**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. 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.

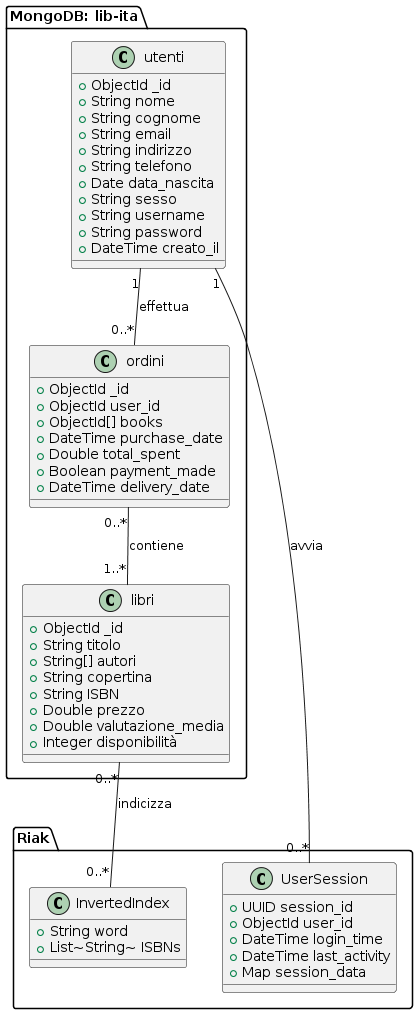
**MongoDB**

* Gestisce lo storico degli acquisti degli utenti.
* Utilizzato per proporre libri in base agli acquisti precedenti degli utenti.

**4. Progettazione del Database**

**Immagine che contiene testo, schermata, cerchio, Prodotti generali

Descrizione generata automaticamente**

 **Struttura dei database utilizzati e come comunicano tra di loro  
  
  
  
  
  
Schema logico delle funzionalità del progetto**

**Immagine che contiene schermata, Blu cobalto, testo

Descrizione generata automaticamente  
Schema logico dei replicaset per ogni network**

**Descrizione delle Tabelle e delle Relazioni**

Struttura delle collezioni e dei bucket, chiavi primarie e secondarie.

**5. Implementazione del Database**

**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 funzionaliltà nel nostro progetto, uno sviluppo interessante tra quelli affrontati durante le lezioni potrebbe essere proprio l’implementazione di alcuni semplici 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.

(la spiegazione appena fatta si può anche spostare, non deve stare qui)

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 sia più conveniente inizializzare i vari nodi tramite una CLI (come il cmd di Windows), così da poter specificare espressamente 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’.

**Decisione e 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 scrittura.  
La nostra assunzione è che il contesto non sia eccessivamente dinamico, per cui 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 (sia in Riak che in Mongo) un quorum in lettura e scrittura di due nodi su tre. In generale la nostra ipotesi 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.  
Per quanto riguarda Riak, è sufficiente accedere al Riak explorer e modificare le opzioni dei quorum nei vari bucket, mentre 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 il modulo ‘writeconcern’ della libreria pymongo (file ‘mongo-concern.py’ nella cartella ‘scripts’).

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 lo script 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, se così fosse, 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 funazione **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

**Evidenza dell'Utilizzo del Database**

* Esempi di inserimento dati nell'inventario
* Esempi di recupero dati sugli acquisti
* Analisi delle performance durante i test

**7. Conclusioni**

**Risultati Ottenuti**

* Successi nel raggiungimento degli obiettivi
* Valutazione delle performance del sistema

**Criticità Riscontrate**

* Problemi tecnici affrontati e soluzioni adottate

**Possibili Miglioramenti Futuri**

* Estensioni funzionali e miglioramenti architetturali

**8. Appendice**

**Codice SQL**

* Script di creazione delle collezioni e dei bucket
* Esempi di query utilizzate

**Diagrammi e Grafici**

* Diagrammi ER e schema logico
* Grafici di performance

**Altro Materiale di Supporto**

* Documentazione aggiuntiva e riferimenti

**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 globali.

**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.

**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.

**Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamente**

**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.

**Utilizzo di Replica Set in MongoDB**

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.

Aggiunta da telegram

Pasquale Manfredi, [30/06/2024 13:08]

Relazione del Progetto: Libreria-NOSQL

Introduzione

Il progetto "Libreria-NOSQL" si propone di realizzare un sistema di gestione per un sito e-commerce di libri utilizzando database NoSQL. Le tecnologie principali coinvolte sono MongoDB per la gestione degli utenti, dei libri e degli ordini, e Riak per implementare una funzionalità specifica di raccomandazione dei libri agli utenti in base ai loro acquisti. L'obiettivo è garantire un sistema scalabile, resiliente e ad alta disponibilità, capace di gestire grandi volumi di dati e di offrire un'esperienza utente fluida e personalizzata.

Configurazione di MongoDB

MongoDB è stato scelto come database principale per gestire gli utenti, i libri e gli ordini per diversi motivi. Innanzitutto, MongoDB è un database documentale che consente di gestire dati complessi in modo semplice ed efficiente. La sua flessibilità permette di adattare rapidamente la struttura dei dati alle esigenze del business.

Cluster e Replica Set

Per garantire l'alta disponibilità e la tolleranza ai guasti, MongoDB è stato configurato con tre cluster geografici: Italiano, Francese e Tedesco. Ogni cluster è stato configurato con la funzionalità di replicaset, che replica i dati su più nodi all'interno del cluster. Questa configurazione offre diversi vantaggi:

Alta Disponibilità: In caso di guasto di uno o più nodi, i dati rimangono accessibili grazie alle repliche presenti sugli altri nodi del cluster.

