Benchmarking Top-K Keyword and Top-K Document Processing with T^2K^2 and $T^2K^2D^2$

Ciprian-Octavian Truică[1,a], Jérôme Darmont[2,b], Alexandru Boicea[1,c], Florin Rădulescu[1,d]

[1] *Computer Science and Engineering Department, Faculty of Automatic Control and Computers, University Politehnica of Bucharest, Bucharest, Romania*
[2] *Université de Lyon, Lyon 2, ERIC EA 3083, France*
[a] *ciprian.truica@cs.pub.ro,* [b] *jerome.darmont@univ-lyon2.fr,*
[c] *alexandru.boicea@cs.pub.ro,* [d] *florin.radulescu@cs.pub.ro*

Abstract

Top-k keyword and top-k document extraction are very popular text analysis techniques. Top-k keywords and documents are often computed on-the-fly, but they exploit weighted vocabularies that are costly to build. To compare competing weighting schemes and database implementations, benchmarking is customary. To the best of our knowledge, no benchmark currently addresses these problems. Hence, in this paper, we present T^2K^2, a top-k keywords and documents benchmark, and its decision support-oriented evolution $T^2K^2D^2$. Both benchmarks feature a real tweet dataset and queries with various complexities and selectivities. They help evaluate weighting schemes and database implementations in terms of computing performance. To illustrate our benchmarks' relevance and genericity, we successfully ran performance tests on the TF-IDF and Okapi BM25 weighting schemes, on one hand, and on different relational (Oracle, PostgreSQL) and document-oriented (MongoDB) database implementations, on the other hand.

Keywords: top-k keywords; top-k documents; text analytics; benchmarking; weighting schemes; database systems

1. Introduction

With the tremendous and continuous development of textual web contents, especially (but not only) on social media, text analytics has become a pivotal trend that still proposes many challenges nowadays. Among text analytics techniques, top-k keyword and document extraction is very popular. For instance, extracting the k most frequent terms from a corpus helps determine trends [1, 2] or detect specific events [3]; and finding the k documents that are the most similar to a query is of course the core task of search engines. Moreover, computing top-k keywords and documents requires building a weighted vocabulary, which can also be used for many other text analytics purposes such as topic modeling [4] and clustering [5].

To compare combinations of weighting schemes, computing strategies and physical implementations, benchmarking is customary. However, most prominent big data benchmarks [6, 7, 8] focus on MapReduce operations and do not specifically model text-oriented workflows. Very few benchmarks do, but still focus on Hadoop-like architectures and remain at the methodology [9] and specification [10] stage. Moreover, computing weighting schemes at the application level can prove inefficient when working with large data volumes, because all the information must be queried, read and processed at a different layer but storage. A better approach is to process the information at the storage layer using aggregation functions and then return the response to the application layer.

Hence, we introduced the Twitter Top-K Keywords Benchmark (T^2K^2) [11], which relies on database storage and features a real tweet dataset, as well as queries with various complexities and selectivities. We also designed T^2K^2 to be somewhat generic, i.e., it can compare various weighting schemes, database logical and physical implementations and even text analytics platforms in terms of computing efficiency. In this paper, we further this work with the following contributions.

1. Since T^2K^2's data model is generic enough to handle any type of textual documents (not only tweets), we complement its workload model with

top-k document queries.

2. We illustrate T^2K^2's relevance and genericity by performing tests on the TF-IDF and Okapi BM25 weighting schemes, on one hand, and on different relational (Oracle, PostgreSQL) and document-oriented (MongoDB) database implementations, on the other hand.
3. Since T^2K^2 queries are analytical by nature, we hypothesize that a star (data warehouse) schema should improve query response and propose an evolution of T^2K^2 named $T^2K^2D^2$, where the last two Ds stand for Dimensional and Documents, respectively.
4. We complement our first experiments with top-k keyword and top-k document benchmarking with $T^2K^2D^2$, still testing the TF-IDF and Okapi BM25 weighting schemes, as well as the Oracle, PostgreSQL and MongoDB database systems.

We designed our benchmarks' queries by logging computational linguists working on a text analysis platform [12, 13] on real-world data. After analyzing and clustering similar user queries, we ended up with 8 queries (4 for T^2K^2 and 4 for $T^2K^2D^2$) that we consider generic enough to benchmark any similar system.

Moreover, note that our benchmarks do not apply to Information Retrieval (IR) system evaluation. They rather target database systems built on top of an IR system, with the aim of facilitating text mining, e.g., text classification, topic modeling or event detection. Benchmark queries are relevant to machine learning tasks, where it is important to analyze different subsets of the dataset to extract knowledge. In contrast, IR systems do not handle subsets of the initial corpus well, nor datasets whose size changes in time, because they compute weights only once. However, this creates two result reproducibility problems [14]. First, when using subsets of the initial corpus, weights are not recomputed, which induces errors when computing the ranking functions for top-k keywords and documents. Second, weights must be recomputed each time the size of the dataset changes, inducing more write operations that are

computationally demanding.

The database approach can solve these result reproducibility problems by dynamically computing weights at search time using fielded data. Although some IR systems do use fielded data [15], as they systems compute weights only once, ranking functions do not update weights when subsets of the initial corpus are used or when the volume of data changes [16], which induces errors during the extraction of top-k documents and keywords. Moreover, computing weights dynamically is ideal for analyzing subsets of the initial dataset or corpora whose size changes over time, e.g., when analyzing stream data [17]. Database systems are designed to manage large volumes of data with a high throughput and can easily manipulate changing volumes of data by using optimized CRUD (Create, Read, Update, Delete) operations. Thus, they are better suited for handling datasets whose size changes over time and analyzing subsets of the initial corpus.

The remainder of this paper is organized as follows. In Section 2, we survey existing big data and more specifically text processing-oriented benchmarks. In Section 3, we recall T^2K^2's data and workload models, and provide $T^2K^2D^2$'s full specifications. In Section 4, we detail how we implement two weighting schemes (namely, TF-IDF and Okapi BM25) and instantiate our benchmarks in Oracle, PostgreSQL and MongoDB. In Section 5, we account for the experiments we performed with both T^2K^2 and $T^2K^2D^2$, which demonstrate the feasibility and relevance of our benchmarks. Finally, we conclude the paper and provide research perspectives in Section 6.

2. Related Works

2.1. Big Data Benchmarks

There exist many big data benchmarks, which are mostly data-centric, i.e., they focus on volume and MapReduce-based applications, rather than on variety, and do not include textual data.

For instance, the quasi-standard TPCxHS benchmark models a simple application and features, in addition to classical throughput and response time met-

rics, availability and energy metrics [8]. Similarly, HiBench is a set of Hadoop programs, ranging from data sorting to clustering, aimed at measuring metrics such as response time, HDFS bandwidth consumption and data access patterns [6].

In contrast, BigDataBench, among 19 benchmarks, features application scenarios from search engines, i.e., the application on Wikipedia entries of operators such as Grep or WordCount [7]. Yet, although BigDataBench is open source, it is quite complex and difficult to extend to test the computation efficiency of term weighting schemes.

2.2. Text Analysis Benchmarks

Term weighting schemes are extensively benchmarked in several subdomains of text analytics. In sentiment analysis, dictionary-based methods are benchmarked on various types of corpora (news articles, movie reviews, books and tweets) [18] to perform both quantitative and qualitative assessments.

In text classification and categorization, benchmarks target different types of texts: short texts represented by sentence pairs with similarity ratings [19]; large texts from Reuters newswire stories on one hand [20], and DBpedia and the Open Directory Project on the other hand [21]; or texts in a specific language [22].

In terms of metrics, all the above-mentioned benchmarks focus on algorithm accuracy. Either term weights are known before the algorithm is applied, or their computation is incorporated with preprocessing. Thus, such benchmarks do not evaluate weighting scheme construction efficiency as we do.

Finally, TextGen is a synthetic textual data generator [23]. TextGen builds corpora by segmenting real-world text datasets and enforces a lognormal word-frequency distribution. However, TextGen aims at testing the compression performance of word-based compressors, and thus provides no workload or metrics that are suitable to our needs.

2.3. Parallel Text Processing Benchmarks

This last family of benchmarks evaluates parallel text processing in big data, cloud applications. However, it is very small (two benchmarks only). The first one is actually a methodology for designing text-oriented benchmarks in Hadoop [9]. It provides both guidelines and solutions for data preparation and workload definition. Yet, as text analysis benchmarks, its metrics measure the accuracy of analytics results, while we aim at evaluating computing performance.

PRIMEBALL features a fictitious news site hosted in the cloud that is to be managed by the framework under analysis, together with several objective use cases and measures for evaluating system performance [10]. One of its metrics notably involves searching a single word in the corpus. However, PRIMEBALL remains a specification only as of today.

3. Benchmark Specifications

Text analysis deals with discovering hidden patterns from texts. In most cases, it is useful to determine such patterns for given groups, e.g., males and females, because they have different interests and talk about disjunct subjects. Moreover, if new events appear, depending on the location and time of day, these subject can change for the same group of people. The queries we propose aim to determine such hidden patterns and improve text analysis and anomaly detection.

