Efficient TDV computation in no SQL environment

LALAOUI Ouassim

Master Research “Information Apprentissage Cognition “

Supervisors :

Nicolas TRAVERS,

Cédric DU MOUZA.

September 9, 2015

1. Introduction

Big data has become a great challenge for many researchers. In fact, a huge amount of data is available on web. This is due to the simplicity to share content on web thought many platforms. This is also due to the development of smartphones that allows peoples to be connected at all time.

Publish/Subscribe systems allow publishers to issue a content that can be an article or an information. This content is defined by key words called items. Several subscribers can receive that content on their smartphone. Each user receives a content defined by items to which he subscribed. A processing efficiency is paramount to ensure rapid delivery.

The *FiND* platform is a Publish/Subscribe system that allows diffusion of new information through RSS feeds. It uses an indexing system that allows an association between users and contents in a reasonable time. Furthermore, the introduction of concepts of partial satisfaction, novelty and diversity of information has led to the computation of Term Discrimination Values (*TDV*) of each item.

TDV is a term weight which depends on its appearance in all publications. The number of publications processed by *FiND* system is of order of millions. This makes TDV computation very expensive in term of time and memory space. This reduces scalability of the system. This work aims at building a solution that consists in, firstly, targeting items whose TDV is likely to change much, and secondly, distributing computing across multiple clusters in a NoSQL environment like in the MongoDB database.

NoSQL databases are used to store large amounts of data. They can store these data on different heterogeneous machines. These data are less structured than SQL data. The Map-Reduce framework allows parallelization of different operations on SQL or NoSQL data. This allows a huge gain processing treatment.

During this work we will try to optimize the calculation of TDVs for terms of a vocabulary from a collection of documents. We will implements several algorithms that will run on a NoSQL database. These algorithms will be distributed over several clusters through Map-Reduce framework.

We will expose different approaches to computing TDV (Section 2). We will study complexities of algorithms (Section 2). We will propose a data structure and algorithms to adapt different approaches of TDV computing to the distributed computing (Section 3). We

will implement our algorithms (Section 4). We will presents results of experimentations (Section 5) .We will conclude at the end our work with further work (Section 6).

2. Related work:

Since TDV is the principal objective of this work, we will focus the related work on different technics to optimize the computation of TDV values. We will compare them in term of complexity for the whole process and also for the computation of a single term. In a second step, we will study how NoSQL databases and the Map-Reduce framework acts in order to explain what have to be done to transform technics from literature to distributed algorithms.

2.1. Term Discrimination Value:

Term discrimination value (TDV) is a concept used in information retrieval in order to measure the relevance of documents for a given query. It indicates influence of a term on similarity between documents. A term with a positive TDV is a good discriminator i.e., the removal of this term from vocabulary make documents more similar. A term with negative TDV is a poor discriminator i.e., the removal of this term from vocabulary make documents less similar. A term with TDV near zero is an indifferent discriminator i.e., the similarity between documents is not modified so after deleting the term.

Let D is a matrix of M rows and N columns. Each row of the matrix represents a document d_i. Each column of the matrix represents the number of occurrence of a term tj in documents. An element d_{ij} of the matrix D is the number of occurrences of the term tj in the document d_i. The set of documents (The M rows) is called collection. The set of terms (The N columns) is called vocabulary.

We will outline in following three approaches to calculate the TDV. D Matrix is used in all three approaches.

2.1.1. First approach (the naïve and centroid methods)[Willett, 1984]

To calculate the TDV a term *ti* with naive method, we must calculate the density of the collection by summing the similarities of all pairs of documents. We must recalculate the density after *ti* removal of all documents (deleting is not physic; simply do not consider the term *ti*). The TDV the term *ti* is obtained by deducting the density without the term *ti* by density with the term *ti* and all dividing the result by the density with the term *ti*.

Let two rows d_i and d_j of matrix D representing two documents. The similarity between d_i and d_j is computed using cosine function coefficient:

$$\cos(d_j, d_k) = \frac{\sum_{i=1}^{N} d_{ji} d_{ki}}{(\sum_{i=1}^{N} {d_{ji}}^2 \sum_{i=1}^{N} {d_{ki}}^2)^{1/2}} (1)$$

The density of a collection of documents represents the relationship between documents, which we refer by Q. It is calculated using two different formulas.

The first formula is to sum similarities between all pairs of documents:

$$Q = \sum_{i=1}^{M-1} \sum_{j=i+1}^{M} \cos(d_i, d_j) \text{ (2)}$$

Let $G = (G_1, \dots, G_k, \dots, G_N)$ the center of gravity of the collection C, then, for $1 \leq k \leq N$:

$$G_k = \frac{\sum_{k=1}^{M} d_{ik}}{M} \text{ (3)}$$

The second formula is to sum similarities between documents and G:

$$Q = \sum_{i=1}^{M} cos(d_i, G) \text{ (4)}$$

The density of collection after the deletion of t_i in all documents is referred by Q_i. D Matrix is converted so that a document d_k is represented as follows:
$d_k = (d_{k\,1}, \dots, d_{k\,i-1}, d_{k\,i+1}, \dots, d_{kN})$. Q_i is computed in the same manner as Q.

The TDV is the difference between the densities with and without the t_i term respectively,

$$DV_i = (Q_i - Q) \,/\, Q \text{ (5)}$$

The following algorithm is designed to compute TDV with the naïve method:

```
(1) Q = 0;
(2) For i := 1 to M − 1 do
(3)     For j := i + 1 to M do
(4)         Q := Q + cos(d_i, d_j);
(5)For i := 1 to N do
(6)         Q_i = 0
(7)         For j := 1 to M − 1 to
(8)             For k := j + 1 to m do
(9)                 Q_i = Q_i + cos(d_j^i, d_k^i);
(10)            DV_i = (Q_i − Q)/Q;
```

Algorithm 1: Naïve method

The calculation of Q requires a complexity of $2M^2N$. The calculation of each value Q_i requires also a complexity of $2M^2N$. There are N values Q_i calculate. This gives that the complexity of the naïve method algorithm is $2M^2N^2 + 2M^2N$. The complexity of calculation of a single term t_l is $4M^2N$.

The following algorithm computes TDV of vocabulary with the centroid method:

(1) $For\ i := 1\ to\ N\ do$
(2) $c_i = 0$
(3) $For\ j := 1\ to\ M\ do$
(4) $c_i := c_i + d_{ji};$
(5) $c_i = c_i/N$
(6) $Q := 0;$
(7) $For\ i := 1\ to\ M\ do$
(8) $Q := Q + \cos(c, d_i);$
(9) $For\ i := 1\ to\ N\ to$
(10) $Q_i = 0;$
(11) $For\ j := 1\ to\ M\ do$
(12) $Q_i = Q_i + \cos(c^i, d_j^i);$
(13) $DV_i = (Q_i - Q)/Q;$

Algorithm 2: Centroid method

Calculation of centroid requires a complexity of MN, calculation Q or Q_i requires a complexity of $2MN$. The number of Q_i calculated is N. This gives that a complexity of the centroid method algorithm is $2MN^2 + 3MN$. The complexity of computing of TDV of a single term t_l is $5MN$.

2.1.2. Second approach (The clustering method) [Can and Ozkarahan, 1990]

The second approach is to calculate the TDV using Covering Cluster concept, which is a clustering method. Indeed, it is shown that there is a relationship clustering and indexing. Two documents in a same cluster are more likely to respond to the same query; two queries with two terms in the same cluster are more likely to return the same documents.

The Cluster Concept Covering Method "C^{3}M" is a clustering partitioning algorithm, which aims at designating documents which are centers of gravity of clusters and, in a single iteration, associate each document to cluster to which it is closest.

Two conditions must be true to apply C^{3}M to calculate the TDV:

(1) $\sum_{j=1}^{N} d_{ij} > 0;\ 0 \leq j \leq N$ Each document contains at least one term.
(2) $\sum_{i=1}^{M} d_{ij} > 0\ ; 0 \leq i \leq\ M$ Each term is assigned to at least one document.

Once the above conditions are satisfied, the calculation of the TDV is achieved in three steps.

a) Construction of C matrix

Each box ij of this matrix indicates the probability of drawing a document d_j taking all the terms of a document d_i. The matrix C therefore indicates how close a document from another document compared to others.

