Big Data Analytics

MongoDB

Marta Caffagnini
Matricola: 168007
307930@studenti.unimore.it

20/11/2021

Indexing e query optimization

Indice

1. Introduzione

Gli indici sono uno strumento molto importante perché, se usati correttamente, permettono a MongoDB di usare l'hardware in modo efficiente e rispondere alle query velocemente. Gli indici permettono di ridurre il lavoro richiesto per estrarre documenti. Usando gli indici correttamente, si evita di scansionare tutti i documenti per trovare l'output della query.

Se gli indici sono usati in modo scorretto, si ottiene l'effetto opposto e l'applicazione sarà più lenta.

1.1 Indici a una o più chiavi

1. Indice single-key. Ogni istanza dell'indice corrisponde a un singolo valore per ogni documento indicizzato. _id è l'indice di default ed è un single-key index. L'indice single-key può essere usato nelle query in cui si ha un match esatto, oppure l'ordinamento di un campo indicizzato, oppure quando la query contiene dei range.

2. Indice compound-key. Spesso il database viene interrogato su più di un attributo. Se si utilizzano più single-key index vengono create delle strutture dati distinte. Quando viene effettuata la query, viene scannerizzata ogni struttura dati e poi calcolata l'intersezione. Un indice compound è un indice la cui istanza è composta da più di una chiave.
   L'ordine delle chiavi è importante.
   Quando una query ha un termine che richiede un match esatto e l'altro termine richiede il valore di un range, la chiave con il range si mette per seconda.
   Se si ha un compound index 'a-b', un indice single-key 'a' sarebbe ridondante.
   L'indice compound, però, può rispondere efficacemente solo a un range o a un ordinamento per query.

1.2 Efficienza degli indici

Ogni indice comporta un costo di manutenzione. Quando si aggiunge o toglie un documento dalla collezione, ogni indice della collezione deve essere modificato per includere o eliminare il documento.
Per questo gli indici devono essere scelti con cura e bisogna assicurarsi che tutti gli indici siano utilizzati e non ridondanti.

Se gli indici non rendono le query più veloci, può essere che gli indici e i dati non siano in RAM.
I data file che contengono i documenti, le collezioni e gli indici sono spostati dentro e fuori dalla RAM dal sistema operativo.

Se la capacità della RAM è sufficiente tutti i data file sono caricati in memoria, altrimenti il sistema operativo continuerà ad accedere al disco.

Per evitare questa situazione gli indici devono essere pochi, necessari e caricati in RAM.

1.3 B-trees

MongoDB memorizza la maggior parte degli indici come B-trees: una struttura dati ideale per gli indici dei database per due motivi:

1. facilitano vari tipi di query come i match esatti, le condizioni con range, l'ordinamento, il match di prefisso, e le query con solo un indice.

2. hanno il vantaggio di rimanere bilanciati anche dopo l'aggiunta e la rimozione di chiavi.

Il B-tree è una struttura dati ad albero in cui ogni nodo può contenere più chiavi.

1.4 Tipi di indice

1. Indice unico. Questo indice è utilizzato per forzare il campo di un documento a non avere valori ripetuti. È più indicato creare l'indice prima di inserire i dati. Se la collezione già contiene dei dati, la creazione dell'indice può fallire, perché chiavi duplicate potrebbero già esistere. In questo caso o si rimuovono uno per volta i documenti con le chiavi ripetute, oppure con l'opzione dropDups vengono eliminati automaticamente tutti i documenti con chiavi ripetute.

2. Indice denso. Gli indici sono densi di default. In una collezione indicizzata esiste un'istanza dell'indice per ogni documento. Se questo non ha una chiave il valore di indice è null.

3. Indice sparso. L'indice sparso viene utilizzato se si vogliono indicizzare solo i documenti che hanno valori per la chiave. Se si vuole un indice sparso bisogna specificare {sparse: true}.

4. Indice multikey. Permette di usare più entry nell'indice per riferirsi allo stesso documento. Un indice è multikey se è creato su un campo il cui valore è un array.

5. Indice hashed. Le entry sono sottoposte a una funzione hash.
   La funzione hash prende un input e lo mappa in un valore di output di lunghezza fissata. L'obiettivo di questo procedimento è distribuire i valori in modo apparentemente casuale.
   Questi indici comportano delle restrizioni:
   - non sono supportate le query con range.
   - gli indici multikey hashed non sono permessi.
   - i valori float sono convertiti in interi prima di essere sottoposti alla funzione di hash.
   Questi indici sono utili per creare uniformità nella distribuzione delle chiavi e modificano la posizione delle istanze degli indici.

2. Gestione degli indici

2.1 Creazione ed eliminazione di indici

Per creare un indice si può chiamare createIndex() e verrà creato un documento che definisce il nuovo indice e lo posiziona nella collezione system.indexes.

