Pasquale Manfredi

Alberto Alpe

Università degli studi di Torino

Corso di Laurea Magistrale di Informatica, a.a 2023/2024

Modelli e Architetture Avanzati di Basi di Dati

Progetto: E-commerce di Libri con Database NOSQL

1. Introduzione e descrizione del problema

Presentazione del Progetto

Nel contesto dell'evoluzione dell'e-commerce e della gestione dei dati, l'adozione di database NoSQL offre molteplici vantaggi, soprattutto per applicazioni con necessità di scalabilità, flessibilità e alta disponibilità. Abbiamo deciso di proporre un progetto di e-commerce specializzato nella vendita di libri, utilizzando una base di dati poliglotta composta da Riak e MongoDB.

Obiettivi Principali

L'obiettivo del progetto è sviluppare un e-commerce che offra una vasta selezione di libri provenienti da diverse regioni del mondo. Utilizzando una combinazione di database NoSQL, il sistema sarà in grado di gestire in modo efficace sia l'inventario dei libri che lo storico degli acquisti degli utenti.
I principali obiettivi includono anche l'adozione di un sistema che consigli i libri a gli utenti in base all'interesse espresso negli acquisti sul portale web.

Analisi del Problema

Identificazione delle esigenze di scalabilità e flessibilità nella gestione dei dati di un e-commerce, compresa la gestione di un ampio inventario e un elevato numero di transazioni.

Requisiti Funzionali e Non Funzionali

Requisiti Funzionali

- Gestione dell'inventario dei libri
- Gestione dello storico degli acquisti
- Sistema di registrazione e autenticazione utenti
- Sistema di raccomandazione di libri basato sugli acquisti precedenti

Requisiti Non Funzionali

- Scalabilità
- Alta disponibilità
- Flessibilità nella gestione dei dati

2. Modello di Business

Selezione Globale

Offrire una vasta gamma di libri provenienti da diverse regioni del mondo, per soddisfare i gusti e le preferenze dei clienti di più Paesi.

Personalizzazione

Utilizzare l'analisi degli acquisti passati per suggerire libri pertinenti e personalizzati agli utenti, migliorando l'esperienza di shopping.

Spedizioni Internazionali

Collaborare con fornitori e corrieri per garantire spedizioni efficienti e tempestive in tutto il mondo.

Programmi di Fidelizzazione

Implementare programmi di fidelizzazione per incoraggiare gli acquisti ripetuti e premiare i clienti fedeli.

Profilo dell'Acquirente

Il nostro acquirente ideale è un appassionato di libri, desideroso di esplorare una vasta gamma di generi e autori provenienti da tutto il mondo. Si tratta di un individuo che apprezza la convenienza dello shopping online e cerca una piattaforma che offra una vasta selezione di libri, insieme a suggerimenti personalizzati basati sui suoi interessi e acquisti passati.

Flusso del Processo di Acquisto

Navigazione e Ricerca

Gli acquirenti navigano attraverso il catalogo online o utilizzano la barra di ricerca per trovare libri di loro interesse.

Selezione del Prodotto

Dopo aver trovato un libro desiderato, gli acquirenti possono visualizzare dettagli aggiuntivi come il prezzo, la disponibilità e le recensioni.

Aggiunta al Carrello

Gli acquirenti aggiungono il libro al loro carrello virtuale e possono continuare a navigare o procedere al pagamento.

Pagamento

Gli acquirenti procedono al pagamento utilizzando metodi di pagamento sicuri e affidabili.

Conferma e Spedizione

Dopo il completamento del pagamento, ricevono una conferma dell'ordine e il libro viene preparato per la spedizione.

Consegna

Una volta spedito, il libro viene consegnato all'indirizzo specificato dall'acquirente, con tempi di consegna stimati.

3. Scelta della Tecnologia di Database

Strumenti scelti per la realizzazione del progetto

- **MongoDB**: per la gestione dell'inventario dei libri e dei profili utenti
- **Riak**: per la gestione del sistema di raccomandazione di libri basata sugli acquisti degli utenti

Motivazione delle Scelte Tecnologiche

- **MongoDB**: scelto per la sua flessibilità nella gestione di dati strutturati e non strutturati e le sue potenti capacità di query.
- **Riak**: scelto per la sua capacità di gestione di dati distribuiti e la sua alta disponibilità, ideale per il sistema di raccomandazione.

Dettagli di Utilizzo

Riak

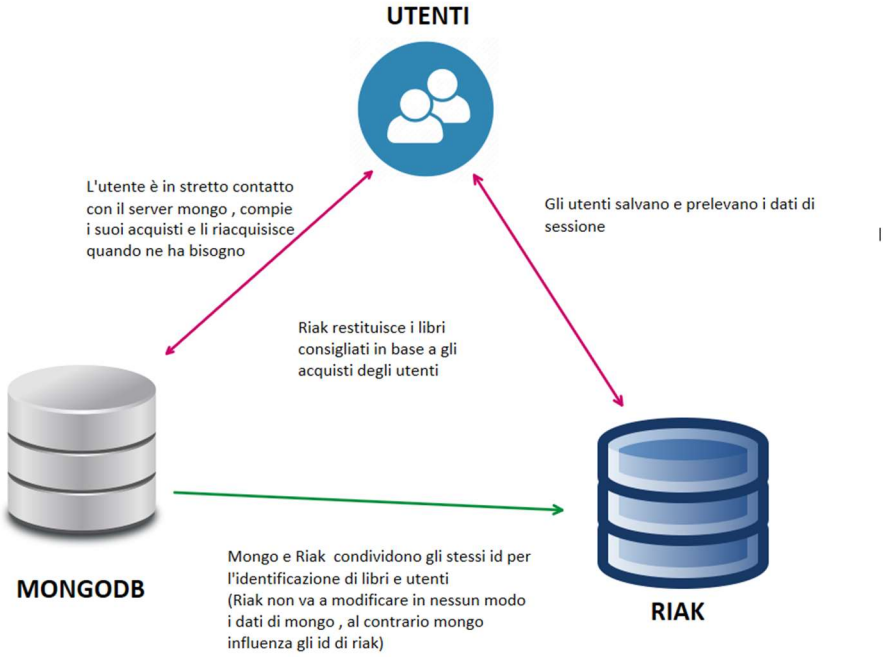
- Organizzato utilizzando i bucket per le regioni, con sotto-bucket per le librerie.
- Implementa un Inverted Index che mappa le parole chiave ai libri, consentendo una ricerca efficiente basata sulle parole chiave.