The first step is to build a C matrix such that each element c_{ij} is given by the following formula:

$$c_{ij} = \sum_{k=1}^{n} s_{ik} * {s'}^{T}{}_{kj} \text{ (6)}$$

where:

(1) $s_{ik} = \frac{d_{ik}}{\sum_{h=1}^{M} d_{ih}}$ (7) (The probability to select t_k from d_i)
(2) ${s'}^{T}{}_{kj} = \frac{d_{jk}}{\sum_{h=1}^{M} d_{hk}}$ (8) (The probability to select d_j from t_k)

Formula (5) can therefore be written as follows:

$$c_{ij} = \sum_{k=1}^{N} \frac{d_{ik}}{\sum_{h=1}^{n} d_{ih}} * \frac{d_{jk}}{\sum_{h=1}^{m} d_{hk}} \text{ (9)}$$

In arise:

(1) $\alpha_i = \frac{1}{\sum_{h=1}^{N} d_{ih}}$ (10)
(2) $\beta_i = \frac{1}{\sum_{h=1}^{M} d_{hk}}$ (11)

Formula (8) becomes:

$$c_{ij} = \alpha_i * \sum_{k=1}^{N} d_{ik} * \beta_k * d_{jk} \text{(12)}$$

An element c_{ii} of matrix C is called decoupling coefficient of d_i document. It shows to which point document d_i is different from others i.e., more c_{ii} is great, more the document d_i does not look at other documents of the collection. This makes elements $c_{ii}, 1 < i < M$ good indicators of the number of clusters to choose when partitioning a collection. An element c_{ii} is appointed by δ_i (δ_i=c_{ii}).

The value $\varphi_i = 1 - \delta_i$ is called coupling coefficient of d_i document. It indicates how much d_i document resemble other documents.

b) Computing number of clusters:

The formula for calculating the number of needed clusters after the partitioning of the collection is as follows:

$$n_c = \sum_{i=1}^{M} \delta_i \text{(13)}$$

This formula gives an approximation of the number of clusters because as the amount $\sum_{i=1}^{M} \delta_i$increases, the density of the collection decrease thus the number of required clusters increases.

Note:

After calculating the number of clusters. An algorithm is executed to designate the cluster centroids and then another is executed to assign each document to a centroid. We will not discuss these algorithms in this article because it does not help to compute TDV.

c) Computing TDV

The TDV value of a term t_l is therefore the difference between the number of clusters before and after the transformation of D matrix such that each document d_k is represented by $d_k = (d_{k\,1}, \dots, d_{k\,l-1}, d_{k\,l+1}, \dots, d_{kn})$ (removal of t_l terms from all documents).

The number of clusters is inversely proportional to the density of a collection. Indeed more a collection is dense, less the number of clusters is great. So instead of calculating TDV value of a term t_l by subtracting densities without and with the term t_l . This TDV value is approximated by subtracting numbers of clusters of the collection with and without the term t_l .

The TDV is thus calculated as follows:

$$TDV_l = n_c - n_{cl} \quad (14)$$

where n_{cl} is the number of clusters after removing the term t_l from the vocabulary and the partitioning of the collection.

After replacing n_c and n_{cl} by their formulas we obtain:

$$TDV_l = \sum_{i=1}^{f_l} [\delta_i - \alpha_i^l (\frac{\delta_i}{\alpha_i} * {d_{il}}^2 * \beta_l)] \quad (15)$$

Where:

(1) $\alpha_i^l = (\sum_{h=1}^{n} d_{ih} - d_{il})^{-1}$: The inverse of sum of elements of row i less d_{il}.
(2) f_l : The number of documents such as $d_{il} \neq 0$.

The following algorithm computes TDV of vocabulary terms with the clustering method:

---

(1) $For\ i := 1\ to\ M\ do$
(2) $\alpha_i = 0$;
(3) $For\ j := 1\ to\ N\ do$
(4) $\alpha_i = \alpha_i + d_{ij}$;
(5) $\alpha_i = 1/\alpha_i$;
(6) $For\ l := 1\ to\ N\ do$
(7) $\beta_l = 0$;
(8) $For\ i := 1\ to\ M\ do$
(9) $\beta_l = \beta_l + d_{il}$;
(10) $\beta_l = 1/\beta_l$;
(11) $For\ i := 1\ to\ M\ do$
(12) $For\ j := 1\ to\ N\ do$

(13) $\delta_i = \delta_i + \beta_j * {d_{ij}}^2;$

(14) $\delta_i = \delta_i * \alpha_i ;$

(15) $For\ l := 1\ to\ N\ do$

(16) $TDV_l = 0;$

(17) $For\ i := 1\ to\ M\ do$

(18) $IF\ d_{il} > 0$

(19) $\alpha_i^l = 1/(1/\alpha_i - d_{il}) ;$

(20) $TDV_l = TDV_l + [\delta_i - \alpha_i^l * (\frac{\delta_i}{\alpha_i} - {d_{il}}^2 * \beta_l)]$

Algorithm 3: Clustering method

The computing of each α_i and each δ_i requires a complexity of N. The computing of eachβ_l requires a complexity of M. Therefore, the computing of all α_i, $\beta_l$and δ_i requires a complexity of $3MN$. After computing all α_i, $\beta_l$and δ_i, calculating the TDV value of each term t_l requires a complexity of MN. This gives that a complexity of the clustering method algorithm is $MN^2 + 3MN$. The complexity of algorithm of calculating of a single term t_l is $4MN$.

2.1.3. Third approach (Dot Product method) [El-Hamdouchi and Willett, 1988]:

This method involves defining and calculating a centroid of the collection. It involves also defining a matrix "dif". The TDV values are functions of components of centroid and values of the matrix "dif". Components of the centroid and elements of matrix will be defined in following.

Let A be a vector representing a linear combination of D_j documents, $1 \leq j \leq x_a$. These documents belong to a given cluster. A is designed as the centroid of the cluster. It is calculated as follows:

$$A = \sum_{j=1}^{x_a} w_j * D_j \text{(16)}$$

where w_j is the weight of the j^{th} document in the cluster.

Let B a vector and its components are defined by the similarities between documents that belong to a second cluster (B is the centroid of the second cluster).

The vector B is calculated as follows:

$$B = \sum_{j=1}^{x_b} w_j * D_j \text{ (17)}$$

The dot product between A et B is given by:

$$DOTPRODAB = \sum_{j=1}^{x_a} w_j d_j * \sum_{k=1}^{x_b} w_k d_k \text{(18)}$$

Through a series of processing, authors will extract a formula for calculating a TDV of term of vocabulary. The goal of transformations that will be performed on the formula (18) is to

extract formulas of calculation densities of a cluster of documents (collection documents) with all terms of vocabulary and without one of terms. These densities are necessary to calculate TDV values of any term of vocabulary.

The formula (18) may be rewritten as:

$$DOTPRODAB = \sum_{j=1}^{x_a} \sum_{k=1}^{x_b} w_j * w_k * DOTPRODJK \text{ (19)}$$

where $DOTPRODJK$ is the dot product between vectors (documents) d_j and d_k.

The weight of document d_j is defined as:

$$w_j = SUMSQJ^{1/2} \text{ (20)}$$

where $SUMSQJ = \sum_{i=1}^{M} {d_{ji}}^2$ (21)

If w_k is defined with the same manner the formula (19) becomes:

$$DOTPRODAB = \sum_{j=1}^{x_a} \sum_{k=1}^{x_b} COSJK \text{ (22)}$$

Where $COSJK$ difines the siminarity between documents d_j and d_k.

Let the cluster A and the cluster B are the same cluster and is called cluster C (A=B=C) and composed by M documents.

The formula for calculating vector C is the following:

$$C = \sum_{j=1}^{M} w_j * D_j \text{ (23)}$$

We obtain that the dot product between C and itself is given by the following formula:

$$DOTPRODCC = \sum_{j=1}^{M} \sum_{k=1}^{M} w_j * w_k * DOTPRODJK \text{ (24)}$$

The dot product between C and itself is equivalent to SUMSQC ($C * C$) the sum of squares of components of the centroid C because the sum of multiplication vectors by weight gives the centroid.