Per eliminare un indice si utilizza il metodo dropIndex() fornendo il nome dell'indice.

Per controllare le informazioni riguardo gli indici si può utilizzare il metodo getIndexes().

2.2 Costruzione indici

Solitamente si dichiarano gli indici prima di mettere l'applicazione in produzione.
In due casi è meglio costruire l'indice dopo l'inserimento dei dati:

- se bisogna importare molti dati. Creare gli indici dopo l'importazione permette di avere indici bilanciati e compatti fin da subito. Si minimizza anche il tempo di costruzione dell'indice.

- quando si vogliono ottimizzare delle nuove query. Per esempio quando si aggiungono o cambiano le funzionalità di un'applicazione.

L'indice viene costruito in 2 step:

- I valori che devono essere indicizzati vengono ordinati, perché un data set ordinato rende più efficiente l'inserimento in un B-tree.

- I valori ordinati vengono inseriti nell'indice.

3. Ottimizzazione delle query

L'ottimizzazione delle query è un processo di identificazione delle query lente, delle cause e delle modifiche per velocizzare.

3.1 Identificazione delle query lente

Il primo step è l'identificazione delle query lente.
Generalmente le query di un'applicazione per restituire l'output non dovrebbero impiegare più di 100 ms.
Per identificare in modo agevole le query lente si può utilizzare il profiler di query presente in MongoDB.

Di default il profiler è disattivato, bisogna quindi attivarlo chiamando la funzione setProfilingLevel() sul database su cui si vuole fare profiling. Come parametro viene passato il livello del profiling.

Il livello 1 è per controllare solo le operazioni più lente di 100 ms.
Il livello 2 è il più verboso, perché viene analizzata ogni lettura e scrittura.
Per disattivare il profiler si passa il parametro 0.
Per cambiare i millisecondi si passa il valore desiderato come secondo parametro.

3.2 Analisi delle query lente

Le cause di query lente sono molteplici.

Il comando explain() fornisce informazioni dettagliate sul percorso di una query e viene chiamato sulla query che si vuole analizzare.
L'output contiene vari campi tra cui:

- *nReturned* che indica il numero di documenti restituiti come output.

- executionTimeMillis che indica il tempo che impiega la query a restituire il risultato.

- *totalDocsExamined* mostra quanti documenti sono stati scanditi. Questo valore viene confrontato con il numero di documenti presenti nel database.

- *cursor* indica il cursore che si sta usando. Se il suo valore è BasicCursor significa che sta avvenendo la scansione della collezione e non di un indice. Se venisse utilizzato un indice, il valore sarebbe BTreeCursor.

- *scanAndOrder* ha valore true se l'ottimizzatore della query non può usare un indice per restituire i risultati ordinati; di conseguenza oltre a scandire la collezione, il risultato deve essere ordinato manualmente.

- *totalKeysExamined* mostra il numero di istanze di index che MongoDB ha scansionato.

L'obiettivo è avere i valori di *nReturned* e *totalDocsExamined* il più simili possibile.

Per velocizzare le query lente viene creato un indice.

3.3 L'ottimizzatore di query di MongoDB

L'ottimizzatore di query è una parte di software che determina quale indice, se esiste, soddisfa più efficacemente una query.

L'ottimizzatore è basato sulle seguenti regole:

1. Evitare scanAndOrder, quindi se la query richiede l'ordinamento l'ottimizzatore cerca di usare un indice.

2. Soddisfare tutti i campi con vincoli di indicizzazione utili. L'ottimizzatore tenta di utilizzare gli indici per i campi nel selettore di query.

3. Se la query include un range o un ordinamento, sceglie un indice dove l'ultima chiave usata può aiutare a soddisfare il range o l'ordinamento.

Se un indice soddisfa tutte e 3 le condizioni allora è ottimale e viene utilizzato. Se più indici sono ottimali, ne viene scelto uno arbitrariamente.

Se la query contiene due campi che sono indicizzati non è immediato sapere quale indice utilizzare.
La scelta si basa sul valore *totalDocsExamined*. L'ottimizzatore sceglie l'indice che richiede la scansione del minor numero di documenti e viene memorizzato il piano migliore per un uso futuro.

4. Esempi di query ottimizzate

Il dataset Amazon utilizzato per l'esercitazione è lo small subsets con le review sui prodotti della categoria Video Game. Il dataset è formato da 497577 documenti.

Il seguente è un esempio di review presente nel dataset:

{

"overall": 5.0,
"vote": "2",
"verified": True,
"reviewTime": "01 1, 2018",
"reviewerID": "AUI6WTTT0QZYS",
"asin": "5120053084",
"reviewerName": "Abbey",
"reviewText": "I now have 4 of the 5 available colors of this shirt... ",
"summary": "Comfy, flattering, discreet--highly recommended!",
"unixReviewTime": 1514764800

}

Le tecniche utilizzate riassunte in precedenza sono il single index, compound index, sparse index e hashed index.