Typically, a benchmark is constituted of a data model (conceptual schema and extension), a workload model (set of operations) to apply on the dataset, an execution protocol and performance metrics [24]. In this section, we provide a conceptual description of T^2K^2 and $T^2K^2D^2$, so that it is generic and can cope with various weighting schemes and database logical and physical implementations. For generalization purposes, the models will contain the words "document" and "documents" for naming the entities, instead of words "tweet" and "tweets".

3.1. Text Preprocessing

Text preprocessing is an important task used in Text Mining and Analysis, Information Retrieval and Natural Language processing. Text preprocessing involves cleaning and preparing texts for different tasks, e.g., classification, IR or sentiment analysis. Usually, textual data contain lot of noise and uninformative parts such as HTML tags, scripts and links. At the word level, common words impact negatively the task at hand. Removing these so-called stop words decreases the problem's dimensionality [25]. Moreover, preprocessing is used to extract information and metadata from text, standardize the document corpus, remove useless information and minimize the dimensionality of the data.

In our benchmarks, tweets are preprocessed *a priori*. Lemmas are extracted from each word and stop words removed to minimize vocabulary size and improve the accuracy of machine learning algorithms such as topic modeling and document clustering. End users can preprocess data as they see fit and then store the information in the database using the models provided by the benchmark. In our preprocessing step, words that are abbreviated are considered words in their own sense, i.e., they are not expanded. For instance, we consider "omg" as a word on his own. Misspellings are currently retained, although in a next version of the preprocessing step, we plan to address this problem. Moreover, abbreviations and misspellings only affect the selectivity of queries, as the vocabulary stores them too.

The preprocessing steps we apply on raw texts to construct the weighted vocabulary follow.

1. For each tweet, we extract tags, i.e., hashtags (which make good keywords), attags, and remove links.
2. We expand contractions, i.e., shortened versions of the written and spoken forms of a word, syllable, or word group, created by omission of internal letters and sounds [26]. For example, "it's" becomes "it is".
3. We split the text of a tweet into sentences.
4. We extract the part of speech for each sentence.

5. We build a clean text by removing punctuation and stop words.
6. For each term in a clean text, we use the part of speech to extract lemmas and create a lemma text.
7. For each lemma t in lemma text d, we compute the number of co-occurrences $f_{t,d}$ and term frequency $TF(t,d)$, which normalizes $f_{t,d}$.

3.2. T^2K^2

3.2.1. Data Model

All the information and metadata extracted during the preprocessing step are modeled using the entity-relationship presented in Figure 1. Let us describe all entities in this conceptual model.

- *Author* stores information about a tweet author, e.g., unique identifier, firstname, lastname, and age.
- *Gender* is used to store the authors gender and to remove duplication in the database.
- *Document* entity stores all the textual information and metadata for a tweet, e.g., the unique identifier of the tweet, the date, and the original and the processed text of the text, i.e., original text (*RawText*), clean text (*CleanText*) and lemma (*LemmaText*).
- *Writes* is the many-to-many relationship between an author and a tweet.
- to eliminate duplicates and minimize the storage space, the *Geo_Location* is used to store the geo-location for each tweet.
- *Word* entity stores the lemma and its unique identifier.
- *Vocabulary* is a many-to-many relationship between the *Word* and *Document* entities that stores, beside the unique identifiers of the lemma and tweet, the number of co-occurrence $f_{t,d}$ and term frequency $TF(t,d)$ for a lemma in a tweet.

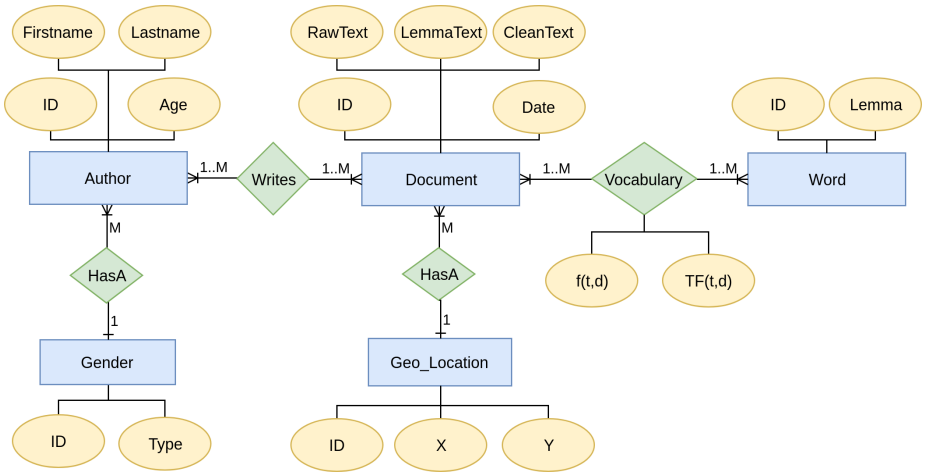


Figure 1: T^2K^2 conceptual data model

3.2.2. Workload Model

T^2K^2 features 4 queries, from $Q1$ to $Q4$, which are grouping queries that extract the top-k keywords with different constraints that are combined in the queries: $c_1(Q1)$, $c_1 \wedge c_2(Q2)$, $c_1 \wedge c_3(Q3)$ and $c_1 \wedge c_2 \wedge c_3(Q4)$. We define constraints c_1 to c_3 below.

- c_1 is $Gender.Type = pGender$ with $pGender \in \{male, female\}$.
- c_2 is $Document.Date \in [pStartDate, pEndDate]$, where $pStartDate$, $pEndDate \in$ *[2015-09-17 20:41:35, 2015-09-19 04:05:45]* and $pStartDate < pEndDate$.
- c_3 is $Geo\_location.X \in [\, pStartX, pEndX]$ and $Geo\_Location.Y \in [pStartY, pEndY]$, where $pStartX, pEndX \in$ *[15, 50]*, $pStartX < pEndX$, $pStartY, pEndY \in$ *[-124, 120]* and $pStartY < pEndY$.

For extracting top-k documents, an additional constraint c_4 on the terms of the vocabulary is needed. This constraint is used to select only the tweets that contain at least one of the word from a list of search terms and construct the tweets hierarchy only for them. Therefore, c_4 is $Words.Lemma = pTerms$ where

$pTerms \in \{\ term \mid term \in vocabulary\ \}$. The queries in this case rewritten using the constraints c_1 to c_4 as $c_1 \wedge c_4(Q1)$, $c_1 \wedge c_2 \wedge c_4(Q2)$, $c_1 \wedge c_3 \wedge c_4(Q3)$ and $c_1 \wedge c_2 \wedge c_3 \wedge c_4(Q4)$.

3.3. $T^2K^2D^2$

3.3.1. Data Model

T^2K^2's workload model (Section 3.2.2) is made of grouping queries akin to OLAP queries. Thus, to improve query performance, we remodel its schema into a multidimensional star schema (Figure 2) featuring a central fact table and multiple dimensions. $T^2K^2D^2$'s schema contains the following entities.

- *Document_Fact* is the central fact entity and contains the number of co-occurrence $f_{t,d}$ and term frequency $TF(t, d)$ for a lemma in a document.
- *Document_Dimension* is the tweets dimension table, containing each tweet's unique identifier and the original and processed text, i.e., original text (*RawTtext*), clean text (*CleanText*) and lemma text(*LemmaText*).
- *Word_Dimension* stores the lema and its unique identifier.
- *Time_Dimension* stores the full date and also its hierarchy composed of minute, hour, day, month and year
- *Author_Dimension* is the author dimension and stores information about an author's unique identifier, gender, age, firstname, and lastname.
- *Location_Dimension* is the geo-location dimension.

3.3.2. Workload Model

To adjust to $T^2K^2D^2$'s multidimensional data model, we rewrite T^2K^2's queries as OLAP queries. We retain the same constraints c_1 to c_4, only adapting entity and attribute names. The redefined constraints' specification follows.

- c_1 is $Autor\_Dimension.gender = pGender$ with $pGender \in \{male, female\}$.