The calculation of the TDV of a t_i term involves the calculation of $\frac{M(M-1)}{2}$ similarities between documents d_j and d_k where $i < k$ and therefore formula (20) becomes as follows:

$$SUMSQC = 2 * \sum_{\substack{j,k=1 \\ (j<k)}}^{M} w_j * w_k * DOTPRODJK + \sum_{j=1}^{N} {w_j}^2 SUMSQJ \text{ (25)}$$

Let Q is the density of the cluster C. The density is calculated as follows:

$$Q = \sum_{\substack{j,k=1 \\ j<>k}}^{M} COSJK \text{ (26)}$$

From the formula (25) we obtain the two following formulas:

$$2Q = SUMSQC - \sum_{j=1}^{N} {w_j}^2 * SUMSQJ \text{ (27)}$$

and:

$$2Q_i = SUMSQCI - \sum_{j=1}^{N} {wi_j}^2 * SUMQJI \text{ (28)}$$

where $SUMSQCI$ and $SUMQJI$ are respectively the sum of square of components of the centroid and the sum of square of components J^{th} document when the term t_i is deleted from the vocabulary. And where wi_j is the weight of document d_j after the same deletion.

The TDV of a term t_i is then defined as $(Q_i - Q)/Q$. But the division by Q, which is the positive constant, does not change the order of TDVs of vocabulary's terms. Thus, division by Q may be neglected. In addition, the multiplication of the right side of the expression by 2 is not going to change the order of TDVs of vocabulary's terms. The TDV of a term t_i can be expressed as follow:

$$DV_i = SUMSQCI - SUMSQC - \sum_{j=1}^{N} {wi_j}^2 * SUMSQJI + \sum_{j=1}^{N} {w_j}^2 * SUMSQJ \text{ (29)}$$

With $w_j = 1/SUMSQJ^{1/2}$ and $wi_j = 1/SUMSQJI^{1/2}$ we obtain:

$$DV_i = SUMSQCI - SUMSQC \text{ (30)}$$

Let c_k ant ci_k les k^{ths} components of C and CI, respectively. Where CI is the centroid of the collection after the term t_i from the vocabulary. Then, by substituting for $SUMSQCI$ and $SUMSQC$ we obtain:

$$\begin{aligned} DV_i &= \sum_{\substack{k=1 \\ k<>i}}^{N} {ci_k}^2 - \sum_{k=1}^{N} {c_k}^2 \\ &= \sum_{\substack{k=1 \\ k<>i}}^{N} ({ci_k}^2 - {c_k}^2) - {c_k}^2 \\ &= \sum_{\substack{k=1 \\ k<>i}}^{N} [(ci_k + c_k) * (ci_k - c_k)] - {c_i}^2 \text{ (31)} \end{aligned}$$

Let KI be a set of documents containing the terms t_i and t_k, KNI set of documents containing t_k but not t_i. For $k <> i$, c_k may be obtained by following formula:

$$c_k = \sum_{D_j \in KI} w_j d_{jk} + \sum_{D_j \in KNI} w_j d_{jk} \text{ (32)}$$

Similarly ci_k, $k <> i$, can be expressed by following formula:

$$ci_k = \sum_{D_j \in KI} wi_j d_{jk} + \sum_{D_j \in KNI} wi_j d_{jk} \text{ (33)}$$

For documents d_j that belong to KNI, since t_i do not appears in this documents, the following equation is checked:

$$wi_j d_{jk} = w_j d_{jk} \quad (34)$$

Thus the formula (24) can by rewritten as follows:

$$DV_i = \sum_{\substack{k=1 \\ k<>i}}^{N} \left[DIF_k^i * \left(DIF_k^i + 2 * c_k\right)\right] - {c_i}^2 \quad (35)$$

Where for each term t_i each term t_k DIF_k is computed as follows:

$$DIF_k^i = \sum_{D_j \in KI} \left(wi_j - w_j\right) * d_{jk} \quad (36)$$

All of the elements DIF_k^i forms the matrix DIF.

The following algorithm computes vocabulary terms with Dot product method:

(1) $For\ i \coloneqq 1\ To\ N\ \ do$
(2) $\quad c_i \coloneqq 0;$
(3) $\quad FOR\ j \coloneqq 1\ To\ M\ Do$
(4) $\quad\quad c_i \coloneqq c_i + w_j d_{ji};$
(5) $For\ i \coloneqq 1\ To\ N\ Do$
(6) $\quad For\ k \coloneqq 1\ To\ N\ Do$
(7) $\quad\quad DIF_k^i = 0;$
(8) $\quad For\ j \coloneqq 1\ To\ M\ Do$
(9) $\quad\quad If\ d_{ji} > 0\ Then$
(10) $\quad\quad\quad For\ k \coloneqq 1\ To\ N\ Do$
(11) $\quad\quad\quad\quad If\ d_{jk} > 0\ Then$
(12) $\quad\quad\quad\quad\quad DIF_k^i \coloneqq DIF_k^i + \left[wi_j - w_j\right] * d_{jk};$
(13) $\quad DV_i \coloneqq 0;$
(14) $\quad FOR\ k \coloneqq 1\ To\ N\ Do$
(15) $\quad\quad DV_i = DV_i + DIF_k^i * \left(DIF_k^i + 2 * c_k\right);$
(16) $\quad DV_i \coloneqq DV_i - {c_i}^2$

Algorithm 4: Dot product method

This algorithm, which calculates the centroid c, involves computing all of its N components. Calculate each component involves calculate weight of each document. Calculate each component involves also browsing all of M documents. Therefore the computation of the centroid c requires a complexity $+M(N-1)$. After computing the centroid c, the computation of TDV of each term t_i requires to calculate the sum of the N values of DIF_k, $1 \leq k \leq N$. This requires a complexity of $+M(N-1)+2N$. Therefore the complexity of Dot product method algorithm is $MN^2 + MN(N-1) + 2N^2 + MN + M(N-1)$. The computation of TDV of single term t_i is $2MN + 2M(N-1) + 2N$.

2.1.4. Complexities:

Assuming that each document is a vector of size N and that there is M documents in the collection, the complexity of algorithms will be given by a function of N and M.

The following table is the synthesis complexities of algorithms presented below for calculation of TDV of single and all terms.

	Naïve method	**Centroid method**	**Clustering method**	**Dot product method**
Single term	$4M^2N$	$5MN$	$4MN$	$2MN + 2M(N-1) + 2N$
Algorithm	$2M^2N(N+1)$	$MN(3+2N)$	$MN^2 + 3MN$	$MN^2 + MN(N-1)$ $+2N^2 + MN + M(N-1)$

Table 1: Complexities of the different methods

Nested loops that contain these algorithms make their huge complexities. In the naïve method, calculating the density is very expensive, In addition, density must be calculated N + 1 times (once for global density and N ignoring a different terms). For the second method the density should be calculated N + 1 times but its calculation is less complex since it requires only to sum similarities between the centroid and documents and not between every pairs of the document. This causes the centroid method is better than the naive method. According to the clustering method, the calculation of components of vectors alpha, beta, and delta have complexities MN. Once these calculations are made, the complexity of calculation of TDV is MN^2. It makes this method better than the centroid one. For the dot Product method, which most expensive costs is the calculation the "dif" matrix. It costs $N^2 + MN^2$, so that this method is better than the centroid method and worse than the clustering method.

Clustering et dot Product methods have therefore a better complexity but the clustering method gives approximated TDVs and dot Product that consume too much memory space.

Note: In practice, the document is represented by a vector of size n. Terms whose occurrences are equal to zero are not represented.

2.2. **No-SQL databases and the Map-Reduce framework**

NoSQL databases are designed to save large amounts of data that data. The backup is done on different servers enabling parallel processing. We will apply implemented methods on large databases and we want to optimize the time processing. This is why we choose NoSQL.

2.2.1. **Map-Reduce**

Map-Reduce is a programming model proposed by google body. A developer using this model must specify a map function is a reduce function. A map function contains an *emit* with a key and a value. Values with same key are grouped and threated by a reduce

function. Map and reduce functions are parallelized and run on multiple clusters simultaneously.

