare ciò è stata utilizzata la Virtual Machine Azure di Microsoft fornitaci dall'Università, nella quale si è usufruito di Nifi per effettuare lo streaming dei messaggi RSVP, di Kafka per lo storage temporaneo e di ArangoDB come database finale distribuito per poter effettuare le interrogazioni necessarie a rispondere alle nostre domande di ricerca e per permetterci di gestire in modo funzionale il grande volume di dati importato, con la possibilità di poter ulteriormente incrementarne il quantitativo grazie alla forte scalabilità orizzontale garantita dal database.

Keywords

AQL — ArangoDB — Cypher — DataManagement — DataVisualization — Kafka — Meetup — Neo4j — Nifi — Python — R — SocialNetwork

¹Data Science, Department of Computer Science, University of Milano Bicocca, Milan, Italy

*Corresponding author: m.perletti@campus.unimib.it

Indice

1	Goals	2
2	Implementation: Architecture	2
2.1	Covered points	2
2.2	Source	2
2.3	Nifi	3
2.4	Kafka	4
2.5	Neo4j	4
2.6	ArangoDB	4
3	Results	4
4	Interesting Challenges	5
	Acknowledgments	7

Introduction

Meetup è stato creato nel 2002 come piattaforma per mettere in contatto le persone nella vita reale. Una volta completata la fase di registrazione, gli utenti possono:

- Selezionare i propri interessi: ciò avviene sottoscrivendo dei topic precompilati dall'applicazione in modo da favorire il sistema di raccomandazione. Questi topic spaziano tra i più svariati ambiti, da quello professionale a quello degli hobby, fino ai più comuni, relativi ad eventi sociali.

- Iscriversi a gruppi locali: una volta selezionati i topic di interesse il sistema di raccomandazione dell'applicazione suggerisce all'utente una serie di gruppi più o meno locali (a seconda dell'area di interesse selezionata dall'utente) che trattano i topic in oggetto o altri topic ad essi correlati. Ciò permette agli utenti di incontrare persone della propria zona che condividono le stesse passioni. Nel caso in cui tra i topic di interesse per l'utente ce ne siano alcuni che non trovano nessun riscontro in gruppi presenti sul territorio, l'applicazione suggerisce all'utente di creare lui stesso un gruppo con oggetto quel topic, consigliando di mettersi in contatto con altre persone che dimostrano quell'interesse, suggerendole attraverso il sistema di raccomandazione.

- Partecipare ad eventi: i gruppi locali, nel corso del tempo, organizzano eventi, incontri, meeting con oggetto i topic dichiarati dai gruppi stessi. Una volta che un gruppo ha organizzato un evento l'utente può visualizzarli sulla

propria home e, una volta visionati i dettagli, decidere di comunicare se parteciperà oppure no e, nel caso volesse partecipare, se ha intenzione di portare degli ospiti, il tutto in maniera non vincolante. Inoltre, per non restringere troppo la sfera di interessi degli utenti, l'applicazione permette anche di selezionare diversi filtri per la home, tra cui uno apposito per visualizzare eventi organizzati da gruppi di cui non si fa parte o un filtro ad hoc per visualizzare gruppi di cui l'utente non è già membro e che magari non trattano topic per cui l'utente ha dichiarato interesse, ma che essendo pur sempre gruppi locali, potrebbe gradire o trovare interessanti.

Il focus centrale dell'intero meccanismo dell'applicazione risulta quindi essere quello del "gruppo" visto come realtà associativa e fautore di momenti di aggregazione durante l'intero anno, molto spesso non guidati da una ristretta cerchia di capi ma, anzi, promotore di iniziative da parte dei suoi stessi membri. La varietà di gruppi, a seconda dei topic trattati, spazia tra:

1. Gruppi di socializzazione, che hanno come obiettivo principale quello di svolgere attività ludiche in compagnia, molto spesso anche promotori di veri e propri eventi di incontro per single.
2. Gruppi professionali, che si pongono l'obiettivo di mettere in contatto persone e professionisti di svariati campi attraverso workshop o presentazioni con il fine di fare crescere gli utenti dal punto di vista professionale.
3. Gruppi creativi, ovvero gruppi di progettazione e più vocati ad hobby ed alla pratica delle più svariate arti.

Sintetizzando, quindi, la missione di Meetup è aiutare le persone a crescere e raggiungere i loro obiettivi attraverso connessioni autentiche e reali.

Ad oggi, Meetup è disponibile in 186 Paesi, e conta più di 40 milioni di membri sulla piattaforma, con più di 320 mila gruppi attivi ed una media di 12 mila eventi al giorno.

1. Goals

Con l'analisi della rete sociale di Meetup e andando a studiare le relazioni che interconnettono le persone attraverso eventi sociali organizzati dai diversi gruppi presenti in tutto il globo, è possibile avere una mappatura più o meno dettagliata degli interessi comuni che portano ad agglomerare un significativo numero di soggetti instaurando relazioni tra di essi.