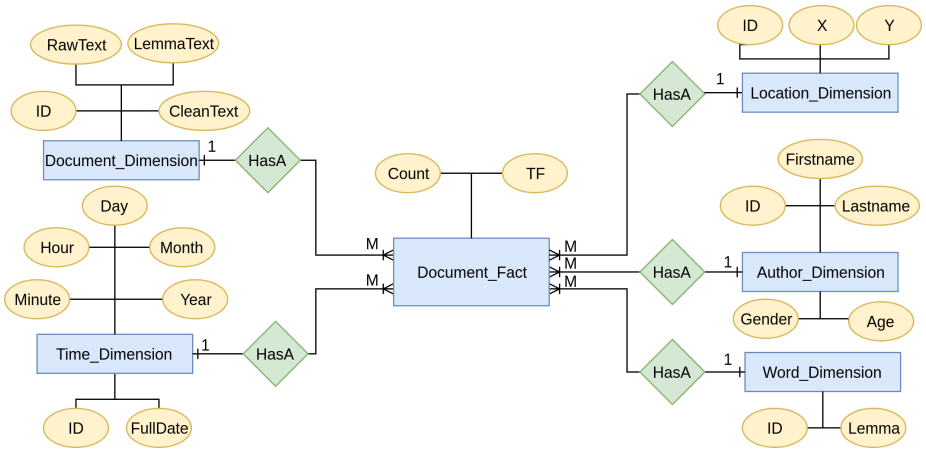


Figure 2: $T^2K^2D^2$ conceptual data model

- c_2 is *Time_Dimension.Date* $\in$ *[pStartDate, pEndDate]*, where *pStartDate, pEndDate* $\in$ *[2015-09-17 20:41:35, 2015-09-19 04:05:45]* and *pStartDate* $<$ *pEndDate.*

- c_3 is *Location_Dimension.X* $\in$ *[pStartX, pEndX]* and *Location_Dimension.Y* $\in$ *[pStartY, pEndY]*, where *pStartX, pEndX* $\in$ *[15, 50]*, *pStartX* $<$ *pEndX*, *pStartY, pEndY* $\in$ *[-124, 120]* and *pStartY* $<$ *pEndY.*

- c_4 is *Word_Dimension.Lemma* $=$ *pTerms* where *pTerms* $\in \{t \mid t \in$ *vocabulary* $\}$.

Then, OLAP queries that extract top-k keywords and documents bear the following contraints: $c_1(Q1)$, $c_1 \wedge c_2(Q2)$, $c_1 \wedge c_3(Q3)$ and $c_1 \wedge c_2 \wedge c_3(Q4)$; and $c_1 \wedge c_4(Q1)$, $c_1 \wedge c_2 \wedge c_4(Q2)$, $c_1 \wedge c_3 \wedge c_4(Q3)$ and $c_1 \wedge c_2 \wedge c_3 \wedge c_4(Q4)$, respectively.

3.4. Performance Metrics and Execution Protocol

We use query response time as the only metric in both T^2K^2 and $T^2K^2D^2$. It is detail for each query as $t(Q_i) \forall i \in [1, 4]$. All queries $Q1$ to $Q4$ are executed 40 times for top-k keywords and 10 times for top-k documents, which is sufficient according to the central limit theorem. Average response times and standard

deviations are computed for $t(Q_i)$. All executions are warm runs, i.e., either caching mechanisms must be deactivated, or a cold run of $Q1$ to $Q4$ must be executed once (but not taken into account in the benchmark's results) to fill in the cache. Queries must be written in the native scripting language of the target database system and executed directly inside said system using the command line interpreter.

4. Benchmark Implementations

4.1. Weighting Schemes

Given a corpus of documents $D = \{d_1, d_2, ..., d_N\}$, where $N = |D|$ is the total number of documents in the dataset and n the number of documents where some term t appears. The TF-IDF weight is computed by multiplying the augmented term frequency $TF(t,d) = K + (1-K) \cdot \frac{f_{t,d}}{\max_{t' \in d}(f_{t',d})})$ by the inverted document frequency $IDF(t,D) = 1 + \log \frac{N}{n}$, i.e., $TFIDF(t,d,D) = TF(t,d) \cdot IDF(t,D)$. The augmented form of TF prevents a bias towards long tweets when the free parameter K is set to 0.5 [27]. It uses the number of co-occurrences $f_{t,d}$ of a word in a document, normalized with the frequency of the most frequent term t', i.e., $\max_{t' \in d}(f_{t',d})$.

The Okapi BM25 weight is given in Equation (1), where $||d||$ is d's length, i.e., the number of terms appearing in d. Average document length $avg_{d' \in D}(||d'||)$ is used to remove any bias towards long documents. The values of free parameters k_1 and b are usually chosen, in absence of advanced optimization, as $k_1 \in [1.2, 2.0]$ and $b = 0.75$ [28, 29, 30].

$$Okapi(t,d,D) = \frac{TFIDF(t,d,D) \cdot (k_1 + 1)}{TF(t,d) + k_1 \cdot (1 - b + b \cdot \frac{||d||}{avg_{d' \in D}(||d'||)})} \tag{1}$$

To extract top-k keywords, the overall relevance of a term t for a given corpus D is computed as the sum of all the TF-IDF (Equation (2)) or Okapi BM25 (Equation (3)) weights for that term.

$$S_{TK}\_TFIDF(t, D) = \sum_{d_i \in D} TFIDF(t, d_i, D) \tag{2}$$

$$S_{TK}\_Okapi(t, D) = \sum_{d_i \in D} Okapi(t, d_i, D) \tag{3}$$

TF-IDF and Okapi BM25 are also used to ranks a set of documents based on the search query's terms appearing in each document. Given a search query $Q = \{q_1, q_2, ..., q_m\}$, where $m = |Q|$ is the number of terms contained in the query, a document d is scored by either summing all the TFDIF (Equation (4)) or the Okapi BM25 (Equation (5)) scores for the query terms in the document.

$$S_{TD}\_TFIDF(Q, d, D) = \sum_{q_i \in Q} TF(q_i, d) \cdot IDF(q_i, D) \tag{4}$$

$$S_{TD}\_Okapi(Q, d, D) = \sum_{q_i \in Q} \frac{TFIDF(q_i, d, D) \cdot (k_1 + 1)}{TF(q_i, d) + k_1 \cdot (1 - b + b \cdot \frac{||d||}{avg_{d' \in D}(||d'||)})} \tag{5}$$

4.2. T^2K^2 implementation

4.2.1. Relational implementation

Dataset. The logical relational schema for T^2K^2 used in both Oracle and PostgreSQL (Figure 3) directly translates the conceptual schema from Figure 1.

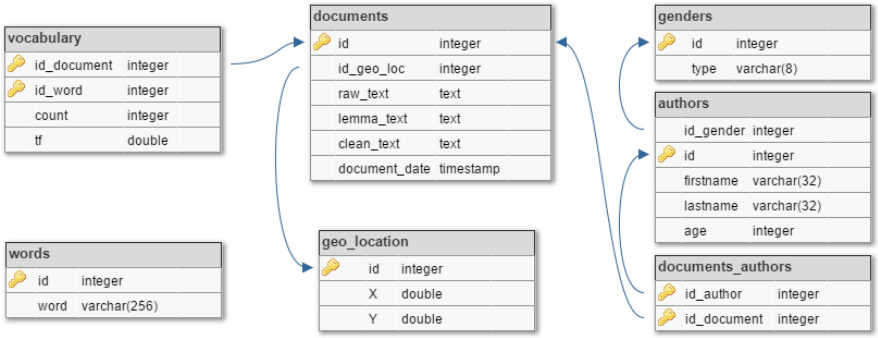


Figure 3: T^2K^2 relational data model

T^2K^2 queries. The specification of $\mathrm{T^2K^2}$'s top-k keywords queries are expressed in relational algebra below.

- $Q1 = \gamma_L (\pi_{documents.id, words.word, f_w} (\sigma_{c_1} (documents \bowtie_{c_5} documents\_authors \bowtie_{c_6} authors \bowtie_{c_7} genders \bowtie_{c_8} vocabulary \bowtie_{c_9} words)))$.
- $Q2 = \gamma_L (\pi_{documents.id, words.word, f_w} (\sigma_{c_1 \wedge c_2} (documents \bowtie_{c_5} documents\_authors \bowtie_{c_6} authors \bowtie_{c_7} genders \bowtie_{c_8} vocabulary \bowtie_{c_8} words)))$.
- $Q3 = \gamma_L (\pi_{documents.id, words.word, f_w} (\sigma_{c_1 \wedge c_3} (documents \bowtie_{c_5} documents\_authors \bowtie_{c_6} authors \bowtie_{c_7} genders \bowtie_{c_8} vocabulary \bowtie_{c_9} words \bowtie_{c_{10}} geo\_location)))$
- $Q4 = \gamma_L (\pi_{documents.id, words.word, f_w} (\sigma_{c_1 \wedge c_2 \wedge c_3} (documents \bowtie_{c_5} documents\_authors \bowtie_{c_6} authors \bowtie_{c_7} genders \bowtie_{c_8} vocabulary \bowtie_{c_9} words \bowtie_{c_{10}} geo\_location)))$

The constraints for the queries are:

- c_1 is the constraint on gender.
- c_2 is the constraint on time.
- c_3 is the constraint on location.
- c_5 to c_{10} are the join conditions as follows:
  - c_5 is the JOIN condition between the *documents* and *documents_authors* entities.
  - c_6 is the JOIN condition between the *documents_authors* and *authors* entities.
  - c_7 is the JOIN condition between the *authors* and *genders* entities.
  - c_8 is the JOIN condition between the *documents* and *vocabulary* entities.
  - c_9 is the JOIN condition between the *vocabulary* and *words* entities.
  - c_{10} is the JOIN condition between the *documents* and *geo_location*.