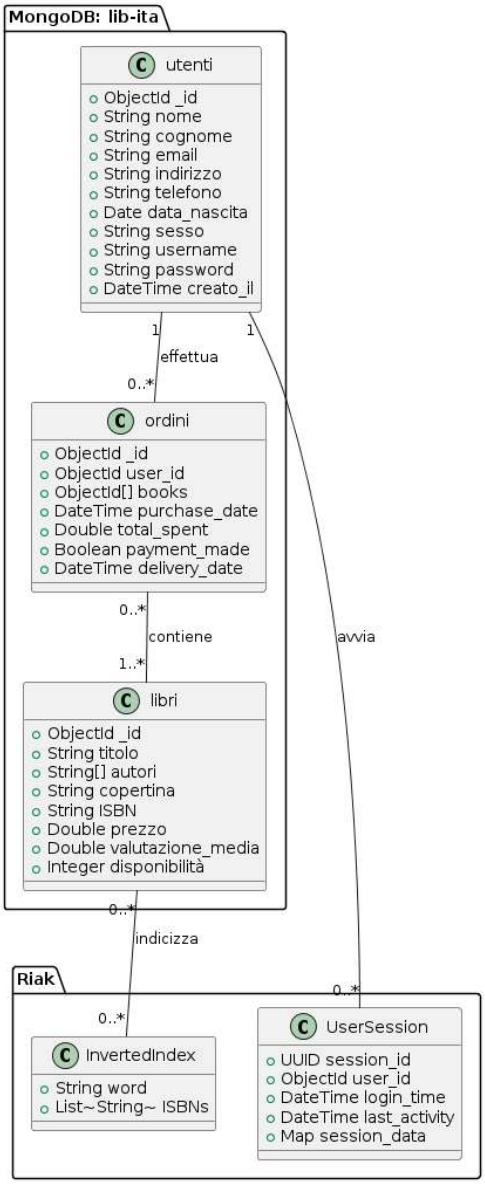
MongoDB

- Gestisce, le collezioni di libri e utenti e lo storico degli acquisti degli utenti.
- Utilizzato per proporre libri in base agli acquisti precedenti degli utenti.

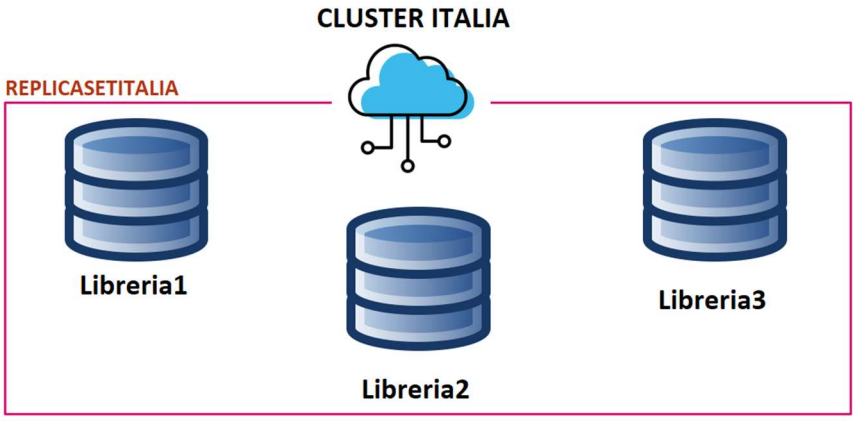
4. Progettazione del Database

Nel nostro progetto, i nodi dei database non sono considerati come magazzini fisici, ma come server distribuiti. Il magazzino dei libri è unico e identificato dal cluster. Nel caso si rendesse necessario gestire più magazzini per paese, MongoDB avrà un campo specifico per identificare la locazione del libro.


UTENTI
L'utente è in stretto contatto con il server mongo , compie i suoi acquisti e li riacquisisce quando ne ha bisogno
Gli utenti salvano e prelevano i dati di sessione
Riak restituisce i libri consigliati in base a gli acquisti degli utenti
MONGODB
RIAK
Mongo e Riak condividono gli stessi id per l'identificazione di libri e utenti
(Riak non va a modificare in nessun modo i dati di mongo , al contrario mongo influenza gli id di riak)


MongoDB: lib-ita
utenti
ObjectId _id
String nome
String cognome
String email
String indirizzo
String telefono
Date data_nascita
String sesso
String username
String password
DateTime creato_il
1
1
effettua
0..*
ordini
ObjectId _id
ObjectId user_id
ObjectId[] books
DateTime purchase_date
Double total_spent
Boolean payment_made
DateTime delivery_date
0..*
contiene
1..*
avvia
libri
ObjectId _id
String titolo
String[] autori
String copertina
String ISBN
Double prezzo
Double valutazione_media
Integer disponibilità
0..*
indicizza
Riak
0..*
0..*
InvertedIndex
String word
List~String~ ISBNs
UserSession
UUID session_id
ObjectId user_id
DateTime login_time
DateTime last_activity
Map session_data

Struttura dei database utilizzati e come comunicano tra di loro



Schema logico dei Replica Set per ogni network

La struttura sopra rappresentata è stata ripetuta (con delle reti dedicate) per i cluster Francia e Germania.