Cercando di ricreare questa rete ci si è posti alcuni obiettivi, considerando anche i limiti dovuti all'acquisizione dati, ovvero quella avvenuta in forma streaming:

- Effettuare misure quantitative semplici per verificare, banalmente, quali siano i paesi in cui la piattaforma è più utilizzata ed affermata;
- Sfruttando le informazioni temporali, verificare quale sia il momento migliore della settimana e della giornata in cui organizzare un meetup per poter attrarre

più persone possibili, considerando che la piattaforma è per lo più utilizzata per incontri di carattere professionale;

- Sfruttando le informazioni spaziali, cercare di individuare quanto ampio sia il raggio di un evento in termini di attrazione degli utenti;
- Utilizzando le informazioni relative ai topic trattati dai gruppi che organizzano gli eventi e ai topic che i membri dichiarano in sede di iscrizione alla piattaforma, valutare l'efficacia del sistema di raccomandazione della piattaforma, sempre considerando i limiti naturali imposti dallo streaming.

L'obiettivo del progetto, in sintesi, è quello di identificare correlazioni o tracce significative che possano determinare delle buone regole nella creazione di eventi di notevole impatto sociale; una piccola guida "**How to**" su come dev'essere organizzato un evento di successo.

2. Implementation: Architecture

2.1 Covered points

L'intera strutturazione del progetto si pone come obiettivi didattici, oltre a quelli pratici già esposti, di coprire due delle tre **V** caratteristiche dei *Big Data*:

1. **Velocity**: attraverso l'applicazione di raccolta dei dati provenienti da una fonte streaming, con tutte le problematiche a questa connessa, come connessione e immagazzinamento dei dati real time.
2. **Volume**: la raccolta dei dati ha prodotto un quantitativo di notevole importanza, comportando la costruzione di un DB di dimensione superiore ai 2 GB. Ciò ha comportato la necessità di applicare tecniche e tool adeguati a compiere task su grandi moli di dati in tempi accettabili.

2.2 Source

Meetup, come già detto, mette a disposizione parte dei dati in suo possesso relativi a membri iscritti alla piattaforma, gruppi creati ed eventi organizzati nella stessa. In particolare, i dati sono forniti in due modalità:

streaming data: la piattaforma condivide dati in streaming, composti essenzialmente da **RSVP**. Le RSVP sono risposte di feedback da parte degli utenti comunicanti agli organizzatori di un evento la loro partecipazione (o la loro non partecipazione, in quanto le risposte possono essere anche negative), all'evento stesso. Un esempio di dato di questo genere è riportato sotto allo scopo di mostrarne la struttura.

- **event**:
 - event_id: Unique alphanumeric identifier
 - event_name: Name of the event
 - event_url: URL to the full event page
 - time: Event time if set in milliseconds since the epoch
- **group**:
 - group_city: Group's home city

- group_country: two-letter code of group's home country
 - group_id: Numeric identifier of the group
 - group_lat: Latitude of group's approximate location
 - group_lon: Longitude of group's approximate location
 - group_name: Name of the group
 - group_state: two-letter code of group's home state, if in US or CA
 - group_topics: Topics associated with this group
 - * topic_name: Longer name
 - * urlkey: Unique keyword
 - group_urlname: Unique portion of group's URL, no slashes
- guests: Number of guests the member is bringing
 - **member**: Member who RSVP'd
 - member_id: Unique numeric id
 - member_name: Full name given
 - other_services: e.g. "twitter": "identifier": "MeetupAPI"
 - photo: Thumbnail URL for member photo if one exists
 - **mtime**: Last modified time of this RSVP, in milliseconds since the epoch
 - response: "yes" or "no"
 - rsvp_id: Unique numeric identifier
 - **venue**: Venue, if public
 - lat: Latitude of the venue
 - lon: Longitude of the venue
 - venue_id: Unique numeric identifier
 - venue_name

Si è proceduto ad estrarre tali dati da websocket, sfruttando Nifi (vedasi prossimo paragrafo).

rest API: Meetup permette anche un accesso piuttosto flessibile al database, esponendo tramite rest API informazioni riguardo a membri, gruppi ed eventi in formato JSON¹. Abbiamo sfruttato questa possibilità per arricchire i dati acquisiti in streaming. In particolare, abbiamo proceduto all'enrichment delle informazioni sui membri, integrando per ciascun utente:

- la localizzazione spaziale, ovvero le coordinate geografiche (lat/lon) dichiarate in fase di registrazione così che il sistema di raccomandazione dell'applicazione possa suggerire gruppi su base geografica.
- i topic in cui l'utente ha dichiarato di avere interesse al momento dell'iscrizione alla piattaforma, anche in questo caso per permettere al sistema di raccomandazione di suggerire gruppi per cui l'utente potrebbe avere interesse.