Function f_w is used to compute the weighting schema using nested queries:

- TF-IDF with the parameters $vocabulary.tf = TF(t, d)$, the total number of tweets in the corpus and the number of tweets where a term appears; the last two parameters are computed using individual nested queries.
- Okapi BM25 with parameters $vocabulary.tf = TF(t, d)$, the total number of tweets in the corpus, the length of each tweet and the number of tweets where a term appears; the last three parameters are computed using individual nested queries.

The last operator is the aggregation operator γ_L where $L = (F, G)$ is:

- $F = sum(f_w)$, the *sum* is the aggregation function that computes $S_{TK}\_TFIDF(t, D)$ (Equation (2)), respectively $S_{TK}\_Okapi(t, D)$ ((Equation (3))).
- $G = (words.word)$ is a list of attributes in the GROUP BY clause, in the case the terms ($words.word$).

The queries for determining the top-k documents are similar to the ones that compute the top-k keywords, i.e., the join conditions and function f_w are the same. Only the constraint c_4 is added to select the tweets that contain the required search terms. The modification appears at the aggregation operator γ_L where the list of the GROUP BY clause attributes changes, this time grouping is done by using the $documents.id$. Moreover, the aggregation $F = sum(f_w)$ computes the hierarchy of tweets, $S_{TD}\_TFIDF(Q, d, D)$ (Equation (4)) for TF-IDF or $S_{TD}\_Okapi(Q, d, D)$ (Equation (5)) for Okapi BM25, where Q is the list of search terms. T^2K^2's top-k documents queries expressed in relational algebra are:

- $Q1 = \gamma_L (\pi_{documents.id, words.word, f_w} (\sigma_{c_1 \wedge c_4} (documents \bowtie_{c_5} documents\_authors \bowtie_{c_6} authors \bowtie_{c_7} genders \bowtie_{c_8} vocabulary \bowtie_{c_9} words)))$.
- $Q2 = \gamma_L (\pi_{documents.id, words.word, f_w} (\sigma_{c_1 \wedge c_2 \wedge c_4} (documents \bowtie_{c_5} documents\_authors \bowtie_{c_6} authors \bowtie_{c_7} genders \bowtie_{c_8} vocabulary \bowtie_{c_8} words)))$.
- $Q3 = \gamma_L (\pi_{documents.id, words.word, f_w} (\sigma_{c_1 \wedge c_3 \wedge c_4} (documents \bowtie_{c_5} documents\_authors \bowtie_{c_6} authors \bowtie_{c_7} genders \bowtie_{c_8} vocabulary \bowtie_{c_9} words \bowtie_{c_{10}} geo\_location)))$

- $Q4 = \gamma_L(\pi_{documents.id,words.word,f_w}(\sigma_{c_1 \wedge c_2 \wedge c_3 \wedge c_4}(documents \bowtie_{c_5} documents\_authors \bowtie_{c_6} authors \bowtie_{c_7} genders \bowtie_{c_8} vocabulary \bowtie_{c_9} words \bowtie_{c_{10}} geo\_location)))$

4.2.2. Document-ortiented implementation

Dataset. In a Document Oriented Database Management System (DODBMS), all information is typically stored in a single collection. An example of DODBMS document is presented in Figure 4. The many-to-many *Vocabulary* relationship from Figure 1 is modeled as a nested document for each record. The information about user and date become single fields in a document, while the location becomes an array.

```
{   _id : 644626677310603264,
    rawText : "Amanda's car is too much for my headache",
    cleanText : "Amanda is car is too much for my headache",
    lemmaText : "amanda car headache",
    author : 970993142,
    geoLocation : [ 32, 79 ],
    gender : "male",
    age : 23,
    lemmaTextLength : 3,
    words : [ { "tf" : 1, "count" : 1, "word" : "amanda"},
              { "tf" : 1, "count" : 1, "word" : "car" },
              { "tf" : 1, "count" : 1, "word" : "headache"} ],
    date : ISODate("2015-09-17T23:39:11Z") }
```

Figure 4: Sample T^2K^2 DODBMS Document

Queries. In DODBMSs, user-defined functions written in JavaScript are used to compute top-k keywords, regardless of the schema. For T^2K^2, the TF-IDF weight can take advantage of both native database aggregation (aggregation pipeline - AP) and MapReduce (MR). However, due to the multitude of parameters involvedand the calculations needed for the Okapi BM25 weighting

scheme, the NA method is usually difficult to develop. Thus, we recommend to only use MR in benchmark runs.

4.3. $T^2K^2D^2$ implementation

4.3.1. Relational implementation

Dataset. The logical multidimensional star schema for $T^2K^2D^2$ used in both relational databases management systems (Figure 5) directly translates from the conceptual schema from Figure 2.

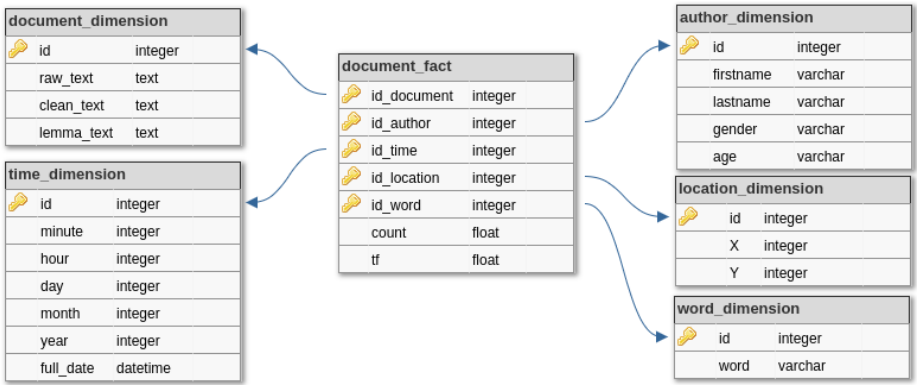


Figure 5: Star schema diagram

Queries. $T^2K^2D^2$'s top-k keywords queries expressed in relational algebra are:

- $Q1 = \gamma_L (\pi_{word\_dimension.word, f_w} (\sigma_{c_1} (document\_fact \bowtie_{c_5} word\_dimension \bowtie_{c_6} author\_dimension)))$.
- $Q2 = \gamma_L (\pi_{word\_dimension.word, f_w} (\sigma_{c_1 \wedge c_2} (document\_fact \bowtie_{c_5} word\_dimension \bowtie_{c_6} author\_dimension \bowtie_{c_7} time\_dimension)))$.
- $Q3 = \gamma_L (\pi_{word\_dimension.word, f_w} (\sigma_{c_1 \wedge c_3} (document\_fact \bowtie_{c_5} word\_dimension \bowtie_{c_6} author\_dimension \bowtie_{c_8} location\_dimension)))$.
- $Q4 = \gamma_L (\pi_{word\_dimension.word, f_w} (\sigma_{c_1 \wedge c_2 \wedge c_3} (document\_fact \bowtie_{c_5} word\_dimension \bowtie_{c_6} author\_dimension \bowtie_{c_7} time\_dimension \bowtie_{c_8} location\_dimension)))$.

The constraints for the queries are:

- c_1 is the constraint on gender.
- c_2 is the constraint on time.
- c_3 is the constraint on location.
- c_5 to c_8 are the JOIN constraints:
  - c_5 is the JOIN constraint between *document_fact* and *word_dimension* entities.
  - c_6 is the JOIN constraint between *document_fact* and *author_dimension* entities.
  - c_7 is the JOIN constraint between *document_fact* and *time_dimension* entities.
  - c_8 is the JOIN constraint between *document_fact* and *location_dimension* entities.

Function f_w is used to compute the weighting schema using nested queries:

- TF-IDF with the parameters $document\_fact.tf = TF(t, d)$, the total number of tweets in the corpus and the number of tweets where a term appears; the last two parameters are computed using individual nested queries.
- Okapi BM25 with parameters $document\_fact.tf = TF(t, d)$, the total number of tweets oin the corpus, the length of each tweet and the number of tweets where a term appears; the last three parameters are computed using individual nested queries.

Finally, we present the aggregation operator γ_L, where $L = (F, G)$ is:

- $F = sum(f_w)$, the *sum* is the aggregation function that computes $S_{TK}\_TFIDF(t, D)$ (Equation (2)), respectively $S_{TK}\_Okapi(t, D)$ ((Equation (3))).
- $G = (word\_dimension.word)$ is a list of attributes in the GROUP BY clause, in the case the terms ($word\_dimension.word$).