2.2.2. **NoSQL**

NoSQL databases are classified in four categories:

- Key-value based databases:

A table in the database contains a list of keys. A list of values is associated to each key. A data searches can only be performed against keys. Examples of key-value based databases are SimpleDB, Dynamo, Voldemort, Redis, BerkleyDB and riak.

- Document based databases:

Designed to store documents in XML, JSON or BSON format. Columns contain semi-structured data. A data searches can be performed against both keys and values. Examples of document based databases are Couched, MongoDB and RavanDB.

MongoDB is a database management system for storing documents in the BSON format. It allows saving documents without a predefined schema. A document consists of objects. These objects can be simples or complex. Simple objects are composed of attributes and values. Complex objects are composed of objects and object lists. It is this DBMS that we will use for our job.

- Column based databases:

They are near database relational databases. Each object has a different number of columns. They are provided for cases where each record has millions of columns. It is not necessary to save “null” values in these tables; however, the table updates costs are very expensive. Examples of column-based databases are Cassandra, HBase, BigTable, HyperTable, SimpleDB, DynamoDB.

- Graph based databases:

They are oriented objects databases. They are used to represent connections between objects. Each node can point to other nodes that will be called neighbors. This optimizes local research and allows finding the related object without jointure. Sometimes, to find links between objects, it is necessary to cross the entire graph which is very expensive. Examples of graph-based databases are Neo4j, InfoGrid, Sone GrapgDB, InfinitGraph, HyperGraphDB.

3. Incremental computation of TDVs

Previous technics, which have been presented in the related work, were dedicated to compute a whole static corpus of documents. But if this corpus evolves over time this computation becomes very expensive and a huge waste of time. Thus we propose in this chapter to focus on an incremental computation of TDVs based on the previous methods. This means to extract minimal information for each method necessary for the computation of a single incoming document, but also to update the minimum amount of structures in the repository. We will show for each method the needed information and provide the complexity of each of them.

Let C be a collection of M documents and N terms. The changes that can be made on this collection are adding and deleting a document.

3.1. Naïve method

When adding or deleting a document, applying this method to recalculate the TDV a term t_i is to calculate the sum of similarities with and without the term ti between the document added or deleted and other documents of the collection. Then, add or deduct obtained values from the old values of the density with and without the density term t_i. TDV of term t_i is equal to the subtraction of the new value density without the term t_i by the new value of the density with the term t_i and the division of the result of subtraction by the new value of the density with the term t_i.

The TDV of a term is calculated using the formula (1). Density Q is calculated using formula (2). Adding or deleting a document implies changes in Q and Q_i values.

To calculate the new density Q^{new} after adding document d_{M+1} we must add to Q^{old} similarities between the document d_{M+1} and documents of the collection C. Q^{new} is obtained through the following formula:

$$Q^{new} = Q^{old} + \sum_{i=1}^{M} \cos(d_i, d_{M+1}) \quad (37)$$

The following algorithm computes TDV of vocabulary terms with naïve method after insertion of a document d_{M+1}:

---

(1) $Q^{new} = Q^{old}$;
(2) $For\ i := 1\ to\ M\ do$
(3) $\quad Q^{new} := Q^{new} + \cos(d_i, d_{M+1})$;
(4)$For\ i := 1\ to\ N\ do$
(5) $\quad {Q_i}^{new} = {Q_i}^{old}$;
(6) $\quad For\ j := 1\ to\ M\ to$
(7) $\quad\quad {Q_i}^{new} = {Q_i}^{new} + \cos(d_j^i, d_{M+1}^i)$;
(8) $\quad DV_i = ({Q_i}^{new} - Q^{new})/Q^{new}$;

Algorithm 5: Insertion in naïve method

The new density Q^{new} after deleting a document d_h, $1 \leq h \leq M$, is calculated by subtracting to Q^{old} similarities between the document d_h and other documents of the collection C. Q^{new} is therefore obtained by the following formula:

$$Q^{new} = Q^{old} - \sum_{\substack{i=1 \\ i<>h}}^{M} \cos(d_i, d_h) \quad (38)$$

The algorithm to perform the calculation of the TDV after removing of a document $d_h$with the naïve method is the same as the insertion in naïve method algorithm. Simply replace d_{M+1} by $d_h$and '−' by '+' in line (4) and replace d_{M+1}^i by d_h^i and '−' by '+'.

The calculation of Q^{new} requires a complexity of 2MN. The calculation of all of the N values ${Q_i}^{new}$ requires a complexity of $2MN^2$. This gives that the complexity of algorithm (5) is $2MN^2 + 2MN$. The complexity of calculation of a single term t_i is $4MN$.

3.2. Centroid method

When adding or deleting a document, the centroid is recalculated. Densities with and without terms are recalculated by summing similarities between the document and the new centroid with and without terms.

As for naïve method, TDV of a term is calculated using the formula (1). Bat the density Q is calculated using the formula (4). Adding or deleting a document implies changes of values c, Q and Q_i.

By adding a document d_{M+1}, the new centroid c^{new}may be calculated using based on the previous. The calculation of ${c_i}^{new}$the ith component of the new centroid is done using the following formula:

$${c_i}^{new} = \frac{N({c_i}^{old}) + d_{M+1}}{N+1} \quad (39)$$

The new density Q^{new} after adding the document d_{M+1} is obtained by the following formula:

$$Q^{new} = \sum_{i=1}^{M+1} \cos(d_i, c^{new}) \quad (40)$$

The following algorithm computes TDV of vocabulary terms with centroid method after insertion of a document d_{M+1} :

---

(1) $Q^{new} = 0$;
(2) $For\ i := 1\ to\ N\ do$
(3) $\quad {c_i}^{new} := (N * {c_i}^{old} + d_{M+1\,i})/(\mathrm{M} + 1)$;
(4) $For\ i := 1\ to\ (M+1)\ do$
(5) $\quad Q^{new} := Q^{new} + \cos(d_i, c^{new})$;
(6) $For\ i := 1\ to\ N\ do$

(7) $Q_i{}^{new} = 0;$
(8) $For\ j := 1\ to\ (M+1)\ to$
(9) $\quad Q_i{}^{new} = Q_i{}^{new} + \cos\left(d_j^i, c^{i\,new}\right);$
(10) $DV_i = (Q_i{}^{new} - Q^{new})/Q^{new};$

Algorithm 6: Insertion in centroid method

By removing a document d_h, $1 \leq h \leq N$, from the collection C, the i[th] component of the new centroid is calculated based on the i[th] component of the previous centroid. $c_i{}^{new}$ is obtained by the following formula:

$$c_i{}^{new} = \frac{N\left(c(i)^{old}\right) - d_h}{M-1} \quad (41)$$

The density Q^{new} after the deleting of the document d_h is done by the following formula:

$$Q^{new} = \sum_{i=1}^{M-1} \cos(d_i, c^{new}) \quad (42)$$

To compute TDV of vocabulary terms after removing of document d_h we mast execute insertion in centroid method algorithm with replacing formula (39) by formula (41) in line (3) and $M+1$ by $M-1$in lines (4) and (8).

Calculation of the new centroid requires a complexity of N. Calculation of $Q_i{}^{new}$or Q^{new}requires a complexity of $2\mathrm{MN}$. The number of $\mathrm{Q_i}$ calculated is N. This gives that a complexity of algorithm (6) is $2\mathrm{MN}^2 + 2\mathrm{MN} + \mathrm{N}$. The complexity of algorithm of calculation of a single term $\mathrm{t_l}$ is $4\mathrm{MN} + \mathrm{N}$.

3.3. Clustering method

When adding or deleting a document, values of components of vectors α, β and δ are changed before recalculating values of TDVs.

For the clustering method, the TDV of a term is calculated using the formula (15). The addition or the removal of a document involves charges of values δ_i, $1 \leq i \leq M$.