L'integrazione è avvenuta tramite http(s) request (a <https://api.meetup.com/2/members>) via python script, chiedendo alla rest api di restituire informazioni in base all'ID dello specifico membro, iterando quindi su tutti gli utenti. Per ovviare al numero di richieste al minuto piuttosto

¹Si veda https://secure.meetup.com/meetup_api per una lista dei metodi disponibili.

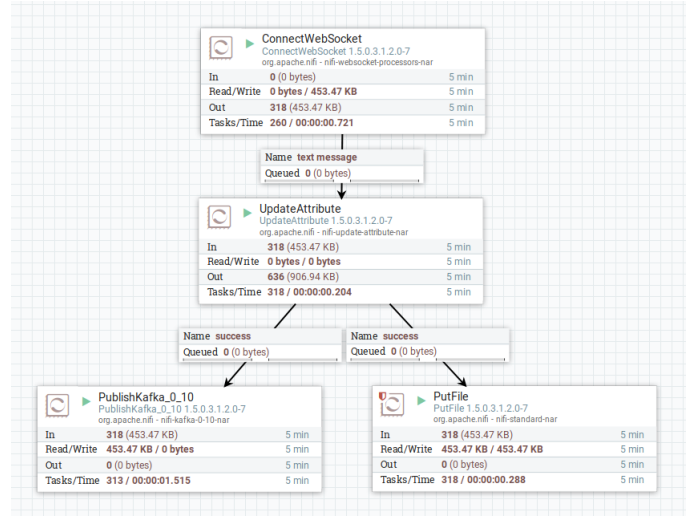


Figura 1. Il workflow Nifi

limitato abbiamo implementato un sistema a più chiavi, facendo un tuning della frequenza di richiesta in modo tale che risultasse appena al di sotto della soglia massima tollerata. Dunque, dopo aver partizionato in modo opportuno la lista di utenti in nostro possesso, abbiamo proceduto all'enrichment con richieste parallele. Inoltre, abbiamo preventivamente stimato il tempo totale necessario all'enrichment per tutti i (più di 800000) membri con tre richieste parallele, controllando che fosse tollerabile.

2.3 Nifi

Per acquisire i dati in streaming dalla piattaforma abbiamo implementato un workflow di Nifi (vedasi **fig. 1**) contenente:

- un nodo per la connessione alla **WebSocket** (`ws://stream.meetup.com/2/rsvps`), tramite un Client Jetty.
- un nodo per il parsing iniziale.
- un nodo **PutFile**: abbiamo deciso di salvare i dati acquisiti su file system per due motivi principali: innanzitutto ciò è utile per avere una copia di backup dei dati, qual ora si verificassero problemi (ad esempio, con il corretto setting di *retention time* di Kafka); in secondo luogo, terminato il periodo di acquisizione, per facilitare la copia dei dati su altre virtual machines, in modo da implementare contemporaneamente la stessa pipeline su più macchine, per poter confrontare (quantitativamente, in termini, ad esempio, di quantità di dati importati, tempi di esecuzione dei vari scripts etc.) i risultati ottenuti.
- un nodo **PublishKafka**, comodo per effettuare l'ingestione dei dati in Kafka, configurandosi questo come un vero e proprio producer (vedasi sezione successiva).

I principali vantaggi che Nifi ci ha fornito sono la gestione facilitata della queue, la possibilità di parsing e la facile interfaccia con il Kafka Producer.

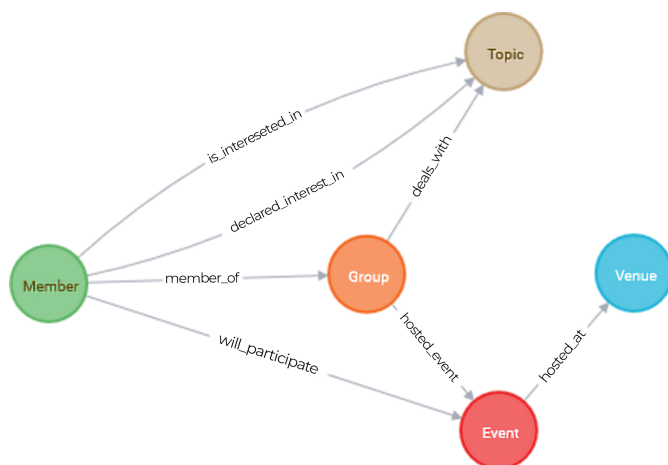


Figura 2. Schema graph DB

2.4 Kafka

Considerata la natura dei dati a nostra disposizione, che ricordiamo consistere in un flusso di messaggi in streaming (RSVP), abbiamo deciso di fare un ingestion in Kafka, potendo successivamente esplorare, iterare, e preprocessare le informazioni tramite Python API. Abbiamo inoltre modificato i parametri di RetentionTime, in modo tale che Kafka fungesse da primo storage temporaneo, in grado di salvare all'interno di un topic tutti i messaggi acquisiti in real time streaming, per poi poterli processare successivamente senza correre il rischio di perdere qualche informazione. Come accennato, con l'API di Kafka per Python abbiamo potuto configurare un consumer per processare i dati, creando diversi `csv`, in modo da facilitare e rendere più efficiente l'import nel database finale.