$T^2K^2D^2$'s top-k keywords queries expressed in relational algebra are:

- $Q1 = \gamma_L (\pi_{document\_fact.id\_document, f_w} (\sigma_{c_1 \wedge c_4} (document\_fact \bowtie_{c_5} word\_dimension \bowtie_{c_6} author\_dimension)))$.
- $Q2 = \gamma_L (\pi_{document\_fact.id\_document, f_w} (\sigma_{c_1 \wedge c_2 \wedge c_4} (document\_fact \bowtie_{c_5} word\_dimension \bowtie_{c_6} author\_dimension \bowtie_{c_7} time\_dimension)))$.
- $Q3 = \gamma_L (\pi_{document\_fact.id\_document, f_w} (\sigma_{c_1 \wedge c_3 \wedge c_4} (document\_fact \bowtie_{c_5} word\_dimension \bowtie_{c_6} author\_dimension \bowtie_{c_8} location\_dimension)))$.
- $Q4 = \gamma_L (\pi_{document\_fact.id\_document, f_w} (\sigma_{c_1 \wedge c_2 \wedge c_3 \wedge c_4} (document\_fact \bowtie_{c_5} word\_dimension \bowtie_{c_6} author\_dimension \bowtie_{c_7} time\_dimension \bowtie_{c_8} location\_dimension)))$.

T^{2}K^{2}D^2 queries that compute the top-k documents on the star schema are similar with the ones that compute the top-k keywords, only the constraint c_4 is added to select the documents that contain the required search terms. Regardless of the weighting schema used, the f_w function is computed differently because it uses different parameters, as follows:

- $document\_fact.tf = TF(t, d)$.
- the length of each tweet is computed using a nested query
- the number of tweets where a term appears is computed using a nested query

Moreover, the aggregation operator γ_L is also different from the one in the top-k Keywords, i.e., the grouping is done using the tweet unique identifier ($document\_fact.id\_document$) and the aggregation function $F = sum(f_w)$ computes the hierarchy of tweets using $S_{TD}\_TFIDF(Q, d, D)$ (Equation (4)) for TF-IDF, respectively $S_{TD}\_Okapi(Q, d, D)$ (Equation (5)) for Okapi BM25, where Q is the list of search terms.

4.3.2. Document-ortiented implementation

Dataset. To implement T^{2}K^{2}D^2's star schema in a DODBMS each dimension becomes a nested document, thus a document contains multiple nested docu-

ments [31], one for each dimension. Figure 6 presents a document sample for implementing a multidimensional model in a DODBMS.

```
{   "_id" : ObjectId("595c8d705f32f5b46802b24c"),
    "document_dimension" : {
      "id_document" : 644626677310603264,
      "rawText" : "Amanda's car is too much for my headache",
      "cleanText" : "Amanda is car is too much for my headache
    ",
      "lemmaText" : "amanda car headache",
      "lemmaTextLength" : 3 },
    "author_dimension" : {
      "author" : 970993142,
      "age" : 23,
      "gender" : "male" },
    "time_dimension" : {
      "minute" : 39, "hour" : 23,
      "day" : 17, "month" : 9, "year" : 2015,
      "full_date" : ISODate("2015-09-17T23:39:11Z") },
    "location_dimension" : { "X" : 32, "Y" : 79 },
    "words" : [ { "tf" : 1, "count" : 1, "word" : "amanda" },
                { "tf" : 1, "count" : 1, "word" : "car" },
                { "tf" : 1, "count" : 1, "word" : "headache" }
     ] }
```

Figure 6: Sample $T^2K^2D^2$ DODBMS Document

Queries. To compute the top-k keywords using $T^2K^2D^2$ implementation and to compute the top-k documents, regardless of the schema, only the MR implementation is used. Also, the benchmark tests are implemented in the DODBMS native JavaScript language.

5. Experiments

5.1. Experimental Conditions

All tests run on an IBM System x3550 M4 with 64GB of RAM and an Intel(R) Xeon(R) CPU E5-2670 v2 @ 2.50GHz. The database systems we use are MongoDB v3.4.6, Oracle v12.1.0.2.0 and PostgreSQL v9.6.1. Initially, we planned to benchmark CouchDB 2.0.0, too. Unfortunately, we found no in-database solution to implement T^2K^2's queries, because of CouchDB's limited programming environment and MR framework. The code of all Oracle and PostgreSQL queries and MongoDB functions, together with benchmarking results, are available on Github[1].

MongoDB's performance is tested both in a single-instance and a distributed environment. The distributed environment is composed of a configuration node and 5 slave nodes; each slave becoming a worker in the MR algorithm. The database is distributed using horizontal scaling to test whether weighting scheme computing time decreases as the number of nodes increases. In MongoDB, horizontal scaling is achieved using sharding, and each individual data partition is refered as a shard [32].

The relational database implementations are tested only on a single-instance environment, because the Oracle Real Application Cluster that is needed to distribute Oracle databases is not free; and because PostgreSQL does not distribute natively, requiring third party software.

Besides indexes on primary keys, no other indexes are used in our tests. The sharding key in the MongoDB distributed environment is the tweet's unique identifier.

Eventually, query parameterization is provided in Table 1.

5.2. Dataset

Experiments are done on a 2 500 000 tweets corpus. The initial corpus is split into 5 different datasets that all keep an equal balance between the number of

[1] `https://github.com/cipriantruica/T2K2D2_Benchmark`

Table 1: Query parameter values

pGender	***pStartDate***	***pEndDate***	***pStartX***	***pEndX***	***pStartY***	***pEndY***	***pWords***
male | female	2015-09-17 00:00:00	2015-09-18 00:00:00	20	40	-100	100	think | today | friday

tweets for both genders, location and date. These datasets contain 500 000, 1 000 000, 1 500 000, 2 000 000 and 2 500 000 tweets, respectively. They allow scaling experiments and are associated to a scale factor (SF) parameter, where $SF \in \{0.5, 1, 1.5, 2, 2.5\}$, for conciseness sake.

5.2.1. Query Selectivity

Selectivity, i.e., the amount of retrieved data ($n(Q)$) w.r.t. the total amount of data available (N), depends on the number of attributes in the where and group by clauses. The selectivity formula used for a query Q is $S(Q) = 1 - \frac{n(Q)}{N}$.

For T^2K^2, all queries traverse the *Document*, *Vocabulary*, *Word* and *Gender* entities, and the *Write* and *Vocabulary* relationships, to extract lemmas and compute weights. All queries use the the *document_fact*, *Word_Dimnesion* and *Author_Dimension* for $T^2K^2D^2$.

All queries filter on gender, to determine the trending words for female and male users. Starting from $Q1$, subsequent queries $Q2$ to $Q4$ are built by decreasing selectivity (Table 2), regardless of the benchmark, i.e., T^2K^2 or $T^2K^2D^2$. Moreover, by adding a constraint on the location in $Q3$ and $Q4$, query complexity changes, too.

Selectivity for the top-k documents is decreased even more by adding a condition on the words attribute for all the queries. Moreover, for both benchmarks, selectivity remains the same (Table 3).

5.2.2. Query Complexity

Complexity relates to the number of traversals involved in the query. Query complexity depends on the number of relationship and entity traversals.

Table 2: Top-k keywords query selectivity

SF	***Q1*** **male**	***Q1*** **female**	***Q2*** **male**	***Q2*** **female**	***Q3*** **male**	***Q3*** **female**	***Q4*** **male**	***Q4*** **female**
0.5	0.336	0.337	0.517	0.517	0.556	0.558	0.677	0.679
1	0.342	0.342	0.662	0.662	0.562	0.565	0.774	0.775
1.5	0.347	0.346	0.736	0.736	0.569	0.572	0.823	0.824
2	0.351	0.350	0.783	0.783	0.574	0.575	0.855	0.856
2.5	0.353	0.354	0.815	0.815	0.579	0.580	0.876	0.877

Table 3: Top-k documents selectivity for 3 search terms

SF	***Q1*** **male**	***Q1*** **female**	***Q2*** **male**	***Q2*** **female**	***Q3*** **male**	***Q3*** **female**	***Q4*** **male**	***Q5*** **female**
0.5	0.9844	0.9848	0.9904	0.9905	0.9921	0.9926	0.9951	0.9954
1	0.9866	0.9868	0.9952	0.9953	0.9932	0.9936	0.9975	0.9977
1.5	0.9835	0.9837	0.9968	0.9968	0.9917	0.9920	0.9984	0.9985
2	0.9822	0.9824	0.9976	0.9976	0.9910	0.9913	0.9988	0.9988
2.5	0.9825	0.9827	0.9981	0.9981	0.9912	0.9915	0.9990	0.9991

Independently from any weighting schema, for T^2K^2 computing the number of tweets where a term appears and the total number of documents implies two aggregation queries that traverse either all entities ($Q3$, $Q4$) or all entities but *Geo_Location* ($Q1$, $Q2$), resulting in a number of 5 traversals for $Q3$ and $Q4$ and 4 traversals for $Q1$ and $Q2$. Moreover, some weighting schemes may require additional queries, e.g., Okapi BM25 involves tweet lengths. Table 4 presents query complexities w.r.t. weighting schemes. Query complexity is the same for T^2K^2 implementation regardless of computing the top-k keywords or top-k documents.