Adding a document d_{M+1} involves adding a value α_{M+1}. The values δ_i, $1 \leq i \leq M$, are change. Changes of values of δ_i are due to changes of values of β_j, $1 \leq j \leq N$. The new values $\beta_j{}^{new}$, $1 \leq j \leq N$ are calculated with the following formula :

$$\beta_j{}^{new} = 1/(1/\beta_j^{old} + d_{M+1\,j}) \quad (43)$$

The new values $\delta_i{}^{new}$,$1 \leq i \leq M+1$, are calculated with the following formula:

$$\delta_i{}^{new} = \alpha_i * \sum_{j=1}^{N} \beta_j{}^{new} * d_{ij}{}^2 \quad (44)$$

The following algorithm computes TDV of vocabulary terms with clustering method after insertion of a document d_{M+1} :

(1) $\alpha_{M+1} = 0;$
(2) $For\ j := 1\ to\ N\ do$
(3) $\quad \alpha_{M+1} = \alpha_{M+1} + d_{M+1\ j};$
(4) $\alpha_{M+1} = 1/\alpha_{M+1}\ ;$
(5) $For\ j := 1\ to\ N\ do$
(6) $\quad {\beta_j}^{new} = 1/(1/\beta_j^{old} + d_{M+1\ j});$
(7) $For\ i := 1\ to\ M+1\ do$
(8) $\quad For\ j := 1\ to\ N\ do$
(9) $\quad\quad {\delta_i}^{new} = {\delta_i}^{new} + {\beta_j}^{new} * {d_{ij}}^2;$
(10) $\quad {\delta_i}^{new} = {\delta_i}^{new} * \alpha_i\ ;$
(11) $For\ l := 1\ to\ N\ do$
(12) $\quad TDV_l = 0;$
(13) $\quad For\ i := 1\ to\ M+1\ do$
(14) $\quad\quad IF\ d_{il} > 0$
(15) $\quad\quad\quad \alpha_i^l = 1/(1/\alpha_i - d_{il})\ ;$
(16) $\quad\quad\quad TDV_l = TDV_l + [{\delta_i}^{new} - \alpha_i^l * ({\delta_i}^{new}/\alpha_i - {d_{il}}^2 * {\beta_l}^{new})];$

Algorithm 7: Insertion in clustering method

With the removal of a document d_h, $1 \le h \le M$ from the collection C. The values β_j $1 \le j \le N$ are change. This involves changing of values δ_i $1 \le i \le M$. The new values ${\beta_j}^{new}$, $1 \le j \le N$, are calculated using the following formula:

$${\beta_j}^{new} = 1/(1/{\beta_j}^{old} - d_{M+1\ j}) \quad (45)$$

The new values ${\delta_i}^{new}$ are calculated with the formula (44).

After removing a document d_h, for computing TDV of vocabulary terms with clustering method we must execute insertion in clustering method with deleting lines (1)-(4), replacing formula (42) by formula (44) in line (6) and replacing $M+1$ by $M-1$ in lines (7) and (13).

The computing of α_{M+1} requires a complexity of N. The computing of all ${\beta_j}^{new}$,$1 \le j \le N$, requires a complexity of M. The computing of all ${\delta_i}^{new}$ requires a complexity of MN. After computing all α_i, $\beta_l$and δ_i, the computing of TDV of each term t_l require a complexity of MN. This gives that a complexity of algorithm (7) is $MN^2 + MN + 2N$. The complexity of algorithm of calculating of a single term t_l is $2MN + 2N$.

3.4. Dot product method

When adding or deleting a document, values of components of centroid and elements of the collection “dif” are modified. New values must be recalculated before reapplying formula of calculation of TDVs with dot Product method.

The TDV of a term is calculated using the formula (35). After adding or deleting a document, values c_i and DIF_k^i, $1 \le k \le N$, are change.

After adding a document d_{M+1},the components ${c_i}^{new}$of the new centroid c^{new} are calculated with the following formula:

$$ {c_i}^{new} = {c_i}^{old} + w_{M+1} d_{M+1\,i} \quad (46) $$

The new values $DIF_k^{i\,new}$, $1 \le k \le N$, are calculated using the following formula:

$$ DIF_k^{i\,new} \coloneqq DIF_k^{i\,old} + (wi_{M+1} - w_{M+1}) * d_{M+1\,k} \quad (47) $$

The following algorithm computes TDV of vocabulary terms with Dot product method after insertion of a document $d_{M+1\,i}$:

(1) $For\ i \coloneqq 1\ To\ N\ \ do$
(2) $\quad {c_i}^{new} \coloneqq {c_i}^{old} + w_{M+1} d_{M+1\,i};$
(3) $For\ i \coloneqq 1\ To\ N\ Do$
(4) $\quad If\ d_{hi} > 0\ Then$
(5) $\quad\quad For\ k \coloneqq 1\ To\ N\ Do$
(7) $\quad\quad\quad If\ d_{M+1\,k} > 0\ Then$
(8) $\quad\quad\quad\quad DIF_k^{i\,new} \coloneqq DIF_k^{i\,old} + [wi_{M+1} - w_{M+1}] * d_{M+1\,k};$
(9) $\quad DV_i \coloneqq 0;$
(10) $\quad For\ k \coloneqq 0\ To\ N\ Do$
(11) $\quad\quad DV_i = DV_i + DIF_k^{i\,new} * \left(DIF_k^{i\,new} + 2 * c_k\right);$
(12) $\quad DV_i \coloneqq DV_i - {c_i}^2\ ;$

Algorithm 8 : Insertion in Dot product method

After deleting of a document d_h , $1 \le h \le M$, from the collection C, the new values ${c_i}^{new}$are calculated with the following formula:

$$ {c_i}^{new} = {c_i}^{old} - w_{wk} d_{ki} \quad (48) $$

The new values $DIF_k^{i\,new}$,$1 \le k \le N$, are calculated after deleting of the document d_h , $1 \le h \le M$, with the following formula:

$$ DIF_k^{i\,new} \coloneqq DIF_k^{i\,old} - (wi_h - w_h) * d_{M+1\,k} \quad (49) $$

The algorithm to execute for calculation of TDV after deletion of term is same to the algorithm for calculation of TDV after insertion of term. But we must replace formula (46) by formula (48) in line (2) and formula (47) by formula (49) in line (8).

Compute the new centroid c^{new} involves computing all of its N components. This requires a complexity. After computing the new centroid C, the computation of TDV of each term t_i requires to calculate the sum of the N values of $DIF_k^{i\,new}$, $1 \le k \le N$. This requires a complexity of $2N$. Therefore the complexity of algorithm (8) is $2N^2 + N$. The computation of TDV of single term t_i is $3N$.

3.5. Complexity:

The following table is the synthesis complexities of algorithms 5-8 for calculation of TDV of single and all terms.

	Naïve method	**Centroid method**	**Clustering method**	**Dot product method**
Algorithm	$2MN^2 + 2MN$	$2\mathrm{MN}^2 + 2\mathrm{MN} + \mathrm{N}$	$2\mathrm{MN}^2 + 2\mathrm{MN} + \mathrm{N}$	$2N^2 + N$

Table 2: Complexities of incremental methods.

For incremental methods the complexity of the naive method is better than the complexity of the centroid method. This is due to the fact that in the centroid method, it is not only necessary to recalculate the centroid but it is also necessary to recalculate all the similarities between documents and the new centroid. Naive and clustering methods have the same complexity despite their differences. The dot Product method has a very good and the best complexity because it depends only in N and N is a small value compared to M.

4. Integration in a NoSQL environment

Since the calculation of TDVs is very computational, we need to distribute it in a NoSQL environment in order to scale it up. To achieve this, we need to model each needed structure in the different methods for the incremental algorithms. This chapter will focus on the data structures for documents which are used in the MongoDB database and which types of files will be loaded to compute the TDVs. Then, we will develop the Map-Reduce algorithms we have implemented to achieve this.

The four methods are run on the same collection. However, they will give different intermediate results. That is why we will associate to each method different files to save these intermediate results.

4.1. Data structure

Data are saved in the JSon format on MongoDB. Two choices are possible to us, either to save a collection of documents or to save a collection of terms.

The updates, which have to be made on the collection, are additions and deletions of documents. Save a collection of terms implies to send for each insertion or deletion to several requests to multiple clusters to change the index. This is why we choose to make a backup of a collection of documents. This is to facilitate updates.

A document in collection is defined by an identifier $docID$. It contains its weight $docWeight$ which is the sum of squares of weights of terms that compose it. The document contains also a list of terms that compose it. Each term is defined by its identifier $termID$ and its weight $TermWeight$. The weight of a term in a document may represent the number of its occurrences in this document. It may indicate the presence of this term in the document. It can also represent this TDV. It can finally represent the standardized TDV i.e. TDV of term divided by the sum of TDVs of all terms.