2.5 Neo4j

Abbiamo scelto inizialmente di salvare i nostri dati in un database a grafo per via della loro struttura naturale, che ben si presta ad essere schematizzata con nodi principali, proprietà e relazioni che interconnettono tali nodi. Abbiamo provveduto quindi all'import dei dati in Neo4j, sfruttando il linguaggio Cypher e l'ottimizzazione dell'import via `csv`, dopo aver in un primo momento tentato un approccio tramite l'api Python Py2Neo. Per facilitare un'eventuale serializzazione del processo di import, abbiamo creato uno script (Bash) che, una volta creati i `csv`, consente di automatizzare le varie chiamate alla cypher-shell, dato che il processo di creazione del database risulta piuttosto laborioso ma facilmente standardizzabile (`import.sh`). Questa soluzione però mal si prestava alla prospettiva di accumulare più dati, anche in vista di un possibile ampliamento degli orizzonti del progetto, dato che Neo4j non permette la distribuzione orizzontale dei propri database ma giova soltanto di una scalabilità di tipo verticale, per questo abbiamo quindi deciso di avvalerci di un altro servizio.

2.6 ArangoDB

Data a questo punto l'esigenza di trovare una soluzione che coniugasse il nostro bisogno di un database a grafo, ma che fosse anche in grado di essere scalabile orizzon-

talmente e quindi distribuibile con processi interni e non tramite segmentazione dei dati all'origine, anche nell'ottica di voler eventualmente implementare uno streaming continuo dall'RSVP API, abbiamo optato per ArangoDB, un DBMS multi-modello in grado di unire i modelli documentale, a grafo e key-value, interrogabili con un unico linguaggio di query condiviso (AQL, un misto tra un linguaggio `SQL-like` e un linguaggio di programmazione). Sfruttando il particolare concetto di grafo implementato in ArangoDB abbiamo così realizzato una *collection* per ogni vertice ed una *edge-collection* per ogni arco del grafo. Nel caso di una ingestione di dati superiore alla nostra la strategia da adottare, data la natura della piattaforma, potrebbe basarsi sulla specificazione di uno shard per ogni continente e specificare in fase di ingestione, per ogni collection e per ogni istanza il continente di riferimento², tutto ciò da specificarsi in fase di creazione del DB. Sulla base inoltre della disponibilità è possibile anche creare più cluster, con conseguente possibilità di implementare un'ulteriore strategia di frammentazione basata su criteri ulteriori. Il database da noi realizzato presenta circa 1 milione e mezzo di singoli nodi e circa 28 milioni di relazioni tra questi, con un tempo di import su ArangoDB stimato in circa venti minuti.

3. Results

Misure quantitative semplici : Con lo scopo di ispezionare alcune misure quantitative abbiamo sottoposto il DB da noi creato ad alcune query per estrarre informazioni quali:

- Quantità di eventi per Paese;
- Quantità di gruppi per Paese;
- Numero massimo di partecipanti agli eventi per Paese;
- Numero medio degli ospiti portati da ogni partecipante agli eventi;
- Trend topic per gli utenti;
- Trend topic per i gruppi.

Come mostrato in **fig. 3** i Paesi che presentano il più alto numero di gruppi attivi sono quelli anglofoni come USA e Gb, e sempre attraverso la visualizzazione sembrerebbe proprio che il numero totale di eventi organizzati in un Paese dipenda direttamente dal numero di gruppi attivi per quel Paese. Anche per quanto riguarda il numero massimo di partecipanti agli eventi per Paese gli USA ovviamente spiccano per conformazione demografica rispetto ai paesi europei. In merito invece ai trend topic, per i gruppi i più quotati sembrerebbero quelli inerenti benessere, argomenti sociali, divertimento ed impresa mentre, per gli utenti, i più presi in considerazione sembrerebbero quelli di tipo tecnologico ed inerenti la computer science.

Utilizzo delle informazioni temporali : Attingendo alle informazioni temporali degli eventi, presenti nei messaggi, ed applicando un'opportuna manipolazione per convertire

²`db._create("sharded_collection",
"numberOfShards": 5, "shardKeys": ["continent"])`

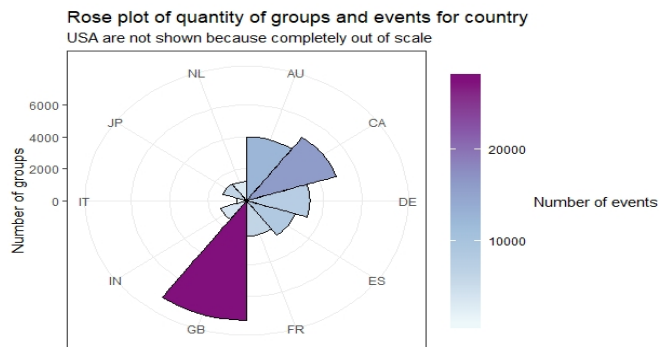


Figura 3. Rose plot dei dieci paesi principali per numero di gruppi ed eventi (USA non considerati per evitare problemi di scala)