Table 5 present the $T^2K^2D^2$ queries complexity for top-k keywords.

The $T^2K^2D^2$ queries complexity for top-k documents is presented in Table 6.

Table 4: T^2K^2 query complexity

	Q1	*Q2*	*Q3*	*Q4*
TF-IDF	12	12	15	15
Okapi BM25	17	17	21	21

Table 5: $T^2K^2D^2$ query complexity for top-k keywords

	Q1	*Q2*	*Q3*	*Q4*
TF-IDF	3	5	5	7
Okapi BM25	4	6	6	8

Table 6: $T^2K^2D^2$ query complexity for top-k documents

	Q1	*Q2*	*Q3*	*Q4*
TF-IDF	5	8	8	11
Okapi BM25	6	9	9	12

5.3. Experimental Results

5.3.1. T^2K^2

Weighting schema comparison. Figure 7 presents a comparison by database system and weighting scheme of query response time for retrieving top-k keywords w.r.t. scale factor SF. This comparison uses only MongoDB's MR query implementations in single-instance and distributed environments, because there is no AP implementation for queries that use Okapi BM25.

In MongoDB, computing TF-IDF is faster than computing Okapi BM25 for all queries, regardless of SF (Figure 7a). The same results are observed in the distributed environment (Figure 7b). The biggest difference in computation time between TF-IDF and Okapi BM25 is obtained for $Q1$, while the smallest is obtained for $Q4$ because of its higher selectivity.

The execution times of $Q2$ and $Q4$ on Oracle and PostgreSQL are almost constant for all tested scenarios, regardless of weighting scheme or SF, because selectivity is constant in these queries. TF-IDF computing time is almost a third

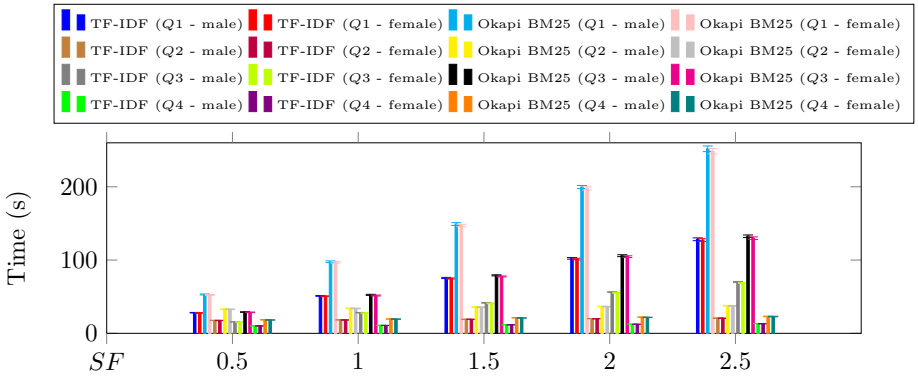
of the computing time for Okapi BM25's for $Q4$ in Oracle (Figure 7c) due to the complexity of the query itself and of the formula for computing Okapi BM25 w.r.t. TF-IDF. In PostgreSQL, the biggest performance difference between TF-IDF and Okapi BM25 is obtained for $Q1$ (Figure 7d) due to the large number of rows returned when joining all tables.

Figure 8 presents a comparison by database system and weighting scheme of query response time for retrieving top-k documents w.r.t. scale factor SF.

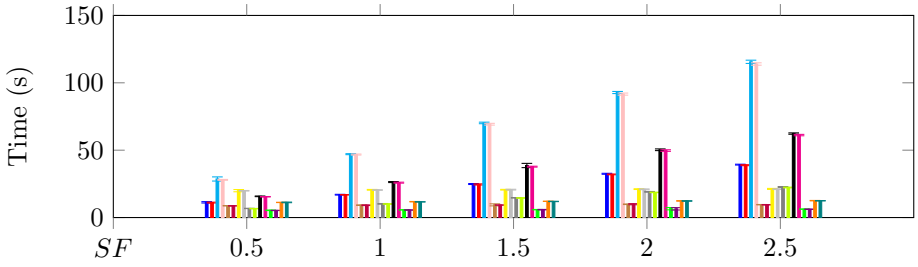
In MongoDB, computing TF-IDF is faster than computing Okapi BM25 for all queries, regardless of SF (Figure 8a). The same results are observed in the distributed environment (Figure 8b). In this case the performance of MongoDB is stable, regardless of the environment (single node or distributed), the performance curve increases linearly (Figures 8a and 8b). Such results are an expected consequence of the variation of query selectivity and complexity, as seletivity decreases and complexity increases from $Q1$ to $Q2$. Furthermore, as SF increases, so does seletivity in all queries.

The execution times of $Q1$ and $Q3$ for Oracle are getting worse when SF increases, since selectivity increases with the number of documents. In Oracle, the execution time for computing TF-IDF is half of the execution time for computing Okapi BM25 (Figure 8c) regardless of SF. This is a direct consequence of query complexity. The complexity of queries that use the TF-IDF weighting is indeed lower than those that use Okapi BM25 (Table 4). Moreover, in Oracle, the execution time is unstable as the standard deviation fluctuates w.r.t SF due to automatic cache cleaning when query results are too big to be stored in the database's internal buffers.

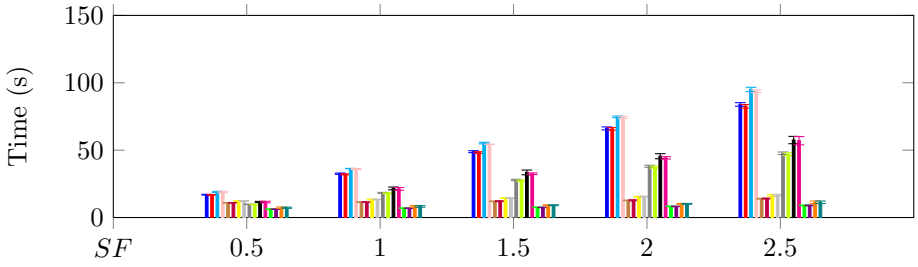
In PostgreSQL, the biggest performance difference between TF-IDF and Okapi BM25 is obtained for all the queries as the SF increases (Figure 8d). Moreover, the performance time for computing Okapi BM25 is smaller in all test scenarios than when computing TF-IDF. This is unexpected, as query complexity for computing the top-k documents with TF-IDF is smaller then when using Okapi BM25. We hypothesize that such results are a direct consequence of how the DBMS builds query execution plans, with Oracle being the most



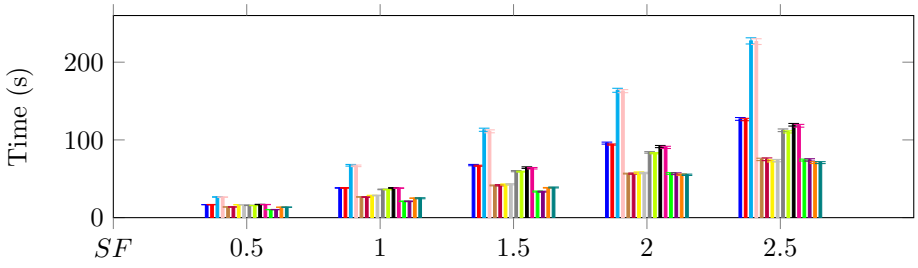
(a) MongoDB MR: TF-IDF vs. Okapi BM25



(b) MongoDB MR Distributed: TF-IDF vs. Okapi BM25



(c) Oracle: TF-IDF vs. Okapi BM25



(d) PostgreSQL: TF-IDF vs. Okapi BM25

Figure 7: Top-k keywords TF-IDF vs. Okapi BM25 comparison

proficient.

Database Implementation Comparison. This set of experiments presents a time performance comparison of database implementations w.r.t. SF and weighting schemes for the top-k keywords queries. We also test the performance of the distributed MongoDB environment, labeled *Dist*. Moreover, MongoDB queries use both the AP and MR implementations for TF-IDF, but only the MR implementation for Okapi BM25.

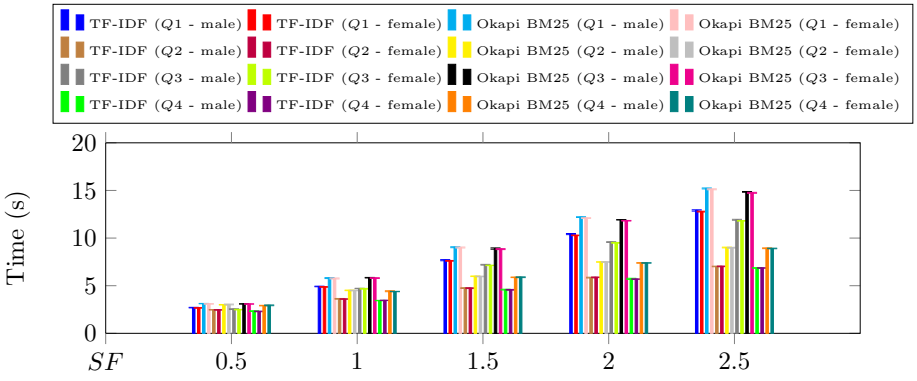
Oracle outperforms PostgreSQL in the single-instance environment for all test cases. Query execution time in Oracle is indeed half of the time than in PostgreSQL in the worse case (Figures 9 and 10). Our hypothesis is that Oracle performs some optimization while PostgreSQL does not, but we could not verity it.