5. Implementazione del Database

Spiegazione teorica di come avviene la connessione di un utente

Abbiamo pensato di lasciare all'utente l'ultima parola sui libri che vuole acquistare e visionare in base alla lingua preferita. In un ipotetico portale web ci sarà l'opzione di selezione Paese. Fisicamente questa opzione permetterà all'utente di cambiare cluster da Italia, il suo Paese geografico, a Francia o Germania. Tutto è salvato in sessioni come segue nella voce "language" e permette un accesso rapido per gli accessi successivi.

Di default il cluster scelto è quello più vicino alla posizione geografica dell'utente. Quando si cambia cluster si effettuerà di fatto l'accesso allo stesso portale ma con ordini e acquisti differenti. Questo viene fatto in un'ottica scalabile e di grandi dimensioni in cui ogni utente acquista da fornitori esteri e segue tutte le politiche d'acquisto di paesi differenti, perciò la scelta migliore ci è sembrata quella di separare logicamente le cose (ulteriore motivazione è che per effettuare statistiche valide nei cluster abbiamo bisogno di frammentare i dati per Paese).

```
Chiave casuale recuperata: 66810c367e0431512d5cabe5
{
    "user_id": "66810c367e0431512d5cabe5",
    "login_time": "2024-06-30T08:38:44.181389+00:00",
    "last_activity": "2024-06-30T08:38:44.181389+00:00",
    "session_data": {
        "preferences": {
            "theme": "dark",
            "language": "it"
        }
    }
}
```

Costruzione dei nodi e dei cluster

Una volta stabilita la necessità di avere cluster di nodi separati per ogni regione geografica e quindi per ogni lingua, abbiamo deciso di implementare (sia con MongoDB che con Riak) tre cluster composti da tre nodi ciascuno: i cluster Italia, Germania e Francia.
Come suggeriscono i nomi, infatti, l'idea è quella di mantenere un partizionamento a livello geografico dei dati, così da rendere più efficienti i singoli nodi, ridurre i tempi necessari al soddisfacimento delle query e, all'occorrenza, poter fare statistiche localizzate sui vari utenti.
Nonostante non abbiamo integrato questa funzionalità nel nostro progetto, uno sviluppo interessante tra quelli affrontati durante il corso potrebbe essere proprio l'implementazione di alcuni script che forniscano statistiche localizzate sui generi di libri - o i singoli titoli - più letti nei vari Paesi, e organizzarne così in modo più efficiente la distribuzione. Inoltre, con una funzione Map-Reduce si potrebbero ottenere statistiche globali.
L'implementazione descritta è agevolata proprio dal fatto che, di base, in ogni area geografica si venderanno libri nella lingua di appartenenza.

Abbiamo dunque installato Docker sulle nostre macchine e seguito le guide ufficiali nella documentazione di Riak e MongoDB per scaricare le immagini *basho/riak-kv:latest* e *mongo:latest* e creare poi nove container organizzati in tre cluster.
Nonostante l'interfaccia grafica di Docker, dopo alcune prove abbiamo stabilito che fosse più conveniente inizializzare i vari nodi tramite una CLI (come il cmd di Windows), così da poter specificare direttamente alcuni parametri specifici come la porta del nodo, la rete e il ReplicaSet di appartenenza.
Come prima cosa si crea una rete per ogni cluster (reti che noi abbiamo chiamato riakItalia, riakFrancia, riakGermania, clusterItalia, clusterFrancia, clusterGermania).

- I nodi Riak sono stati chiamati:
  -Italia1, Italia2, Italia3
  -Francia1, Francia2, Francia3
  -Germania1, Germania2, Germania3
- I nodi MongoDB invece si chiamano:
  -libreria1, libreria2, libreria3 (Italia)
  -libreria4, libreria5, libreria6 (Francia)
  -libreria7, libreria8, libreria9 (Germania)

Per MongoDB, inoltre, abbiamo specificato il Replica Set di ogni cluster – rispettivamente ReplicaSetItalia, ReplicaSetFrancia, ReplicaSetGermania.

Per una vista completa su tutti i comandi eseguiti, consultare il file 'Comandi-Mongo-Riak.txt'.

Consistenza dei Dati e Replica Set

Replica Set per i Cluster Italia, Francia e Germania

Un Replica Set in MongoDB è un gruppo di istanze di mongod che mantengono gli stessi dati. Il replica set fornisce ridondanza e aumenta la disponibilità dei dati. Questo significa che se un nodo fallisce, un altro nodo può prendere il suo posto senza interrompere il servizio.
I nodi di un Replica Set in MongoDB possono essere:

- **Primary**: Il nodo che riceve tutte le operazioni di scrittura. Ci può essere solo un nodo primario in un replica set.