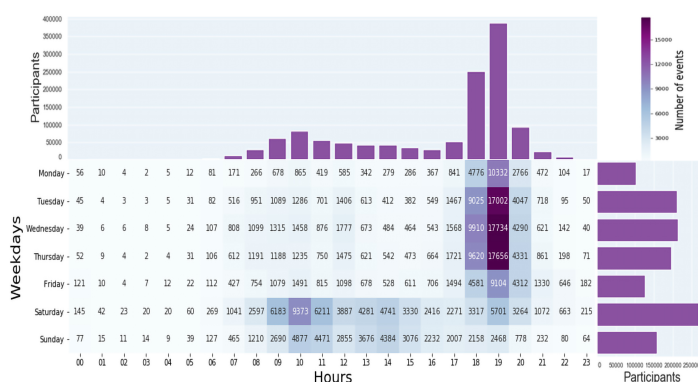


Figura 4. Heatmap distribuzione temporale degli eventi

l'originale formato timestamp in un formato umanamente più leggibile, siamo stati in grado di rispondere alla nostra seconda domanda di ricerca. Dopo aver infatti adattato i metadati temporali alle time zone così da avere una panoramica coerente sugli eventi Meetup tenutisi nel mondo, indipendentemente dal fuso orario, abbiamo provveduto a realizzare una tabella di Pivot, tenente conto della numerosità di partecipanti agli eventi discriminati per giorno della settimana e per fascia oraria della giornata. Una volta realizzata la tabella è stato quindi possibile procedere alla costruzione della *heatmap* in **fig. 4** per rappresentare al meglio queste informazioni. Appare infatti chiaro dalla visualizzazione come i momenti migliori della settimana per organizzare un evento risultino essere il Martedì, il Mercoledì ed il Giovedì nella fascia oraria 18.00/20.00 ed il Sabato mattina tra le 10.00 e le 12.00. Ciò si conforma bene alla nostra intuizione che la maggior parte dell'utenza di Meetup sembrerebbe essere rappresentata da lavoratori, professionisti, che sono quindi più propensi a partecipare ad eventi in orario successivo alla fine della giornata lavorativa o durante giorni non lavorativi, ma in questo caso prevalentemente la mattina.

Utilizzo delle informazioni spaziali : Per esplorare a fondo le informazioni in nostro possesso, abbiamo sfruttato i metadati spaziali presenti nei messaggi per realizzare

una visualizzazione interattiva in formato di mappa esplorabile, focalizzata sull'Europa. Con questa mappa l'utente può potenzialmente esplorare l'offerta di eventi su base spaziale e visualizzarli localizzati nel luogo in cui sono stati organizzati e, cliccandoci sopra, può acquisire informazioni quali le caratteristiche dell'evento, come il nome dell'evento, il nome del luogo in cui è stato organizzato, l'orario, le keywords relative all'argomento trattato nell'evento e altro ancora. La mappa è stata costruita stratificando più layer, in modo da permettere all'utilizzatore di filtrare a priori ciò che potrà visualizzare, attraverso l'utilizzo di widget. Questi filtri permettono di focalizzare la visualizzazione su un certo Paese europeo, oppure di filtrare gli eventi sulla base dei topic trattati dagli stessi. Per limiti computazionali, inoltre, è stata effettuata una selezione a priori degli eventi, per cui sulla mappa saranno visibili solamente gli eventi che hanno raggiunto il più alto numero di adesioni, ma la struttura così come realizzata e in presenza di mezzi idonei potrebbe facilmente permettere di visualizzare la totalità degli eventi presenti a livello europeo o mondiale.

Valutazione efficacia sistema di raccomandazione : Con lo scopo di valutare il sistema di raccomandazione della piattaforma e sfruttando le informazioni a nostra disposizione abbiamo calcolato la Jaccard similarity per ogni utente, prendendo in considerazione i topic comuni tra quelli dichiarati dall'utente in fase di registrazione ed i topic dei gruppi di cui l'utente fa parte, come si vede in **fig. 5**. Questa misurazione ci ha permesso di giungere ad una certa constatazione riguardo la modalità di acquisizione dei dati. La nostra misurazione risulterebbe infatti viziata dalla raccolta dati tramite streaming poichè noi siamo a conoscenza, per ogni utente, soltanto dei gruppi di cui fa parte condizionatamente al fatto che durante il periodo di streaming egli abbia risposto ad un evento organizzato da questi gruppi, e non della totalità dei gruppi a cui sarebbe iscritto. Questo particolare molto probabilmente contribuisce in maniera significativa alla determinazione di una bassa similarità tra i topic conformi ai gusti dichiarati dagli utenti e la totalità dei topic dichiarati sia dagli utenti che dai gruppi. Va inoltre tenuto conto che sia gli utenti in fase di registrazione che i gruppi al momento della creazione, molto probabilmente, prestano poca attenzione alla quantità e alla sparsità dei topic inseriti nel sistema, data anche la loro elevata granulosità, condizionando così la nostra misurazione.