4.1 Query 1

- **Qual è la recensione con overall più alto di Vincent G. Mezera?**

```
db.reviews.find({"reviewerName":"Vincent G. Mezera"}).sort({overall:-1}).limit(1);
```

Viene utilizzato il comando explain("executionStats") sulla query in modo da avere informazioni dettagliate sull'esecuzione della query:

```
db.reviews.find({"reviewerName":"Vincent G.
Mezera"}).sort({overall:-1}).limit(1).explain("executionStats");
```

Il comando restituisce:

nReturned	1
executionTimeMillis	691
totalKeysExamined	0
totalDocsExamined	497577

nReturned indica il numero di documenti che fanno match con le condizioni scritte nella query. In questo caso è 1 perchè si vuole ottenere solo la recensione con overall più alto tra quelle scritte da Vincent G. Mezera.

Il valore di executionTimeMillis indica il tempo di selezione del piano di query e il tempo di esecuzione della query espresso in millisecondi.

Il valore totalKeysExamined indica il numero di istanze di indice scansionate. Nessun indice è stato creato, infatti il numero di chiavi esaminate è 0.

totalDocsExamined indica il numero di documenti esaminati durante l'esecuzione della query. Il numero di documenti è 497577. Questo è il numero totale di documenti presenti nella collezione, quindi la query scansiona ogni documento. Si deduce che avviene lo scanAndOrder, che può essere evitato creando un indice che verrà poi utilizzato dall'ottimizzatore di query.

Per ottimizzare la query è stato creato un compound index sui campi reviewerName e overall, che sono i campi su cui viene interrogato il database.

MongoDB generalmente usa solo un indice nella query, quindi interrogazioni su campi multipli come questa richiedono compound index per essere efficienti.

L’indice avrà i valori di overall messi in ordine decrescente così sarà più efficiente trovare la recensione con l’overall più alto. L’ordine dei campi nel compound index è importante. Come primo campo viene messo reviewerName perchè la query richiede un match esatto su quel campo.

```
db.reviews.createIndex({reviewerName:1,overall:-1});
```

Eseguendo nuovamente la query con il metodo explain() si ottiene:

nReturned	1
executionTimeMillis	8
totalKeysExamined	1
totalDocsExamined	1

Il tempo di esecuzione e il numero di documenti scansionati si è ridotto. Viene esaminata una sola chiave e viene restituito un solo documento.

Cancellazione indice:

```
db.reviews.dropIndex( "reviewerName_1_overall_-1")
```

4.2 Query 2

- **Elencare le recensioni del prodotto 0804161380**

```
db.reviews.find({asin:"0804161380"})
```

Viene utilizzato il comando explain("executionStats") sulla query.

```
db.reviews.find({asin:"0804161380"}).explain("executionStats")
```

Il comando restituisce:

nReturned	49
executionTimeMillis	11496
totalKeysExamined	0
totalDocsExamined	497577

Vengono sempre esaminati tutti i documenti presenti nella collezione. Si deduce che avviene lo scanAndOrder.

Viene creato un single index sul campo asin su cui avviene la ricerca dei documenti.

```
db.reviews.createIndex({asin:1});
```

Eseguendo nuovamente la query con il metodo explain() si ottiene:

nReturned	49
executionTimeMillis	3
totalKeysExamined	49
totalDocsExamined	49

L’indice creato è stato utilizzato e il tempo di esecuzione si è ridotto.

Cancellazione indice:

```
db.reviews.dropIndex( "asin_1")
```

4.3 Query 3

- **Qual è il prodotto più recensito il giorno 09-02-2015?**

```
db.reviews.aggregate([
	{ $match: { reviewTime: "09 2, 2015"}},
	{ $group: { _id: "$asin" , count: { $sum: 1 } }},
	{ $sort: { count: -1 }},
	{ $limit: 1 }
])
```

Viene utilizzato il comando explain("executionStats") sulla query.

```
db.reviews.explain("executionStats").aggregate([
	{ $match: { reviewTime: "09 2, 2015"}},
	{ $group: { _id: "$asin" , count: { $sum: 1 } }},
	{ $sort: { count: -1 }},
	{ $limit: 1 }
])
```

Il comando restituisce:

nReturned	1

executionTimeMillis	858
totalKeysExamined	0
totalDocsExamined	497577

Vengono sempre esaminati tutti i documenti presenti nella collezione. Si deduce che avviene lo scanAndOrder, che può essere evitato creando un indice che verrà poi utilizzato dall'ottimizzatore di query.