- **Secondary**: I nodi che replicano i dati dal nodo primario. Possono servire operazioni di lettura (se configurati per farlo) e diventare il nodo primario se l'attuale primario fallisce.
- **Arbiter**: Un nodo che partecipa alle elezioni per determinare il nuovo nodo primario ma non mantiene una copia dei dati. È utile per mantenere un numero dispari di voti nel replica set.

Avendo solo tre nodi nel nostro esempio abbiamo optato per la soluzione mostrata nell'immagine, in cui i tre nodi hanno tre priorità diverse, così che se il nodo 'primary' non fosse disponibile, sarebbe il terzo nodo a diventare automaticamente il master del Replica Set. In questo esempio dunque non è necessaria la presenza di un nodo arbitro.

```
ReplicaSetItalia [direct: primary] test> config.members[0].priority = 1
1
ReplicaSetItalia [direct: primary] test> config.members[1].priority = 0.5
0.5
ReplicaSetItalia [direct: primary] test> config.members[2].priority = 0.75
0.75
ReplicaSetItalia [direct: primary] test> rs.reconfig(config)
{
  ok: 1,
  '$clusterTime': {
    clusterTime: Timestamp({ t: 1719742605, i: 1 }),
    signature: {
      hash: Binary.createFromBase64('AAAAAAAAAAAAAAAAAAAAAAAAAAA=', 0),
      keyId: Long('0')
    }
  },
  operationTime: Timestamp({ t: 1719742605, i: 1 })
}
```

Vantaggi del Replica Set:

- **Disponibilità**: Se il nodo primario fallisce, i secondari possono eleggere un nuovo primario.
- **Scalabilità delle letture**: Le letture possono essere distribuite tra i nodi secondari.
- **Ridondanza**: I dati sono replicati su più nodi, riducendo il rischio di perdita di dati.

Abbiamo deciso di utilizzare cluster con replica set in quanto i dati utilizzati non risultano eccessivamente grandi (sono solo stringhe di testo). Per quanto riguarda gli utenti, abbiamo pensato di non utilizzare uno sharding in quanto le richieste e gli ordini sul portale web non sarebbero così eccessive da necessitare uno shard. Il traffico non sarebbe così elevato dato l'argomento (libri) e quindi anche per gli utenti è più utile mantenere un replica set per la consistenza e per garantire in caso di guasti che la ridondanza di dati non crei disagi agli utenti.

Implementazione del quorum nei cluster

Nonostante i cluster siano composti da soli tre nodi, abbiamo comunque ipotizzato e provato a implementare il concetto di quorum in lettura e scrittura.
Data la nostra assunzione che il contesto non sia eccessivamente dinamico, si può richiedere una consistenza relativamente alta, e un quorum in lettura e scrittura più alto del classico 50%+1.
Di conseguenza, avendo tre nodi per cluster abbiamo implementato un quorum in lettura e scrittura di due nodi su tre. In generale la nostra idea è quella di un quorum che si avvicini anche al 75% dei nodi, anche se non avendo esperienza effettiva su applicazioni in contesti reali ci è difficile stabilire se sia effettivamente l'opzione migliore.

Quorum in Riak
Per quanto riguarda Riak, è sufficiente accedere al Riak Explorer dal browser e modificare le

opzioni ‘Read Quorum’ e ‘Write Quorum’ nei vari bucket, anche se per farlo bisogna prima creare un bucket-type nuovo (noi lo abbiamo chiamato ‘libreria’).
Un’altra proprietà che è possibile modificare per un cluster Riak è legata al Vector Clock, per la gestione delle versioni. Le due scelte possibili sono dvv_enabled e last_write_wins:

vector clocks

- dvv_enabled è impostato su true, abilitando l'uso dei dot-version vectors.
- last_write_wins è impostato su false, indicando che i conflitti devono essere gestiti utilizzando vector clocks (o DVV).

dvv_enabled abilitato è consigliato perché i DVV sono una versione migliorata dei vector clocks, offrendo una gestione dei conflitti più efficiente e scalabile; **last_write_wins** disabilitato è consigliato per dati critici dove è importante mantenere la cronologia delle versioni e risolvere i conflitti in modo deterministico.

Quorum in MongoDB
Per Mongo la soluzione migliore si è rivelata la modifica dei cosiddetti ‘Write Concern’ e ‘Read Concern’ del Replica Set tramite un semplice script python che usa i moduli `WriteConcern` e `ReadConcern` della libreria `pymongo` (file ‘mongo-concern.py’ nella cartella ‘scripts’).

write concern
Il write concern è un valore numerico che sostanzialmente corrisponde al quorum.
WriteConcern=2 indica dunque che un valore è da considerarsi scritto correttamente se è stato scritto su due nodi su tre.

read concern
Per le letture abbiamo impostato ReadConcern = “linearizable", assicurando così che i dati letti siano gli ultimi scritti. Questo valore di read concern garantisce la linearizzabilità, cioè che tutte le operazioni di lettura siano ordinate in modo consistente.
Altre opzioni possibili per il read concern sono "local" (che è il valore di default e restituisce dati dal nodo a cui vengono richiesti senza garantire che siano i più recenti) e “majority” (parametro che assicura che le letture considerino solo dati confermati dalla maggioranza dei nodi).

Inoltre, nonostante i nostri tre cluster facciano riferimento a tre differenti Paesi, si potrebbero implementare ulteriori cluster per ogni Paese così da snellire il flusso di richieste al singolo cluster, a patto che questi possano comunicare tra loro e mantenere comunque una consistenza a livello geografico.