Tolleranza ai Guasti: La presenza di più repliche dei dati garantisce che, anche in caso di fallimento di un nodo, i dati non vengano persi.

Consistenza dei Dati: La replicazione assicura che tutti i nodi del cluster abbiano una copia aggiornata dei dati, riducendo il rischio di inconsistenze.

Sharding e Nodo Arbiter

La necessità di implementare lo sharding dipende dal volume dei dati che il sistema deve gestire. Lo sharding distribuisce i dati su più nodi, migliorando la scalabilità orizzontale e le performance di lettura e scrittura. Se si prevede un volume di dati molto elevato, lo sharding diventa necessario per evitare che un singolo nodo venga sovraccaricato. Tuttavia, se il volume dei dati è gestibile da un singolo cluster, lo sharding potrebbe non essere necessario.

Un nodo arbiter è utile in un replicaset con un numero pari di nodi. L'arbiter non contiene una copia dei dati, ma partecipa al processo di elezione del nodo primario in caso di failover. Aggiungere un arbiter può prevenire situazioni di split-brain, in cui due nodi potrebbero erroneamente ritenersi entrambi primari.

Configurazione di Riak

Riak è stato scelto per implementare la funzionalità di raccomandazione dei libri agli utenti e per gestire le sessioni degli utenti. Riak è un database key-value altamente scalabile, progettato per offrire un'elevata disponibilità e tolleranza ai guasti.

Cluster e Quorum

Similmente a MongoDB, Riak è stato configurato con tre cluster geografici, ognuno con tre nodi. La scelta di mantenere i cluster nelle stesse aree geografiche dei cluster MongoDB consente di ridurre la latenza e migliorare le performance delle operazioni distribuite.

Il quorum in Riak determina il numero di repliche che devono confermare una lettura o una scrittura prima che l'operazione venga considerata completata. Per un progetto che tratta di libri, un quorum di lettura e scrittura di 2 (su 3) è ragionevole. Questo garantisce un buon equilibrio tra consistenza e disponibilità dei dati.

Gestione delle Sessioni

Riak è utilizzato anche per memorizzare le sessioni degli utenti. Memorizzare i libri consigliati nella sessione può migliorare l'esperienza utente, consentendo di offrire raccomandazioni personalizzate in tempo reale. La durata della sessione (TTL) dovrebbe essere sufficientemente lunga da coprire una sessione di acquisto tipica, ad esempio 30 minuti. Questo assicura che le informazioni di sessione rimangano valide durante tutto il processo di acquisto.

Pasquale Manfredi, [30/06/2024 13:08]

Azioni dell'Utente e Interazione con i Database

Gli utenti interagiscono con il sistema attraverso diverse azioni, ognuna delle quali coinvolge specifici database e operazioni:

Registrazione e Login: Quando un utente si registra o effettua il login, le informazioni vengono salvate in MongoDB. Questo consente di gestire i profili utente in modo efficiente e sicuro.

Navigazione e Ricerca: Le ricerche effettuate dagli utenti e le loro preferenze di navigazione vengono tracciate e memorizzate. Questi dati vengono utilizzati per generare raccomandazioni personalizzate tramite Riak.

Acquisto Libri: Gli ordini effettuati dagli utenti vengono salvati in MongoDB. Le raccomandazioni basate sugli acquisti vengono gestite tramite Riak, sfruttando le capacità di gestione dei dati distribuiti.

Gestione del Carrello: I carrelli degli utenti sono mantenuti in MongoDB. Questa scelta garantisce la consistenza e l'integrità dei dati, evitando i potenziali problemi di affidabilità che possono derivare dall'uso delle sessioni per memorizzare i carrelli.

Considerazioni sui 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.

Motivazioni per l'Uso di MongoDB e Riak

MongoDB: È stato scelto per la sua capacità di gestire documenti complessi e relazioni tra dati. La configurazione con replicaset garantisce alta disponibilità e tolleranza ai guasti, rendendo MongoDB ideale per gestire utenti, ordini e carrelli.

Riak: È stato scelto per la gestione delle raccomandazioni e delle sessioni degli utenti. La sua capacità di gestire dati distribuiti con elevata tolleranza ai guasti lo rende adatto per queste funzionalità specifiche.

Sezione di Statistiche

La raccolta e l'analisi dei dati sono fondamentali per migliorare l'esperienza utente e ottimizzare le operazioni del sito e-commerce. 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: Per paese o globalmente, per identificare le tendenze di vendita e migliorare l'inventario.

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.

Consigli e Scalabilità

Utilizzando i dati delle vendite e delle interazioni degli utenti, è possibile sviluppare algoritmi di raccomandazione che suggeriscano i libri più venduti e quelli più adatti alle preferenze degli utenti. Questa funzionalità, sebbene non ancora implementata, rappresenta un'idea futura per migliorare ulteriormente l'esperienza utente.

Il progetto è altamente scalabile sia verticalmente che orizzontalmente. La scalabilità verticale può essere ottenuta aggiungendo più risorse ai nodi esistenti, mentre quella orizzontale può essere ottenuta aggiungendo più nodi ai cluster. Tuttavia, esistono limiti, come la latenza di rete tra cluster geografici e la complessità nella gestione dei dati distribuiti.

Conclusione

Il progetto "Libreria-NOSQL" è un esempio di come le tecnologie NoSQL possano essere utilizzate per creare un sistema di gestione per un sito e-commerce scalabile, resiliente e ad alta disponibilità. La combinazione di MongoDB e Riak offre un'architettura robusta e flessibile, in grado di gestire grandi volumi di dati e di offrire un'esperienza utente personalizzata e coerente.