Per ottimizzare la query è stato creato un compound index sui campi reviewerTime e asin, che sono i campi su cui avviene la ricerca dei documenti che soddisfano la query. Entrambi i campi sono stati indicizzati in ordine crescente.

```
db.reviews.createIndex({reviewTime:1,asin:1});
```

Eseguendo nuovamente la query con il metodo explain() si ottiene:

nReturned	1
executionTimeMillis	1
totalKeysExamined	224
totalDocsExamined	1

Il tempo di esecuzione si è molto ridotto e la ricerca è avvenuta utilizzando l'indice creato (totalKeysExamined è maggiore di 0).

Cancellazione indice:

```
db.reviews.dropIndex( "reviewTime_1_asin_1")
```

4.4 Query 4

- **Quali sono le recensioni del prodotto B0000E5U6I con overall maggiore di 2?**

```
db.reviews.find(
    { $and: [ { asin: { $in:["B0000E5U6I"] } }, { overall: { $gt: 2 } } ] }
)
```

Viene utilizzato il comando explain("executionStats") sulla query.

```
db.reviews.find(
      { $and: [ { asin: { $in:["B0000E5U6I"] } }, { overall: { $gt: 2 } } ] }
```

```
).explain("executionStats")
```

Il comando restituisce:

nReturned	106
executionTimeMillis	788
totalKeysExamined	0
totalDocsExamined	497577

Vengono sempre esaminati tutti i documenti presenti nella collezione. Si deduce che avviene lo scanAndOrder.

Per ottimizzare la query è stato creato un compound index sui campi asin e overall, che sono i campi su cui sono poste le condizioni della query. Nel compound index l'ordine delle chiavi è importante.
La query ha un termine che richiede un match esatto (asin) e l'altro richiede il valore di un range (overall), la chiave con il range si mette per seconda.

```
db.reviews.createIndex({asin:1, overall:1});
```

Eseguendo nuovamente la query con il metodo explain() si ottiene:

nReturned	106
executionTimeMillis	3
totalKeysExamined	106
totalDocsExamined	106

La query è stata ottimizzata e il valore di executionTimeMillis è diminuito.

Cancellazione indice:

```
db.reviews.dropIndex( "asin_1_overall_1")
```

4.5 Query 5

- **Elencare recensioni con 2 come valore di vote**

```
db.reviews.find( { "vote": "2" } );
```

Viene utilizzato il comando explain("executionStats") sulla query.

```
db.reviews.find( { "vote": "2" } ).explain("executionStats");
```

Il comando restituisce:

nReturned	30860
executionTimeMillis	748
totalKeysExamined	0
totalDocsExamined	497577

Vengono sempre esaminati tutti i documenti presenti nella collezione. Si deduce che avviene lo scanAndOrder, che può essere evitato creando un indice che verrà poi utilizzato dall'ottimizzatore di query.

Per ottimizzare la query viene creato un indice sparso, perché non tutte le recensioni hanno un valore nel campo vote. L'indice sparso permette di indicizzare solo i documenti che hanno valori per la chiave vote senza occupare memoria memorizzando indici con valori null.

```
db.reviews.createIndex({vote:1}, { sparse: true });
```

Eseguendo nuovamente la query con il metodo explain() si ottiene:

nReturned	30860
executionTimeMillis	107
totalKeysExamined	30860
totalDocsExamined	30860

Il tempo di esecuzione è calato.

Se avessi usato l'indice classico, questo avrebbe occupato 2,2 MB, mentre l'indice sparse occupa 630 KB.

Cancellazione indice:

```
db.reviews.dropIndex( "vote_1")
```

4.6 Query 6

- **Elencare le recensioni verificate del prodotto "0700026657"**

```
db.reviews.find(
      { $and: [ { asin: { $in:["0700026657"] } }, { verified: true } ] }
)
```

Viene utilizzato il comando explain("executionStats") sulla query.

```
db.reviews.find(
	{ $and: [ { asin: { $in:["0700026657"] } }, { verified: true } ] }
).explain("executionStats");
```

Il comando restituisce:

nReturned	10
executionTimeMillis	6281
totalKeysExamined	0
totalDocsExamined	497577

Vengono sempre esaminati tutti i documenti presenti nella collezione.

Per ottimizzare la query è stato creato un compound index sui campi asin e verified, che sono i campi su cui sono poste le condizioni della query. Sulla chiave asin è stato creato un indice hash che garantisce una più uniforme distribuzione dei dati.

```
db.reviews.createIndex({asin:"hashed", verified:1 });
```

Eseguendo nuovamente la query con il metodo explain() si ottiene:

nReturned	10
executionTimeMillis	8
totalKeysExamined	10
totalDocsExamined	10

Il tempo di esecuzione è stato ottimizzato e l'indice creato è stato utilizzato nella ricerca.

Cancellazione indice:

```
db.reviews.dropIndex( "asin_hashed_verified_1")
```