4. Interesting Challenges

Durante la realizzazione del progetto abbiamo dovuto affrontare diverse problematiche legate ai più svariati ambiti, da quello della scalabilità in import dei dati a quello della velocità di processing, fino all'implementazione di una vera e propria costruzione continua del database con processo live streaming.

Symbolic Links : Avendo effettuato lo streaming utilizzando una sola Virtual Machine (VM), per poter trasferire i dati anche sulle altre macchine abbiamo provveduto a creare parallelamente all'ingestion diretta in Kafka anche un backup direttamente nella macchina. Ciò ci ha

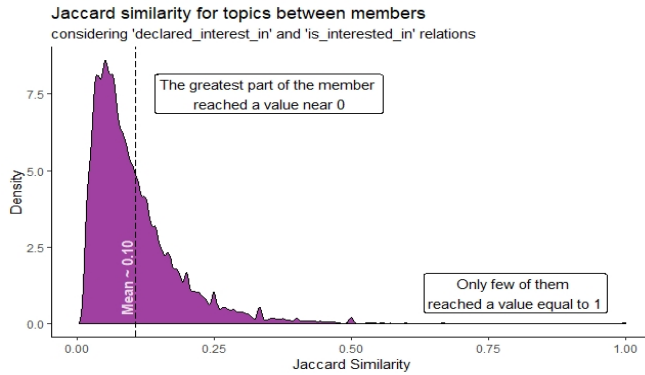


Figure 5. Jaccard similarity degli utenti riguardo i topic corrispondenti tra quelli dichiarati in fase di registrazione e quelli dei gruppi di cui fanno parte

permesso di trasferire direttamente questo backup dalla macchina principale alle altre. Successivamente nelle nuove VM si rendeva quindi necessario implementare un ingestion dalla VM a Kafka per mezzo di un workflow Nifi, così da emulare la stessa procedura dello streaming. Per ovviare poi ad un bug/limite dei nodi *getFile/FetchFile* di Nifi, che si verifica nel tentativo di eseguire un ingestion di elevata quantità di file in una sola volta, abbiamo escogitato una soluzione creando dei link simbolici, dividendo la creazione in sottocartelle, facendo così leggere al nodo *GetFile* questi link e impostando il nodo in modo che venisse eliminato il link dopo la singola lettura/ingestion. Ciò ci ha permesso di conservare i dati originali ed impedire che i nodi di import li eliminassero, facendoci risparmiare tempo per quanto riguarda la creazione dei database di backup sulle diverse VM.

Ottimizzazione tempi di estrazione dei messaggi :

Con il fine di estrarre le singole entità presenti nei messaggi, che per la natura dello streaming risultavano ripetersi più volte a seconda delle interazioni degli utenti con il servizio, abbiamo tentato inizialmente un approccio mediante liste su python. Sinteticamente questo approccio consisteva nel creare un Kafka consumer tramite l'API kafka per python e, ciclando, produrre tutti i messaggi in nostro possesso estraendo le singole entità, come utenti, gruppi e altro, utilizzando due liste di cui una contenitore per le singole entità ed una come "black list" per tenere traccia delle entità già inserite e quindi da non riprocessare. Questo approccio si è però dimostrato limitato in termini di performance nel processare grandi volumi di dati e ci ha portati ad optare per una soluzione alternativa: utilizzando dei dizionari, e sfruttando gli ID presenti per ogni entità contenuta nei messaggi, abbiamo utilizzato questi ID come chiavi dei dizionari assegnandogli come valore tutte le proprietà associate a quell'entità. La differenza in termini di prestazioni si è dimostrata estremamente netta. Come si vede in **fig. 6** la differenza è estremamente marcata, a partire dall'estrazione di più di 100 mila singole istanze il tempo richiesto dalle liste cresce esponenzialmente mentre quello dei dizionari resta sostanzialmente costante, per questo motivo per poter estrarre i CSV necessari all'import, su Neo4j prima e su ArangoDB dopo, abbiamo deciso di