In MongoDB, using AP instead of MR for computing TF-IDF yields better execution time. In the worst case, AP execution time is indeed half of the time than MR's (Figure 9), because MongoDB is optimized to use AP instead of MR. Thus, AP provides an alternative to MR and may be preferred sfor aggregation tasks, where the complexity of MP may be unwarranted. However, AP bears some limitations on value types, result size [33] and memory use [34].

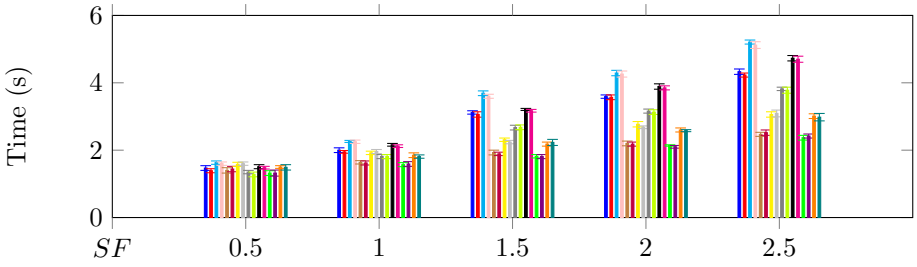
In the distributed environment, computing TF-IDF using AP takes almost half the time than when using MR for all queries, too, regardless of SF (Figure 9), because AP is also optimized to perform better in a distributed environment than MR, though in both cases, the workload is distributed among all shards.

Figure 11 presents a time performance comparison of database implementations w.r.t. SF and weighting schemes for the top-k documents queries that use the TF-IDF weighting schema, while Figure 12 for the queries that use the Okapi BM25 weighting schema.

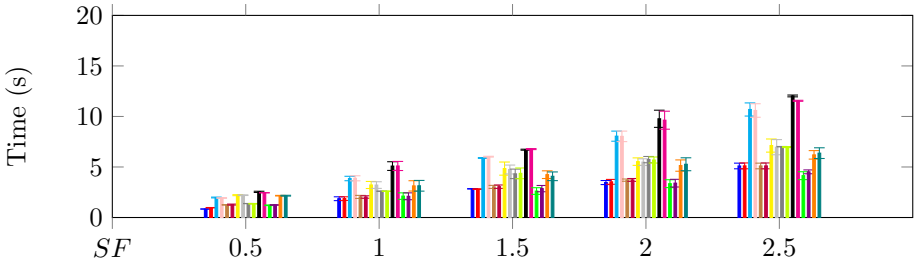
PostgreSQL has the worst execution time regardless of the weighting schema or SF. Our hypothesis is that PostgreSQL does not optimize the query execution plan due to the absence of indexes.