Script e query utilizzati

Essenzialmente gli script sono stati divisi nel repository in base alle loro funzioni:

Script di visualizzazione:

Book_suggested

Lo script Book_suggested è lo script più interessante dell'intera sezione in quanto è quello che si occupa di visualizzare l'operato dell'inverted index e di conseguenza in base ad un utente hardcoded suggerisce in base alle sue letture 5 libri che potrebbero interessare all'utente

Spiegazione dettagliata:
il codice si connette a MongoDB utilizzando un URL specifico che include informazioni sul replica set, garantendo così una connessione affidabile e ridondante. Una volta stabilita la connessione, accede al database denominato lib-ita (/ lib-fra / lib-ted) e alle collezioni ordini e libri. La funzione get_isbn_list_from_mongodb cerca gli ordini associati a un utente specifico, identificato da un ObjectId hardcoded. Per ogni ordine trovato verifica se il pagamento è stato effettuato (payment_made=True) e, in caso positivo, raccoglie gli ISBN dei libri associati a questi ordini. Infine, la connessione a MongoDB viene chiusa e la lista degli ISBN raccolti viene restituita.

Successivamente `get_keywords_and_isbns_from_riak` che effettua una richiesta HTTP GET al database Riak. Questa funzione tenta di ottenere i dati associati a una parola specifica, controllando che la risposta sia nel formato atteso, ovvero una lista di ISBN. In caso di errori durante la richiesta o di formato di risposta inatteso, la funzione gestisce questi casi restituendo una lista vuota o segnalando l'errore.
`main`, inizia chiamando `get_isbn_list_from_mongodb` per ottenere la lista di ISBN. Utilizza poi un URL per inviare una richiesta MapReduce a Riak, configurando il corpo della richiesta per ottenere tutte le parole chiave presenti nell'indice invertito di Riak. Una volta ricevuta la risposta, la funzione elabora i risultati ottenuti estraendo le parole chiave e verificando per ciascuna di esse se gli ISBN corrispondenti sono presenti nella lista ottenuta da MongoDB.

Per ogni ISBN verificato, il codice crea una mappa (`isbn_count_map`) per contare la frequenza degli ISBN associati a quelli ottenuti da MongoDB. Alla fine del processo, ordina gli ISBN in base alla frequenza e seleziona i primi cinque più frequenti. Utilizzando i dati da MongoDB, stampa i titoli dei libri associati a questi ISBN.

Book_view
Questo codice Python è progettato per recuperare un libro casuale dal database MongoDB e stamparne i dettagli.

Spiegazione dettagliata:

La funzione principale del programma è get_random_book, che si occupa di connettersi al database MongoDB e di recuperare un libro casuale dalla collezione dei libri. All'inizio della funzione, si stabilisce una connessione a MongoDB utilizzando un URL specifico che include informazioni su tre server di replica set. Questo garantisce che il programma possa connettersi al database in modo ridondante, migliorando l'affidabilità della connessione
Una volta stabilita la connessione, il programma accede al database ed Il libro selezionato randomicamente con la libreria random viene poi stampato in formato JSON

Load_book_data

Questo codice Python è progettato per ottenere informazioni dettagliate su un libro sia da un'API esterna (Open Library) che da un database MongoDB. Il codice integra i dati provenienti da entrambe le fonti e li stampa in un formato leggibile

Spiegazione dettagliata:

La funzione **get_book_data_from_api** accetta un numero ISBN come parametro e utilizza l'API di Open Library per recuperare le informazioni sul libro corrispondente. La funzione costruisce l'URL dell'API con i parametri necessari e invia una richiesta GET. Se la richiesta ha successo (status code 200), la risposta JSON viene analizzata per estrarre i dati del libro, come il titolo, gli autori, la copertina, gli editori, la lingua, il numero di pagine e la data di pubblicazione.

La funzione **get_book_data_from_mongodb** si connette a MongoDB utilizzando l'URL di connessione e accede al database. Cerca un documento nella collezione che corrisponde all'ISBN fornito. Se trova il documento, estrae informazioni specifiche come il prezzo, la valutazione media e la disponibilità.

Se i dati vengono recuperati con successo dall'API, la funzione li stampa in un formato leggibile, elencando il titolo, gli autori, la copertina, gli editori, la lingua, il numero di pagine e l'anno di pubblicazione. Analogamente, se i dati vengono recuperati con successo da MongoDB, stampa il prezzo, la valutazione media e la disponibilità del libro.

Orders_view

Questo codice Python è progettato per recuperare un ordine casuale da un database MongoDB e stamparne i dettagli in formato leggibile.

Spiegazione dettagliata

La funzione **get_random_orders** si occupa di connettersi al database MongoDB e di recuperare un ordine casuale dalla collezione degli ordini. All'inizio della funzione, si stabilisce una connessione a MongoDB utilizzando un URL specifico che include informazioni su tre server di replica set. Questo garantisce che il programma possa connettersi al database in modo affidabile, utilizzando la ridondanza offerta dal replica set.
Una volta stabilita la connessione, il programma accede al database.

L'ordine selezionato viene poi stampato in formato JSON.

Session_view
Il programma utilizza la libreria requests per inviare richieste HTTP all'API di Riak e json per formattare i dati in modo leggibile. La funzione principale, get_random_session_key, si connette al database Riak utilizzando un URL fornito e richiede tutte le chiavi presenti nel bucket. Se il bucket contiene chiavi, la funzione seleziona una chiave casuale e invia una nuova richiesta per ottenere i dettagli della sessione associata a quella chiave. I dettagli della sessione vengono poi formattati e stampati in modo leggibile.