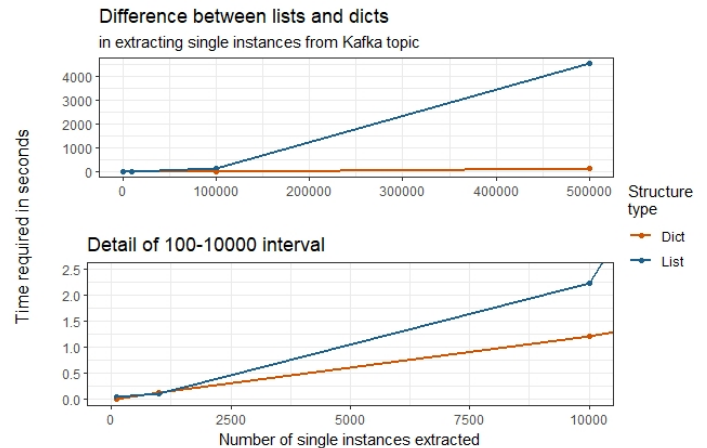


Figure 6. Differenza di prestazioni fra liste e dizionari

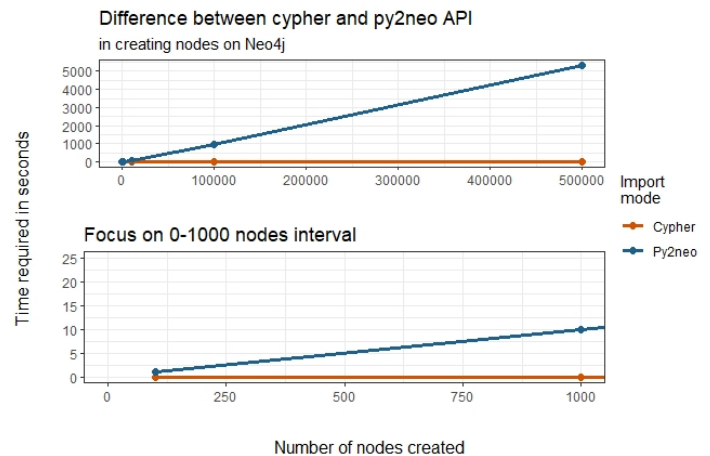


Figure 7. Differenza di prestazioni fra Cypher e Py2Neo API

adottare i dizionari.

Import Cypher vs Py2Neo : Una volta ottenuti i CSV necessari all'import dei nodi e delle relazioni abbiamo innanzitutto tentato un approccio con l'API Py2Neo per python. Come per le liste in precedenza, l'API si è comportata bene sulle ridotte dimensioni, mentre al crescere del volume dei dati da importare si è resa inutilizzabile per gli elevati tempi d'attesa. Come si vede in **fig. 7** l'import tramite cypher shell, meccanismo nativo di import per Neo4j, si dimostra fin dalle piccole dimensionalità molto più efficiente di Py2Neo mantenendo praticamente costante il tempo di import, con prestazioni che toccano circa i 12 secondi per importare 500 mila nodi distinti, contro i circa 80 minuti richiesti da Py2Neo.

Import Cypher vs Arangoimp Come si può vedere in **fig. 8** il confronto tra Cypher e Arangoimp nell'import dati si gioca sull'ordine dei millisecondi per piccole importazioni mentre di pochi secondi per grandi moli di dati. Arangoimp si dimostra sostanzialmente costante per i volumi testati, mentre l'import tramite Cypher comincia a crescere in termini di tempo richiesto partendo da volumi

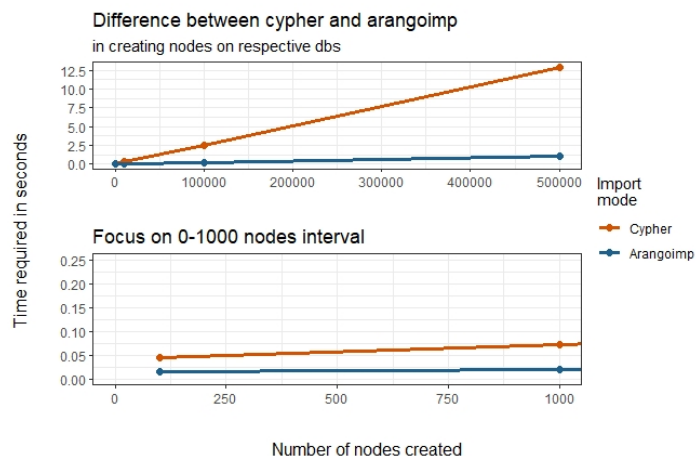


Figura 8. Differenza di prestazioni fra Cypher e Arangoimp

superiori a 100mila istanze. Anche per quanto riguarda l'import delle relazioni tra i nodi Arangoimp si è dimostrato molto più rapido. Questo probabilmente è ascrivibile al fatto che Neo4j sfrutta un meccanismo chiamato Index free adjacency che permette di incrementare le prestazioni a livello di query a discapito di una maggiore lentezza in fase di import.