The following figure is a document in the collection.

```
{
    docID : "1",
    docWeight:"3",
    terms :[{termID:"0",termWeight:"1"},
    {termID:"1",termWeight:"1"},
    {TermID:"5",termWeight:"1"}
    ]
}
```

Figure 1: Structure of a document of the collection

4.2. Files

These files will save the results obtained when computing TDV of terms of given collection. These backups will allow recalculating values of TDV changed after changing the collection

by minimizing the number of calculations to be redone. The changes that can be brought to a collection are adding and deleting document.

These files will be initialized when calculating TDV of all vocabulary terms taking into account all documents in the collection. Adding and deleting document involve a change of values contained in these files. Update of these files is necessary every time the collection is changed.

A set of files is associated with each method:

4.2.1. Naïve method

Two files are associated with this method:

- File Q_naive: Used to save the density Q calculated with the naïve method.
- File Qi_naive: Contains densities Q_i, $1 < i < N$, calculated with the naive method.

4.2.2. Centroid method

The files that are associated with the centroid method are:

- File centroid_centroid: Each value stored in this file represents the sum of weights of a term t_i. These values are used to calculate components of the centroid of the collection by dividing the values found in the file on the number of documents in the collection.
- File Q_centroid: Used to store density Q calculated with the centroid method.
- File Qi_centroid : Used to store densities Q_i, $1 < i < N$ calculated with the centroid method.

4.2.3. Clustering method

The following files are associated with the clustering method:

- File alpha_clustering: This file contains the sum of weights of each document. Inverse of each value represents a value α_i, $1 < i < M$.
- File beta_clustering: Contains values that associated with the vocabulary. Each value represents the sum of weights of a term t_i in documents that contain it. Inverse of values contained in the file represent β_j, $1 < j < N$.
- File delta_clustering: Contains values δ_i, $1 < i < M$. δ_i being the result of multiplying α_i by the sum of multiplication of squares of weighs of terms of document d_i by β_j associated to terms.

4.2.4. Method dotProd:

Two files are associated with dot Product method:

- File centroid_dotProd: This file contains values associated with terms of vocabulary. Each value is associated to a term t_j. It represents the sum of multiplication of the weights of a term in documents that contain it by the weight of documents. Thus, these values represent the components of centroid of dot Product method
- File difIK_dotProd: Values DIF_k^i $1 < i < N$, $1 < k < N$, are saved in this file.

4.3. Implementation:

Each one of the four methods is implemented in Map-Reduce to calculate TDVs of vocabulary terms. These methods are implemented so as to be tested on a collection previously defined (Figure 1). Each one of the four methods is implemented in two ways: the complete calculation and incremental calculation. The complete calculation allows calculating TDVs of all terms after seizure of all documents of collection, in other words, for the complete calculation methods defined in section 2 are applied to the collection of documents. The incremental calculation is used to calculate values of TDVs changed after adding or deleting a document. For the incremental calculation of TDVs methods defined in Section 3 are applied to the collection.

4.3.1. Centroid method:

The centroid method is implemented as follow for the **complete** and incremental calculation.

Complete calculation:

- Computing vector c:

The computing is done using the map-reduce function of the document collection to get all values of components the centroid.

The map function is applied to the terms of a document. For each term of a document sending an *emit* with *TermID* as a key and *termWeignt* as value. This *emit* allows grouping for each term its weights in documents.

$Function\ map()$

$For(term\ in\ this.terms)$

$emit(this.terms[term].termID,$

$\{termId: this.terms[term].termID, weight: this.terms[term].termWeight\})$

Algorithm 9: Map function for computing vector c

After calling the map function, a call is made to a reduce function that calculate amounts of emitted values with same keys. The components of vector c are obtained.

$Function\ reduce(key, value)$

$var\ sum = 0$

$For(var\ i\ in\ values)$

$sum = sum + values[i].weight$

$return\ \{termId\ :\ key, weight: sum\}$

Algorithm 10: Reduce function for computing vector c

- Calculating Q:

The function is called. It supports setting the centroid of the collection. For each document, it calculates the similarity between the document and the centroid and makes an emit with null as key and value of similarity as value. This *emit* allows to have similarities between documents and the centroid.

$Fonction\ map()$

$var\ centoid = [chaineCentroid]$

$var\ centroidWeight$

$var\ sim = 0$

$for(var\ term\ in\ this.terms)$

$sim += this.terms[term].termWeight * centroid[this.terms[term].termID]$

$sim = sim/\ (this.DocWeight * centroidWeight)$

$emit(null, sim)$

Algorithm 11: Map function for computing density Q

The reduce function called after map function calculate the sum of emitted values. It gives the density Q.

$Function\ reduce(key, value)$

$var\ sum = 0$

$for(var\ i\ in\ values)$

$sum = sum + values[i].Similarite$

$return\ sum$

Algorithm 12: Reduce function for computing density Q

- Calculating Q_i:

The map function is called. For each term, the similarity between the document and the centroid is calculated after removing the term. The identifier of the term is emitted as the key and the similarity as values. This emit allows to have for each term similarities between documents and the centroid by ignoring this term.

$Fonction\ map()$

$var\ centoid = [chaineCentroid]$

$var\ centroidWeight$

$var\ sim$

$var\ p1$

$var\ p2 = centroidWeight$

$var\ p3$

$for(var\ term\ in\ this.terms)$

$sim = 0$

$p3 = p2$

$for(var\ term1\ in\ this.terms)$

$if\ term\ != term1$

$sim = sim + this.terms[term1].termweight * centroid[this.terms[term1].termID]$

$else$

$p1 = this.docWeight - this.terms[term1].termweight * this.terms[term1].termWeight$

$p3 = p2 - this.terms[term1].termweight * this.terms[term1].termWeight$

$sim = sim/\ (p1 * p3)$

$emit(this.terms[term].termID, \{termId: this.terms[term].termID, Similarite: sim\})$

Algorithm 13: Map function for computing densities Q_i

After calling the map function. The reduce function is called to calculate the amounts of values emitted with same cases. We obtain values of Q_is.

$Fonction\ reduce(key, value)$

$var\ sum = 0$

$for(var\ i\ in\ values)$

$\quad sum = sum + values[i].Similarite$

$return\ \{termId : key,\ Similarite : sum\}$

Algorithm 14: Reduce function for computing densities Q_i

- Calculating TDV_i:

Once we obtained the values of Q and Q_is, by looping through all terms each value TDV_i by the formula $(Q_i - Q)/Q$.

Incremental calculation:

The incremental calculation of TDVs with the centroid method needs four steps. The first and the last will run in central memory, the other two are map reduce functions.

- Modify the vector c:

Add the weight of each term of the document added to the value of component of the vector centroid whose index is the identifier of the term. Deduct the value of this term if the document is deleted.

Once the centroid is changed the three next steps consist of:

1. Calculating the density Q with algorithm.
2. Calculating densities Q_i with algorithm.
3. Calculating TDV of each term t_i by application of formula $(Q_i - Q)/Q$.

4.3.2. Naive method

In the following we present the details of the computation of integer and incremental calculation of TDV with the naïve method.

Complete calculation:

TDVs are calculated using the formula (Q_i-Q/ Q). They are calculated after obtaining density Q and densities Q_i. The density Q is obtained by summing similarities between all pairs of

documents. A Q_i is obtained by summing similarities between all pairs of documents by ignoring the term t_i in all documents.

- Calculating similarities:

Theses similarities are obtained through a method proposed by (Esayed and all, 2008) which is to apply a map-reduce to the collection and another on the obtained results. The two step are presented in following:

Step 1:

A map function is called. For each term of a document this function sends an emit with the Id of the term as key and an object as value. This object contains the Id of the document and value which is the division of the weight the term by the weight of the document.

$Function\ map()$

$var\ sim = 0$

$for(var\ term\ in\ values)$

$sim = this.terms[term].termWeight/this.docWeight$

$emit(this.terms[term].termID,$
$\{termId: this.terms[term].termID, Chaine: \{doc: this.docID, Occ: sim\}\})$

Algorithm 15: Map function in first step for computing similarities

A reduce function is called. It groups objects emitted with same key in an array.

