MongoDB Implementation

Aggregations

retrieveUsersInformation()

This simple aggregation query is used to retrieve the UserID and the number of researches done for each user to allow the administrators to have an overview of the most active users

results = collection.aggregate(Arrays.asList(

new Document("$group", new Document("\_id","$userID")

.append("value", new Document("$sum", 1))),

new Document("$sort", new Document("value", -1))));

calculateTrendingKeyWords()

This aggregation query selects the articles from the queryDate to the current date. From each document the keywords are taken and grouped.

For each keyword the occurrence on each article are summed and saved in "Occur" and the number of Article in which the keyword is present is saved in "NumberOfArticles".

This informations are used to compute the formula for the trending keywords of the specified period. The trending keywords are the first 500 keywords ordered by value.

Value of the keyword = (Number of total occurrences) \* (Number of Article Containing Keyword)^2

This formula gives more importance to the quantity of Article in which a specific keyword is found than the total number of occurrences. This is needed to avoid to giving importance to keywords present in fewer articles but with an elevated number of occurrences.

//ValueOfKeyword=NumberOfArticles^2\*Occ

results = collection.aggregate(Arrays.asList(

new Document("$match", new Document("date", new Document("$gt",

queryDate))),

new Document("$unwind", "$Keywords"),

new Document("$group", new Document("\_id", "$Keywords.keyword")

.append("Occur", new Document("$sum", "$Keywords.Occ"))

.append("NumberOfArticles", new Document("$sum", 1))),

new Document("$project",new Document("\_id",1)

.append("Value", new Document("$multiply",indexes) )),

new Document("$sort", new Document("Value", -1)),

new Document("$limit",500)));

suggestedArticles(User u)

This aggregation query retrieve the most used “filters” by an user in a specific period of time (form queryDate to the current Date) the top three filters retrieved are used to find articles suggested for the user

results = collection.aggregate(Arrays.asList(

Aggregates.match(and(eq("userID",u.userID),gte("dateRead",queryDate))),

new Document("$group", new Document("\_id", "$filters")

.append("value", new Document("$sum", 1))),

new Document("$sort", new Document("value", -1)),

new Document("$limit", 3)));

Indexes

 Indexes

public static void createIndexes() {

MongoCollection<Document> collection =database.getCollection("Article");

BasicDBObject obj = new BasicDBObject();

obj.put("Topic", 1); //

obj.put("Date", -1);

collection.createIndex(obj);

obj = new BasicDBObject();

obj.put("Keywords.keyword", 1);

obj.put("Date", -1);

collection.createIndex(obj);

collection = database.getCollection("Users");

obj = new BasicDBObject();

obj.put("userID", 1);

collection.createIndex(obj);

}

From an analysis we have individuated the heavy read operations on our Database in particular:

The user authentication is an operation done with an elevate frequency

Write: Once an user is saved in our Database no more write operation is required for that user

Read: Assuming that on average an user access to our service 2 times a day

Assuming 200 users are using our service every day

Statistics write vs read in one month

+ 200 writes on the user collection

+ 200\*2\*32 = 12800 read operations

MongoDB Results :

Query:

db.Users.find({"userID":"RiccardoXe"}).explain("executionStats")

Without Index

"executionStats" : {

"executionSuccess" : true,

"nReturned" : 1,

"executionTimeMillis" : 1,

"totalKeysExamined" : 0,

"totalDocsExamined" : 1602,

…

}

With index

"executionStats" : {

"executionSuccess" : true,

"nReturned" : 1,

"executionTimeMillis" : 0,

"totalKeysExamined" : 1,

"totalDocsExamined" : 1,

…

}

Usually users are interested in specific topics.

Taking the example above, if 100 users access to our service twice a day, with a great probability they will make a search by the tag they are interested in

Statistics write vs read

10/20 articles for a specific topic per day assuming 4 major topics

+ 20\*4=80 write per day

+ 2\*200=400 read operation by topic per Day

Obviously, this index is more useful in a scenario where an elevated number of users use our application every day.

Query

db.Article.find({"Topic":"Serie A"}).explain("executionStats")

Without Index

"executionStats" : {

"executionSuccess" : true,

"nReturned" : 161,

"executionTimeMillis" : 21,

"totalKeysExamined" : 0,

"totalDocsExamined" : 22720,

With Index

"executionStats" : {

"executionSuccess" : true,

"nReturned" : 161,

"executionTimeMillis" : 0,

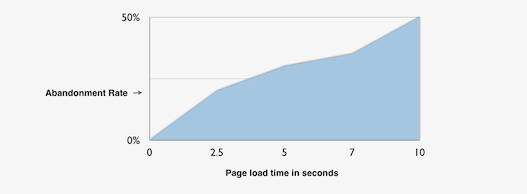
"totalKeysExamined" : 161,

"totalDocsExamined" : 161,

Keywords

Our service offers a specific table where one user can find the trending keyword of a chosen period. When an user select one of the keywords, a query find all the documents containing it. Assuming that users are very likely to read articles that contain trending different keywords every day.

Even if this operation is balanced between write and read, out main objective is to ensure fast response time to our users! If a user has to wait more than 2/3 second for a response is less prone to keep using our application.



Immage: <https://www.relentlesstechnology.com/wp-content/uploads/2012/02/site-speed-visitor-patience1.png>

Statistics for one day

Assuming that 50 articles are written containing 10 significant keyword each

and 200 users are using our service every day and each one search for the top 5 trending keywords we have

+ 50\*10=500 writes

+ 200\*5=1000 reads

Query

db.Article.find({"Keywords.keyword":"prestito"}).explain("executionStats")

Without Index

"executionStats" : {

"executionSuccess" : true,

"nReturned" : 66,

"executionTimeMillis" : 279,

"totalKeysExamined" : 0,

"totalDocsExamined" : 22720

With Index

"executionStats" : {

"executionSuccess" : true,

"nReturned" : 66,

"executionTimeMillis" : 0,

"totalKeysExamined" : 66,

"totalDocsExamined" : 66,