User_view
Questo codice Python è progettato per recuperare un utente casuale da un database MongoDB e stamparne i dettagli in un formato leggibile.

Script di popolamento:

generate_inverted_index

Questo codice Python esegue un'integrazione tra MongoDB, Open Library API e Riak per creare e aggiornare un indice invertito basato sugli abstract dei libri.

Aggiornamenti possibili:

in alcune versioni di riak con specifici plugin è possibile implementare un ttl per gli elementi dei bucket. Nel nostro specifico caso ciò non era implementabile e avremmo necessitato di un server python che periodicamente controllasse che il ttl fosse scaduto.
Se la versione di riak fosse invece compatibile si potrebbe implementare la parte di codice mancante.

spiegazione dettagliata

La funzione **get_abstract_from_isbn** prende un numero ISBN come input e utilizza l'API di Open Library per recuperare l'abstract del libro associato. La funzione invia una richiesta HTTP GET all'API di Open Library e analizza la risposta JSON per estrarre l'abstract, se disponibile.

La funzione **create_inverted_index** invece prende un abstract come input e crea un indice invertito. L'abstract viene tokenizzato, le parole comuni (stop words) vengono rimosse e le parole rimanenti vengono stemmizzate utilizzando Porter Stemmer. L'indice invertito viene creato come un dizionario che associa ogni parola alla posizione in cui appare nell'abstract.

La funzione **update_inverted_index** aggiorna l'indice invertito in Riak per una determinata parola. Verifica se un documento per la parola esiste già in Riak; se esiste, aggiorna la lista di ISBN associati alla parola, altrimenti crea un nuovo documento con l'ISBN fornito.

populate sessions_riak

Questo codice Python è progettato per simulare la creazione e il salvataggio di sessioni utente, utilizzando MongoDB per recuperare gli utenti e i libri, e Riak per salvare le sessioni.

Spiegazione dettagliata

La funzione get_mongo_client configura e restituisce un client MongoDB.

la funzione fetch_random_user_ids recupera un numero casuale di `user_id` dal database MongoDB. Si connette al database e alla collezione specificati, estrae tutti gli `_id` degli utenti e ne seleziona casualmente un certo numero (definito da `count`).

La funzione `fetch_random_books` recupera un numero casuale di libri dal database MongoDB.

Simile alla funzione per gli utenti, estrae tutti gli `_id` dei libri e ne seleziona un numero casuale compreso tra 1 e `max_books`. Anche qui, se la collezione è vuota, solleva un'eccezione.

save_session_to_riak salva una sessione in Riak utilizzando una richiesta HTTP POST. Genera un `session_id` basato su `user_id`, prepara i dati della sessione e invia la richiesta. Se la richiesta ha successo, stampa una conferma; altrimenti, stampa un messaggio di errore.

Funzione `simulate_sessions`

Questa funzione è il cuore del programma. Esegue i seguenti passaggi:

1. Configura il client MongoDB.
2. Imposta l'URL del bucket Riak e il TTL per le sessioni.
3. Imposta il TTL per il bucket chiamando `set_bucket_ttl`.
4. Definisce il numero di sessioni da creare.
5. Per ogni sessione da creare:
    - Recupera un `user_id` casuale da MongoDB.
    - Recupera un numero casuale di libri (massimo 5) da MongoDB.
    - Crea i dati della sessione, includendo l'ID utente, l'ora di login, l'ultima attività e le preferenze.
    - Salva la sessione simulata in Riak utilizzando `save_session_to_riak`.

populate_libri_mongodb

Questo codice Python è progettato per recuperare dati sui libri utilizzando l'API di Open Library per una lista di ISBN, generare dati aggiuntivi fittizi come il prezzo, la valutazione media e la disponibilità, e inserire i dati risultanti in una collezione MongoDB.

Spiegazione dettagliata

la funzione **get_books_data** accetta una lista di ISBN e utilizza l'API di Open Library per recuperare i dati sui libri e per ciascun ISBN nella lista viene costruito l'URL per la richiesta API utilizzando l'ISBN corrente ,
Se i dati del libro sono disponibili, la funzione estrae il titolo, gli autori, la copertina e altri dettagli. Genera inoltre dati aggiuntivi come il prezzo (valore casuale tra 5 e 50 dollari), la valutazione media (valore casuale tra 1 e 5) e la disponibilità (numero casuale tra 0 e 1000).
I dati filtrati e arricchiti vengono aggiunti a una lista books_data. La funzione restituisce la lista books_data contenente i dati dei libri.

populate_order_mongodb

Questo codice Python è progettato per generare e inserire ordini fittizi in un database MongoDB, utilizzando dati fittizi per gli utenti, i libri e gli ordini.

Spiegazione dettagliata

La funzione **to_datetime** converte un oggetto datetime.date in un oggetto datetime.datetime, necessario per memorizzare correttamente le date nel database MongoDB.

la funzione **generate_order** genera un ordine fittizio: Viene selezionato un user_id casuale dalla lista degli ID degli utenti , determinato un numero casuale di libri (da 1 a 5) e vengono selezionati i corrispondenti ObjectId casuali dalla lista degli ID dei libri.

Viene generata una data di acquisto casuale compresa nell'ultimo anno.
Viene calcolato il totale speso come somma di prezzi casuali per ciascun libro, con prezzi compresi tra 5 e 50 dollari, Viene determinato casualmente se il pagamento è stato effettuato o meno.