$Function\ reduce(key, value)$

$var\ chaine = []$

$for(var\ i\ in\ values)$

$chaine.push(value[i])$

$return\ \{termId: key, Chaine: chaine\}$

Algorithm 16: Reduce function in first step for computing similarities

Step 2:

A map function is called. For each document of the collection obtained after the first step this function perform for each pair of objects "*Chaine*" that contained by this document a multiplication between values *Occ* these objects and send and emit with a combination of values doc of these objects as key and the results of the multiplication as value.

$Function\ map()$

$var\ sim = 0$

$if\ this.value.Chaine[0]$

$for(var\ d\ in\ this.value.Chaine)$

$for(var\ d1\ in\ this.value.Chaine)$

$if\ d < d1$

$sim = this.value.Chaine[d].Chaine.Occ$

$* this.value.Chaine[d1].Chaine.Occ$

$emit(\{Id1: this.value.Chaine[d].Chaine.doc,$
$Id2: this.value.Chaine[d].Chaine.doc\ \},$
$\{Ids: \{Id1: this.value.Chaine[d].Chaine.doc,$
$Id2: this.value.Chaine[d].Chaine.doc\}, Sim: sim\})$

Algorithm 17: Map function in second step for computing similarities

The following reduce function is called to calculate amounts of values emitted with same keys.

$Function\ reduce(key, value)$

$var\ sum = 0$

$for(var\ i\ in\ values)$

$sum = values[i].Sim$

$return\ \{Ids: key, Sim: sum\}$

Algorithm 18: Reduce function in second step for computing similarities

- Calculating Q:

Density Q is computed by summing all similarities obtained.

- Calculating Q_is:

After calculating the density Q. Each density Q_i is calculated by applying to the collection in the first step a map-reduce similar to that of the second step by adding a condition in the

map function to ignore the term t_i. This map-reduce is applied N times such that N is the number of the vocabulary terms.

The following function map execute the same job that the function of algorithm (17) by ignoring the term which t_i as Id.

$Function\ map()$

$var\ sim = 0$

$var\ vec = [chaineTerms]$

$var\ index = i$

$if(this.value.termID\ != vec[index])$

$if\ this.value.Chaine[0]$

$for(var\ d\ in\ this.value.Chaine)$

$for(var\ d1\ in\ this.value.Chaine)$

$if\ d < d1$

$sim = this.value.Chaine[d].Chaine.Occ$

$* this.value.Chaine[d1].Chaine.Occ$

$emit(\{Id1: this.value.Chaine[d].Chaine.doc,$
$Id2: this.value.Chaine[d].Chaine.doc\ \},$
$\{Ids: \{Id1: this.value.Chaine[d].Chaine.doc,$
$Id2: this.value.Chaine[d].Chaine.doc\}, Sim: sim\})$

Algorithm 19: Map function for computing similarities by ignoring a term t_i

The reduce function is called to sum values emitted with same keys.

$Function\ reduce(key, value)$

$var\ sum = 0$

$for(var\ i\ in\ values)$

$sum = values[i].Sim$

$return\ \{Ids: key, Sim: sum\}$

Algorithm 20: Reduce function for computing similarities by ignoring a term t_i

Incremental calculation:

The formula for calculating TDV of each term $t_i$is $(Q - Q_i)/Q$. After adding or deleting a document value Q and values Q_i are modified. Functions map "algorithm (11)" and reduce "algorithm (12)" are used to compute the new value of density Q by replacing centroid by inserted or deleted document and adding (or subtracting) obtained result to the value of Q saved in file Q_naive. By inserted or deleted document functions map "algorithm (13)" and reduce "algorithm (14)" are used to compute new values of Q_i. Each value obtained is added (or subtracted) to value that which corresponds to it in file Qi_naive. Then, we can compute new values of TDV.

4.3.3. Clustering Method:

We will detail in that follows the implementation of the integer and incremental calculations whit the clustering method.

Complete calculation:

The calculation of TDV values is divided into for distributed computations, the first computation is for calculating vector α, the second is for calculating vector β, the third is for calculating vector δ and the last is for calculating TDV values which depend on the tree vectors.

- Calculating the inverse of α_is:

The map function will be applied to all the terms of documents of the collection. At each call of the map function serval emit are sent. Each emit has $docID$ as key and $termWeight$ as value. This function is called to have for each document weights of terms that compound it.

$Function\ map()$

$for\ (var\ term\ in\ this.terms)$

$$emit(this.docID,\ \{docId: this.docID, weight: term.termWeight\})$$

Algorithm 21: Map function for computing vector α

The reduce function is called after calling the map function. This function calculates the sums of values emitted with same keys. This reduce function gives components of α.

$Fonction\ reduce(key, value)$

$for(var\ i\ in\ values)$

$$sum = sum + values[i].weight$$

$return\ \{docId\ :\ key,\ weight\ :\ sum\}$

Algorithm 22: Reduce function for computing vector α

- Calculating inverses of β_js:

A call the map function is performed for each document. This function is applied to all terms of documents. The key of each emit is $termID$ and its value is $termWeight$. This function is called to group for each term its weight in documents.

$Fonction\ map()$

$for(var\ term\ in\ this.terms)$

$$emit(this.terms[term]termID, \{termId: this.terms[term].termID$$

$$, weight: this.term[term].termWeight\})$$

Algorithm 23: Map function for computing vector β

A reduce function is called. It calculates the sums of values emitted with same key. This function gives components of vector β.

$Fonction\ reduce(key, value)$

$for(var\ i\ in\ values)$

$$sum = sum + values[i].weight$$

$return\ \{termId\ :\ key,\ .weight : sum\}$

Algorithm 24: Reduce function for computing vector β

- Calculating the vector δ:

The map function is called for each document of the selection. This function takes the vector "*chaineBeta*" which contain β_js as parameter. The map function is applied all terms t_j of documents of the collection by multiplying the square of the weight of the term by β_j emitting the obtained value with the key $termID$. This function allows grouping for each term squares of its weight in documents multiplied by the component of the vector β corresponding to this term.

$Fonction\ map()$

$var\ sum = 0$

$var\ beta = [chaineBeta]$

$for(term\ in\ this.terms)$

$$sum = this.terms[term].termweight * this.terms[term].termWeight / beta[this.terms[term].termID]$$

$emit(this.docID\ , \{docId: this.docID, Sum: sum\})$

Algorithm 25: Map function for computing vector δ

The reduce function is called to calculate the sums of values emitted by map function with the same key and multiply them by α_{Key}. Components of vector δ are obtained with this function.

$Fonction\ reduce(key, value)$

$var\ alpha = [chaineAlpha]$

$var\ sum = 0$

$Foreach(var\ i\ in\ values)$

$\quad sum = sum + values[i].Sum$

$sum = \ sum/alpha[key]$

$Return\ \{docId\ :\ key,\ \ Sum\ :\ sum\}$

Algorithm 26: Reduce function for computing vector δ

- Calculating TDV_i:

The map function is called. It takes vectors α, β and δ as parameters. For each term t_l of a document d_i the value $\delta_i - \alpha_i^l * (\frac{\delta_i}{\alpha_i} - {d_{il}}^2 * \beta_l)$ is calculated and emitted with t_l as key. Calling this function allows to have multiple groups of values. The sum of vales of each group gives TDV a given term.

$Fonction\ map()$

$var\ alpha = [chaineAlpha]$

$var\ beta = [chaineBeta]$

$var\ delta = [chaineDelta]$

$var\ val = 0$

$for(var\ term\ in\ this.terms)$

$$val = chaineDelta[this.terms[term].termID] - (1/(alpha[this.docID] - this.terms[term].termWeight) * (delta[this.docID] * alpha[this.docID] - (this.terms[term].termWeight * term.termWeight/beta[this.terms[term].termID]))$$

$emit\ (this.terms[term]: termID, \{termId :$
$this.terms[term]: termID, tdv: val\})$

Algorithm 27: Map function for computing *TDV*

A reduce function is called after the calling of map function. It calculates the sums of values emitted with same key. This reduce function gives values of *TDV*s.