Scalability : In prima istanza, dopo un primo streaming effettuato alla fine di dicembre 2018, abbiamo sfruttato questi dati per calibrare le nostre scelte in fatto di gestione dimensionalità. Questo primo streaming contava circa 1 milione di messaggi provenienti dall'API Meetup. Proprio grazie a questo quantitativo iniziale abbiamo potuto tarare le nostre decisioni su una dimensionalità realistica e già piuttosto complessa. Per questa ragione abbiamo deciso di sfruttare in prima istanza Kafka, che ha permesso di effettuare il primo storage senza essere condizionato in fase di consuming dalla dimensionalità dello storage stesso. Anche la scelta di un database a grafo, e nello specifico di Neo4j in principio, è stata presa con l'intento di ottenere il massimo delle prestazioni possibili in fatto di scalabilità, anche se poi sostituito successivamente dall'adozione di ArangoDB, ancora più performante in fase di import, e con la possibilità di essere distribuito orizzontalmente. Infine le scelte operate in fatto di processing dei dati con i dizionari per l'estrazione delle singole entità prima, e con l'applicativo Arangoimp per l'import su ArangoDB dopo, si sono dimostrate essere vincenti per quello che è stato il vero e proprio streaming effettuato a febbraio 2019. Quest'ultimo infatti contava più di 2 milioni e mezzo di messaggi, con una dimensionalità finale del database di 1 milione e mezzo di singoli nodi e circa 28 milioni di relazioni tra questi.

Conversione dei metadati temporali : Durante la fase di analisi dei dati con l'obiettivo di realizzare statistiche e visualizzazioni, ci siamo imbattuti nel problema della formattazione dei metadati temporali. Nello specifico tutti i metadati temporali provenienti dall'API di Meetup si presentano in formato timestamp in millisecondi con fuso orario UTC; questo rappresentava per noi un problema nel-

	name	time	country	state	lon	lat		
0	☐Lunch meetup at Shanghai Slims 13th April Sat...	1.555128e+12	cn	none	121.47	31.23		
1	It Meetup Salud Miu003vil, IA Salud - Observa...	1.554413e+12	co	none	-75.59	6.25		
	name	time	country	state	lat	lon	timezone_str	
555	Data Showdown 2019	1.559115e+12	au	au	-33.87	151.21	Australia/Sydney	
556	Database DevOps - Keep Your Delivery Processe...	1.557356e+12	us	nc	35.69	-78.64	America/New_York	
	name	country	state	lat	lon	timezone_str	time	datetime_timezone
0	☐Lunch meetup at Shanghai Slims 13th April Sat...	cn	cn	31.23	121.47	Asia/Shanghai	1.555128e+12	13/04/2019 12:00:00
1	It Meetup Salud Miu003vil, IA Salud - Observa...	co	co	6.25	-75.59	America/Bogota	1.554413e+12	04/04/2019 17:00:00

Figura 9. Passaggi conversione timestamp into datetime timezoned

la rappresentazione temporale degli eventi ad esempio, oppure per quella della sottoscrizione agli eventi da parte degli utenti. Per ovviare a questo inconveniente, e formattare questi metadati in formato date-time condizionati alle rispettive time-zone locali abbiamo sfruttato i metadati riguardanti la localizzazione degli utenti al momento della sottoscrizione agli eventi e, quindi, alla generazione del messaggio RSVP. Partendo quindi da queste informazioni in formato latitudine/longitudine abbiamo innanzitutto localizzato le corrispondenti time-zone attraverso la libreria *timezonefinder* per python e, successivamente, attraverso la libreria *pytz* abbiamo potuto convertire i timestamp in date-time condizionatamente alla time-zone prima definita. Lo sviluppo dei dati è visibile in **fig. 9**, dove è possibile visualizzare come il dataset si sia trasformato durante i passaggi descritti.

Implementazione ingestion in database live streaming

Con l'obiettivo di implementare il meccanismo di ingestion continuo nel DB, sulla base di uno streaming no-stop, si è implementata una pipeline che, a partire dal consumer Kafka, acquisisce i messaggi, li elabora con lo stesso meccanismo utilizzato per creare i csv e, sfruttando *python-arango*, un api di interfaccia tra python e ArangoDB, effettua l'ingestion nel database, preservandone la consistenza effettuando un check per evitare di inserire duplicati. Per questa fase di ingestion previste due possibili strategie:

- Un ingestion continua, per cui ogni messaggio viene inserito nel DB in seguito al check della consistenza;
- Un ingestion per fasi, la quale consiste nel creare dei batch db tramite l'api, per cui si raccolgono un determinato numero di messaggi e si effettua un caricamento più consistente.

Per testare la tenuta della pipeline abbiamo effettuato l'ingestion di tutti i circa 2 milioni di messaggi precedentemente archiviati in Kafka, verificando quindi la resistenza ad un grande carico: tipicamente infatti la frequenza dei messaggi oscilla tra i 100 ed i 300 messaggi al secondo, ben al di sotto delle frequenze testate, ovvero tra 1000 e 10000 messaggi al secondo.

Acknowledgments

So long and thanks for all the fish.