Se il pagamento è stato effettuato, viene generata una data di consegna casuale compresa tra la data di acquisto e oggi. Altrimenti, la data di consegna è None, Viene creato un dizionario che rappresenta l'ordine con tutte le informazioni generate.

populate_users_mongodb
Questo codice Python è progettato per generare dati fittizi per utenti e inserirli in una collezione MongoDB. Utilizza la libreria Faker per generare dati realistici come nomi, indirizzi e-mail, numeri di telefono e altre informazioni personali.

Spiegazione dettagliata

la funzione **generate_fake_user** genera un utente fittizio con nome , cognome , email , indirizzo , telefono, data di nascita , sesso , username , password, data creazione
Il codice specifica il numero di utenti da generare (100 in questo caso) e utilizza un ciclo per creare una lista di utenti fittizi chiamando generate_fake_user ripetutamente. Gli utenti generati vengono poi inseriti nella collezione utenti di MongoDB utilizzando insert_many.

6. Prove di Utilizzo del Database

Descrizione dei Test Effettuati

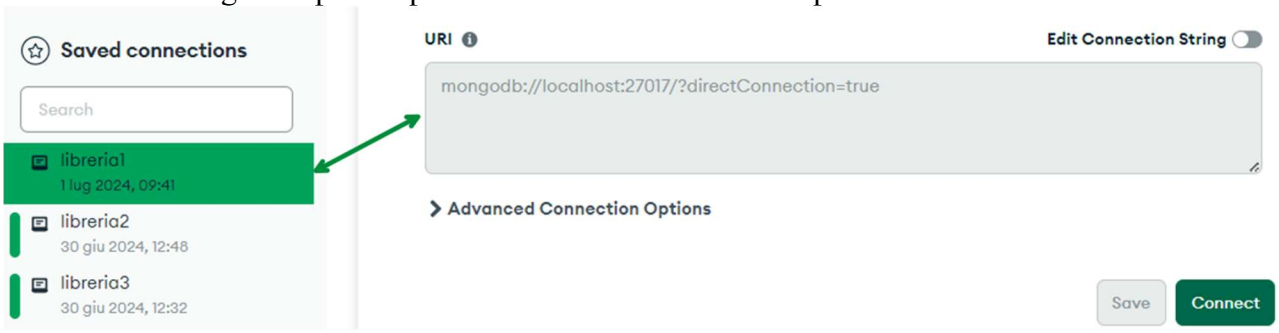
- Test di carico e performance
- Test di integrità dei dati

  Durante la progettazione dei database sono stati effettuati numerosi test per evidenziare il funzionamento del replica set.

  Funzionamento effettivo del replicaset MondoDB:

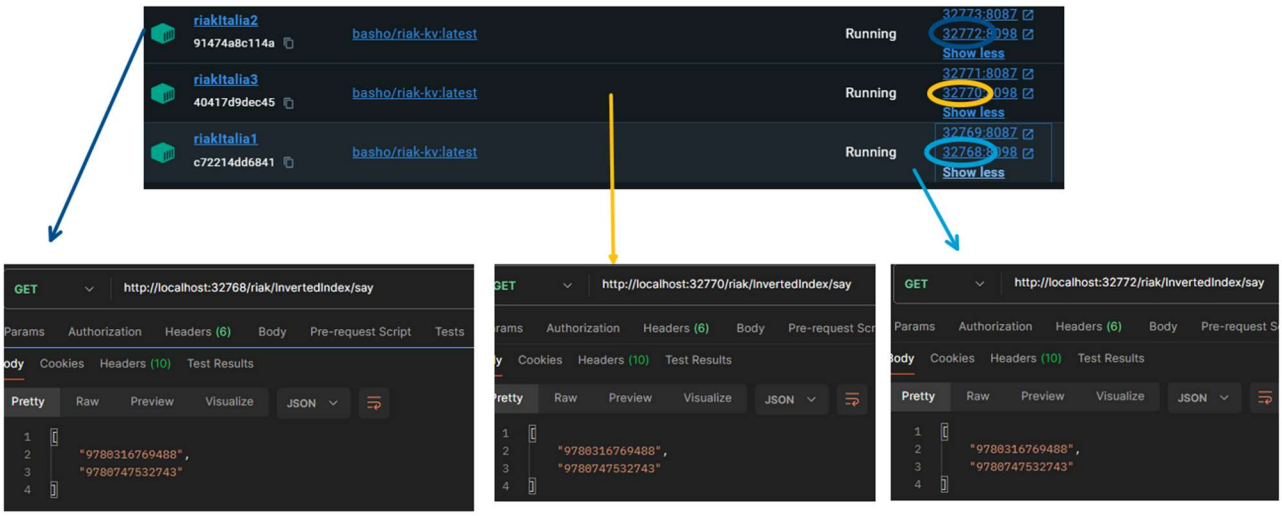
  Abbiamo utilizzato connessioni dirette sui nodi per verificare se esistesse un master.

Utilizzando MongoCompass la procedura è risultata molto semplice:



Funzionamento struttura ad anello riak:

per quanto riguarda riak non è stato usato alcun software GUI in quanto non esiste nessun progetto ufficiale. Abbiamo utilizzato Postman per il test effettivo sull'inserimento dei dati così come si può notare:



I dati risultano replicati correttamente su tutti i 3 nodi

Evidenza dell'Utilizzo del Database

- Esempi di inserimento dati nell'inventario

Dati inseriti in maniera hardcoded attraverso lo script python

Populate_libri_mongodb.py

Ovviamente questo è solamente un esempio di inserimento , data la vasta presenza di libri , editori e tipologie di libri (Narrativa italiana , Fumetti giapponesi , 3Dbook , …) Siamo consci del fatto che la struttura del DB non sarà fissa. Questo avvalora l'uso di DB NoSQL che non necessitano di una struttura fissa ma giustificano schemi flessibili e varietà di modelli.