$Fonction\ reduce(key, value)$

$var\ sum = 0$

$for(var\ i\ in\ values)$

$sum = sum + values[i].tdv$

$return\ \{termId :\ key,\ val : sum\}$

Algorithm 28: Reduce function for computing TDV

Incremental calculation:

When adding or deleting a document some TDV will change. It is very difficult to know what values of TDV will modified, that is why we will recalculate TDV of all terms. To recalculate TDVs four steps are needed, three will be in main memory:

- Calculate the value α_{M+1} that is the sum of weights of terms of the document d_{M+1} and add it to alpha_clustering file if the modification of collection is adding a document. Remove α_i from alpha_clustering file if the modification of the collection is deleting a document d_i.
- Modify $\beta_j$for all terms t_j of the document added or deleted document and save them in beta_cluetering file.
- Add δ_{M+1} if the modification of collection is adding a document. Remove δ_i if the modification of collection is deleting a document. Modify values of δ associated with documents that contains some of terms of document added or deleted.

Once the new values of vectors α, β and δ are obtained and the new values of TDV can be recalculated using the map function defined by algorithm (27) and the reduce function defined by algorithm (28)**.**

4.3.4. Dot Product method:

We will see in the following the details of the implementation of integer and incremental calculations of TDV with Dot Product method.

Complete calculation:

The calculations of TDV need steps that are map reduce functions that are computing of vector c and computing of matrix DIF and one step that will be executed on central memory that is computing of TDV values.

- Computing vector c:

To compute centroid's components of dot Product method, the map function is called. For each term of a document is associated an emit that has the identifier of the term as key and its weight as value. Calling of this allows grouping for each term its weights in documents divided by weights of documents.

$Fonction\ map()$

$var\ sum$

$for\ (var\ term\ in\ this.terms)$

$sum = this.terms[term].termWeight/this.docWeight$
$emit(this.terms[term].termID,$
$\{termId: this.terms[term].termID, weight: sum\})$

Algorithm 29: Map function for computing vector c of dot Product method

The reduce function is called after the map function. It calculates amounts of weights issued with the same keys. The components of the vector c are then obtained.

$Fonction\ reduce(key, value)$

$var\ sum = 0;$

$Foreach(var\ i\ in\ values)$

$sum = sum + values[i].weight;$

$return\ \{termId : key, weight: sum\};$

Algorithm 30: Reduce function for computing vector c of dotProd method

- Computing matrix DIF :

A map function is called it send for each document d_j, for each pair of terms $\{t_i, t_k\}$ an emit with the key $t_i, t_k\}$ and the value $(wi_j - w_j) * d_{jk}$ where w_j is the weight of the document d_j, wi_j is the weight of the document d_j without t_i and d_{jk} is the weight of the term t_k in the document d_j. Calling this function allows to have multiple groups of values. The sum of each group gives a box of the matrix "dif".

$Fonction\ map()$

$var\ val$

$for\ (var\ term\ in\ this.terms)$

$for\ (var\ term1\ in\ this.terms)$

$val = (1/(this.docWeight - this.terms[term].termWeight * this.terms[term].termWeight) - 1/this.docWeight) * this.terms[term1].termWeight$

$emit(\{Id1: this.terms[term].termID, Id2: this.terms[term1].termID\},$

$\{Ids: \{Id1: this.terms[term].termID, Id2: this.terms[term1].termID\}, , weight: val\})$

Algorithm 31: Map function for computing collection dif:

After the calling to map function, a reduce function is called, it sum values that corresponds to same keys. The collection "dif" is obtained after the calling of this function.

$Reduce\ map()$

$var\ sum$

$for\ (var\ i\ in\ values)$

$sum += value[i].dif$

$return\ \{Ids: key, dif: sum\}$

Algorithm 32: Map function for computing collection "dif":

- Computing TDV:

For computing TDV of each term, a map function is called, it sends for each document of the collection "dif" (matrix "dif") an emit with the first part of the id of the document as the key and the addition of the value "dif" and the multiplication of the number two by of the component of centroid which corresponds to the second part of the id of the document.

$Function\ map()$

$var\ centroid\ = [chaineCentroid]$

$var\ sum$

$sum = this.value.dif * (this.value.dif + 2 * centroid[this.value.Ids.Id2])$

$emit(\ this.value.Ids.Id1\ ,\{\ termID: this.value.Ids.Id1\ , tdv: sum\})$

Algorithm 33: Map function for computing TDV in dotProd method

A reduce function is called for summing values emitted with same keys. TDVs are then obtained.

$Function\ reduce()$

$var\ sum = 0$

$for\ (var\ i\ in\ values)$

$\quad sum = values[i].tdv$

$return\ \{tremID: key\ , tdv: sum\}$

Algorithm 34: Reduce function for computing TDV in dot product method

Incremental method:

After adding or deleting a document changing TDV values will be in main memory. The operation of loading the vector centroid and the matrix DIF and modify the changed values. This will allow recalculate TDV values. Three steps are then needed to recalculate TDV values:

- Recalculate modified components of centroid. These components are terms of document added or deleted. To change a component consists to add to the current value the value of division of the weight of the term in document added by the weight of document and deduct the value of this division from the current value in the case of deletion.
- Modify for each vector DIF^i cases k that correspond to terms contained by added or deleted document.
- Recalculate values of TDVs.

5. Experiments

The experiments will be performed on a dataset that represents all of the publications of *FiND* system in November 2010. These data are available on file in ".csv" format. It is from this data that MongoDB databases of different sizes are generated.

The experiments will be performed on machines of the "salles des marchées 2" which is a pratical works room in CNAM . This room supports thirty machines called samar31, samar32, ..., samar60. In order to compare the different methods we will do locally and distributed tests. To make distributed tests the machine samar32 is designed as mongos server, machines samar33, samar34, and samar35 are designed as configuration servers and other machines are the shards.

Experiments on a single server:

Experiments on multiple servers:

6. Conclusion

The amount of data that is generated by the web requires the adaptation of different platforms to enable scaling. The *FiND* platform should store millions of items and use of very expensive calculations in terms of memory and time. This is why these calculations have to be optimized and distributed.

To enable this scaling, we used *NoSQL* databases and Map-Reduce framwork. Different methods were implemented to run on *NoSQL* data. To enable the distribution, these methods use functions Map-Reduce.

The DBMS that was adopted is *MongoDB*, this system perfectly meets our requirement that is to have *NoSQL* data distributed across multiple clusters. A data structure has been proposed and the implementation of the methods has been carried out taking account of the structure.

The first experiments carried out confirm that the naive method is the most expensive method and that must be avoided in a real context. These experiments indicate that dot Product method is poorly distributable despite its good complexity in theory. The centroid and clustering methods are the most interesting, but the centroid method gives accurate results and clustering method gives approximate results.

We have seen this work find the system performances can be improved by minimizing the time of calculation of *TDV* values but there's many ways to improve this work.

We can achieve more experiences especially in a distributed environment to see the behavior of these methods when run on a large number of clusters. We can also make tests with larger data to approximate a real context. We choose to use the DBMS *MongoDB* but we can explore ways to backup data and show if it improve performances.

Bibliography

(Willett, 1984) Peter Willett, An algorithm for the calculation of exact term discrimination values, Information Processing & Management, Volume 21, Issue 3, 1985, Pages 225-232, ISSN 0306-4573,

(Can and Ozkarahan,1990) Fazli Can and Esen A. Ozkarahan. 1990. Concepts and effectiveness of the cover-coefficient-based clustering methodology for text databases. *ACM Trans. Database Syst.* 15, 4 (December 1990), 483-517. DOI=10.1145/99935.99938

(El-Hamdouchi and Willett, 1988) Abdelmoula El-Hamdouchi and Peter Willett. 1988. An improved algorithm for the calculation of exact term discrimination values. *Inf. Process. Manage.* 24, 1 (January 1988), 17-22. DOI=10.1016/0306-4573(88)90073-8

(Esayed and all,2008) Tamer Elsayed, Jimmy Lin, and Douglas W. Oard. 2008. Pairwise document similarity in large collections with MapReduce. In *Proceedings of the 46th Annual Meeting of the Association for Computational Linguistics on Human Language Technologies: Short Papers* (HLT-Short '08). Association for Computational Linguistics, Stroudsburg, PA, USA, 265-268.