Allo stesso modo saranno inseriti utenti ed ordini , per maggiori dettagli vedere la sezione relativa al codice populate_order_mongodb e populate_users_mongodb

- Esempi di recupero dati sugli acquisti

Abbiamo simulato attraverso lo script python i dati che dovranno essere mostrati. In un'ottica di portale web ovviamente verranno filtrati i dati da mostrare all'utente , verranno fatte le opportune "join" con i libri e gli utenti in modo da mostrare i titoli e verrà nel caso utilizzato un file css per rendere meglio visibili i dati.

Sono stati allo stesso modo predisposti script di configurazione python per sessioni Riak , utenti, e libri.

7. Conclusioni

Risultati Ottenuti

- **Gestione Efficace dell'Inventario dei Libri**: Utilizzando MongoDB per la gestione dell'inventario, siamo riusciti a creare una struttura flessibile e scalabile che permette l'inserimento, la modifica e l'eliminazione dei libri in modo rapido e sicuro. La struttura NoSQL ha permesso di adattarsi facilmente a cambiamenti nella struttura dei dati senza dover modificare l'intero schema del database.
- **Storico degli Acquisti degli Utenti**: Anche in questo caso, MongoDB ha dimostrato la sua efficacia permettendo di tracciare in modo dettagliato gli acquisti effettuati dagli utenti. Questo ha facilitato l'implementazione del sistema di raccomandazione, che può accedere rapidamente ai dati storici per suggerire libri pertinenti.
- **Sistema di Raccomandazione Efficace**: Riak è stato utilizzato per la gestione del sistema di raccomandazione. Grazie alla sua architettura distribuita e alla capacità di gestire grandi volumi di dati in modo efficiente, il sistema è in grado di analizzare gli acquisti precedenti degli utenti e suggerire libri in modo accurato e tempestivo.
- **Alta Disponibilità e Scalabilità**: La scelta di utilizzare replica set in MongoDB e configurazioni di cluster in Riak ha assicurato che il sistema possa gestire un aumento del traffico e garantire l'accesso continuo ai dati anche in caso di guasti a uno dei nodi. Questo ha aumentato la resilienza e la disponibilità complessiva del sistema.

Valutazione delle performance del sistema

- **Test di Carico e Performance**: Durante i test di carico, il sistema ha dimostrato di poter gestire un elevato numero di richieste simultanee senza un degrado significativo delle prestazioni. MongoDB ha gestito efficacemente le operazioni di lettura e scrittura grazie alla configurazione del replica set, mentre Riak ha mostrato tempi di risposta rapidi per le operazioni di raccomandazione dei libri.
- **Integrità dei Dati**: I test di integrità hanno confermato che i dati sono stati replicati correttamente tra i nodi dei vari cluster. Anche in caso di guasti simulati, i dati non sono stati persi grazie ai meccanismi di replica e quorum implementati.
- **Elasticità del Sistema**: Grazie alla natura distribuita dei database NoSQL utilizzati, il sistema ha dimostrato di poter scalare orizzontalmente in modo efficace. Questo significa che è possibile aggiungere nuovi nodi ai cluster esistenti per gestire un aumento del carico senza dover riprogettare l'intera architettura.

Criticità Riscontrate

In generale, non avendo esperienza diretta nell'ambito, le criticità riscontrate hanno riguardato soprattutto gli strumenti e le interfacce utilizzati (Docker, Postman, Riak Explorer, MongoDBCompass): questi hanno richiesto una fase iniziale di sperimentazione durante la quale abbiamo creato ed eliminato più volte i vari nodi e cluster. Questi vari tentativi ci hanno comunque permesso di costruire il file di testo contenente i comandi da eseguire per inizializzare un progetto simile in pochi minuti.

Possibili Miglioramenti Futuri

- Query per le statistiche

La raccolta e l'analisi dei dati sono fondamentali per svariati motivi: migliorare l'esperienza utente, ottimizzare le operazioni del sito e-commerce, prevedere e sopperire al meglio alle richieste di libri, e magari anche per l'implementazione di una pipeline di pulizia e analisi dei dati che li renda vendibili ad aziende terze. Dividendo i cluster per Paese, è possibile raccogliere dati dettagliati su vendite e traffico. Esempi di statistiche che possono essere generate includono:

Libri più venduti: localmente o globalmente, per identificare le tendenze di vendita e migliorare l'inventario (anche in base alla stagionalità);

Traffico Utenti: monitorare il traffico per Paese, identificando le aree con maggior afflusso di utenti e pianificando campagne di marketing mirate;

Tempo Medio di Acquisto: analizzare il tempo medio impiegato dagli utenti per completare un acquisto, migliorando l'interfaccia utente e i processi di checkout.

- Scalabilità:

L'obiettivo del progetto è ottenere un prodotto altamente scalabile sia verticalmente che orizzontalmente. La scalabilità verticale può essere ottenuta aggiungendo più risorse ai nodi esistenti(come potenziare RAM , CPU, ecc), mentre quella orizzontale può essere ottenuta aggiungendo più nodi ai cluster. La nostra idea è quella di utilizzare una scalabilità orizzontale in caso l'utenza sia maggiore e quindi ci sia maggior traffico con su una mole di dati di potenziali grandi dimensioni; tuttavia esistono limiti, come la latenza di rete tra cluster geografici e la complessità nella gestione dei dati distribuiti.