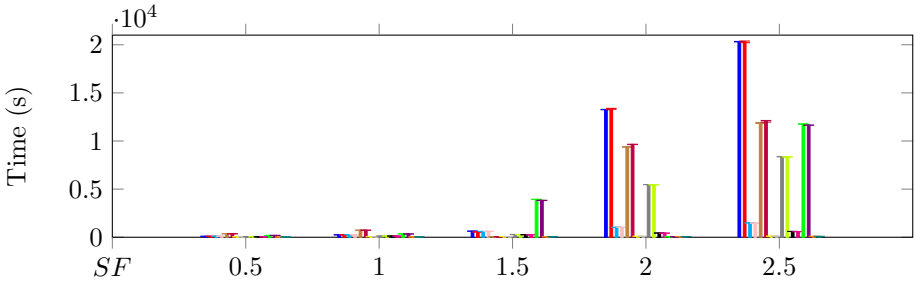
(a) MongoDB MR: TF-IDF vs. Okapi BM25



(b) MongoDB MR Distributed: TF-IDF vs. Okapi BM25

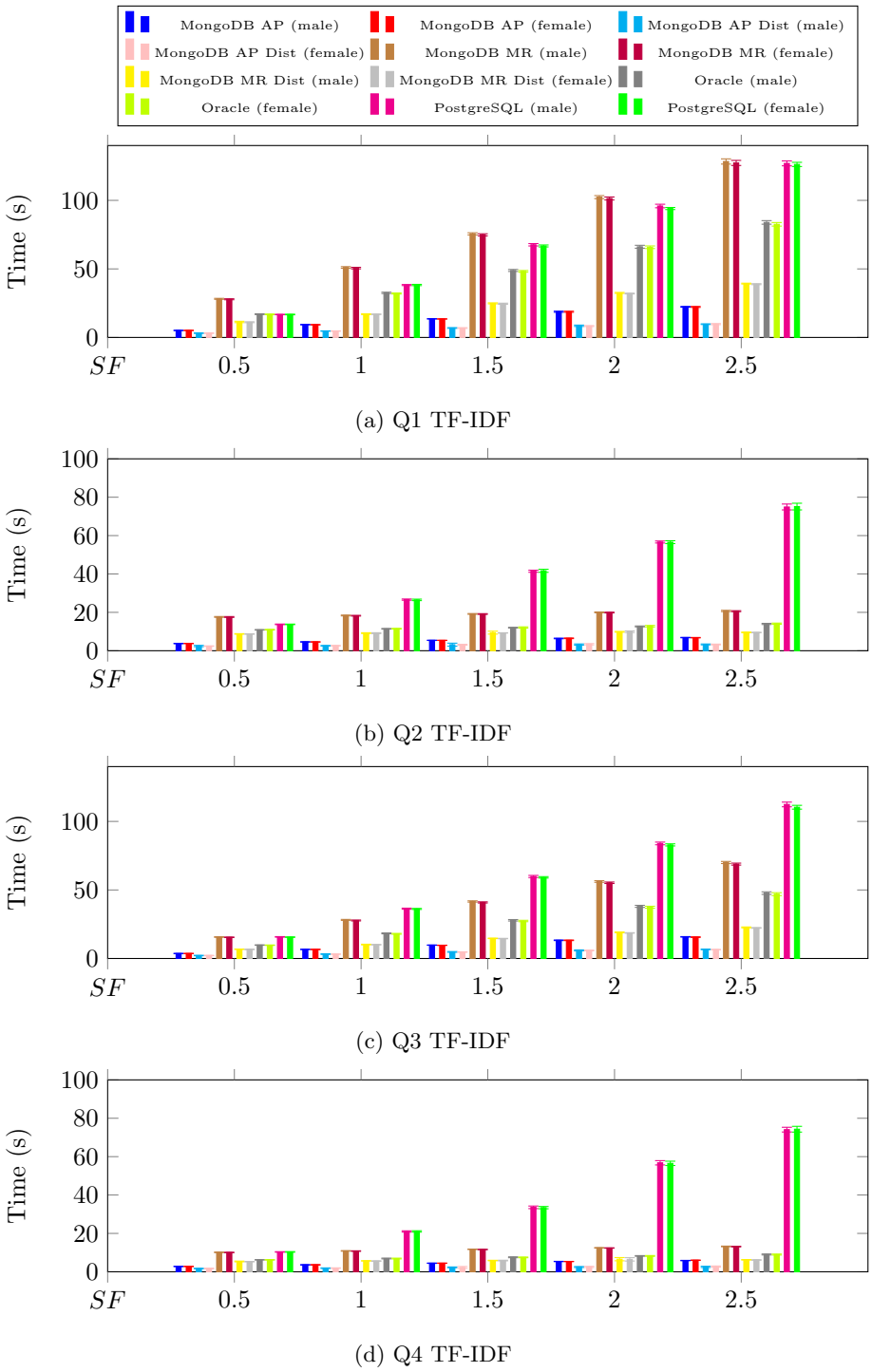


(c) Oracle: TF-IDF vs. Okapi BM25



(d) PostgreSQL: TF-IDF vs. Okapi BM25

Figure 8: Top-k documents TF-IDF vs. Okapi BM25 comparison



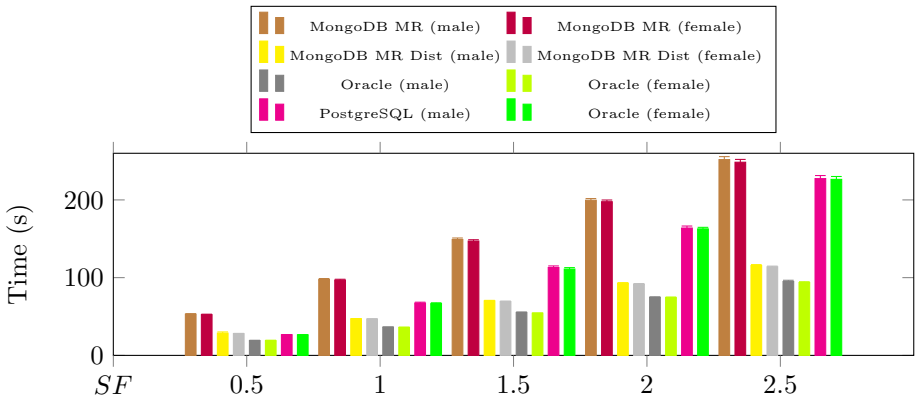
(a) Q1 TF-IDF

(b) Q2 TF-IDF

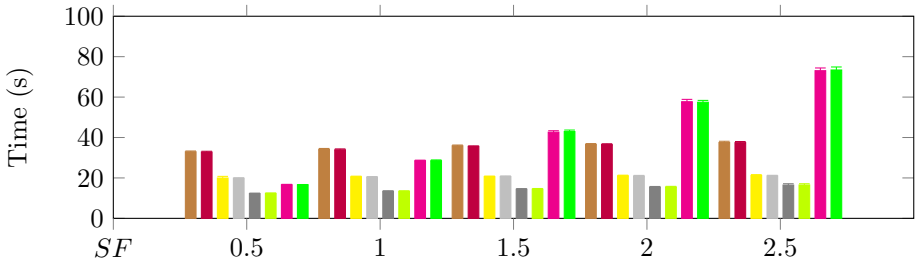
(c) Q3 TF-IDF

(d) Q4 TF-IDF

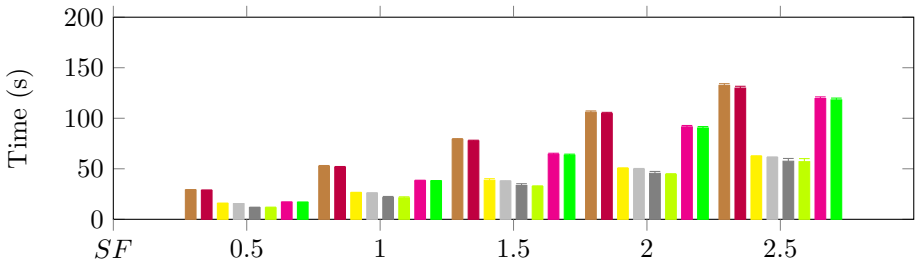
Figure 9: Top-k keywords query comparison: TF-IDF



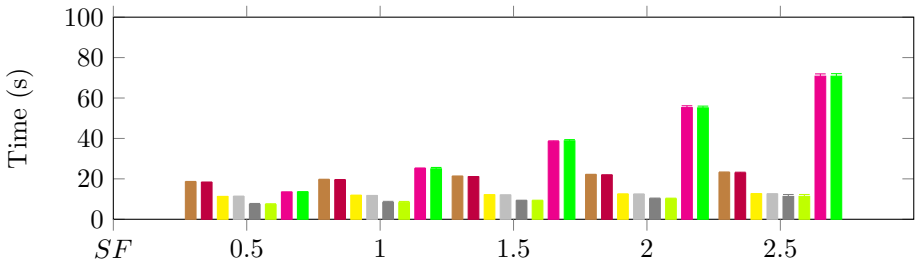
(a) Q1 Okapi BM25



(b) Q2 Okapi BM25



(c) Q3 Okapi BM25



(d) Q4 Okapi BM25

Figure 10: Top-k keywords query comparison: Okapi BM25

Oracle outperforms MongoDB in a single instance environment for both weighting schemas. Moreover, Oracle is optimized to make the best use of system resources in a single instance environment [35].

MongoDB in a distributed environment has the best performance time regardless of the weighting schema. Query execution performance in MongoDB is improved because the workload is distributed between shards and results are aggregated after each shard finishes a task.

5.3.2. $T^2K^2D^2$

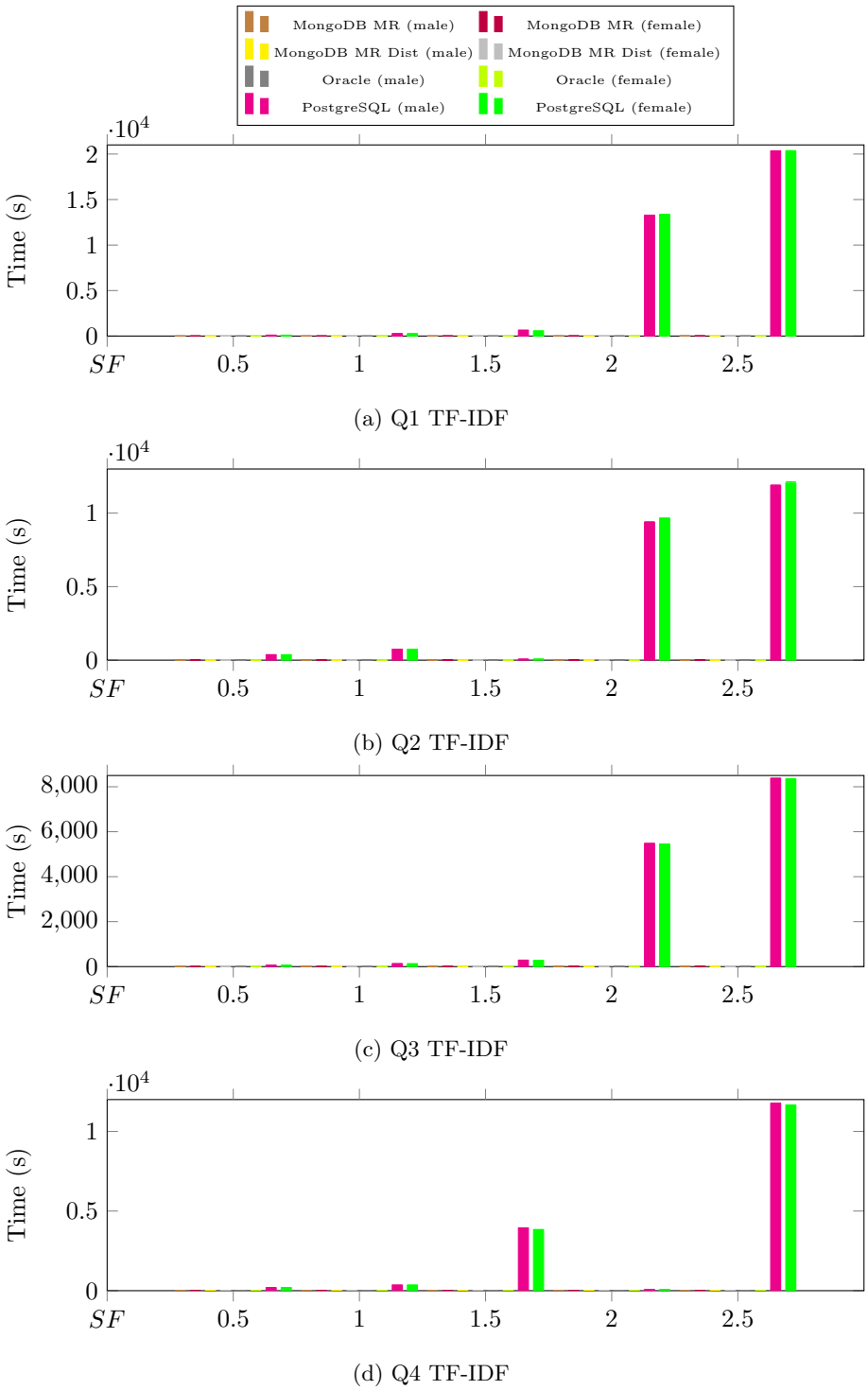
Weighting schema comparison. Figure 13 present a comparison by database system and weighting schema of query response time for retrieving the top-k keywords w.r.t. scale factor SF for the multidimensional implementation, while Figure 14 for retrieving top-k documents.

For MongoDB the same patterns in performance appear as for the T^2K^2 test scenarios. The time performance is almost non-existent between the queries run on T^2K^2 (Figures 7a and 7b) and the ones executed on $T^2K^2D^2$ (Figures 13a and 13b). This pattern emerges because MongoDB is a schemaless database and the data modeling does not really influence performance. The same results are optained for the top-k documents (Figures 14a and 14b).

Using a multidimensional model improves performance with both Oracle and PostgreSQL. Using such a model, the indexes used for primary and foreign keys to associate entities make query execution plan optimization easy, because a join does not have to pass through multiple tables as in the normalized model.

For Oracle, the execution time of queries $Q1$ and $Q3$ that use TF-IDF is half the time of the execution time of the same queries that use Okapi BM25 for top-k keywords (Figure 13c) w.r.t. SF. For top-k documents, the execution time for queries $Q1$ and $Q2$ is almost the same regardless of weight. It should be noted that Oracle's query execution time for top-k documents is unstable, as the standard deviation fluctuates. We think this fluctuation might be a direct consequence of cache and buffer cleaning.

PostgreSQL query execution time is also improved when using a multidi-



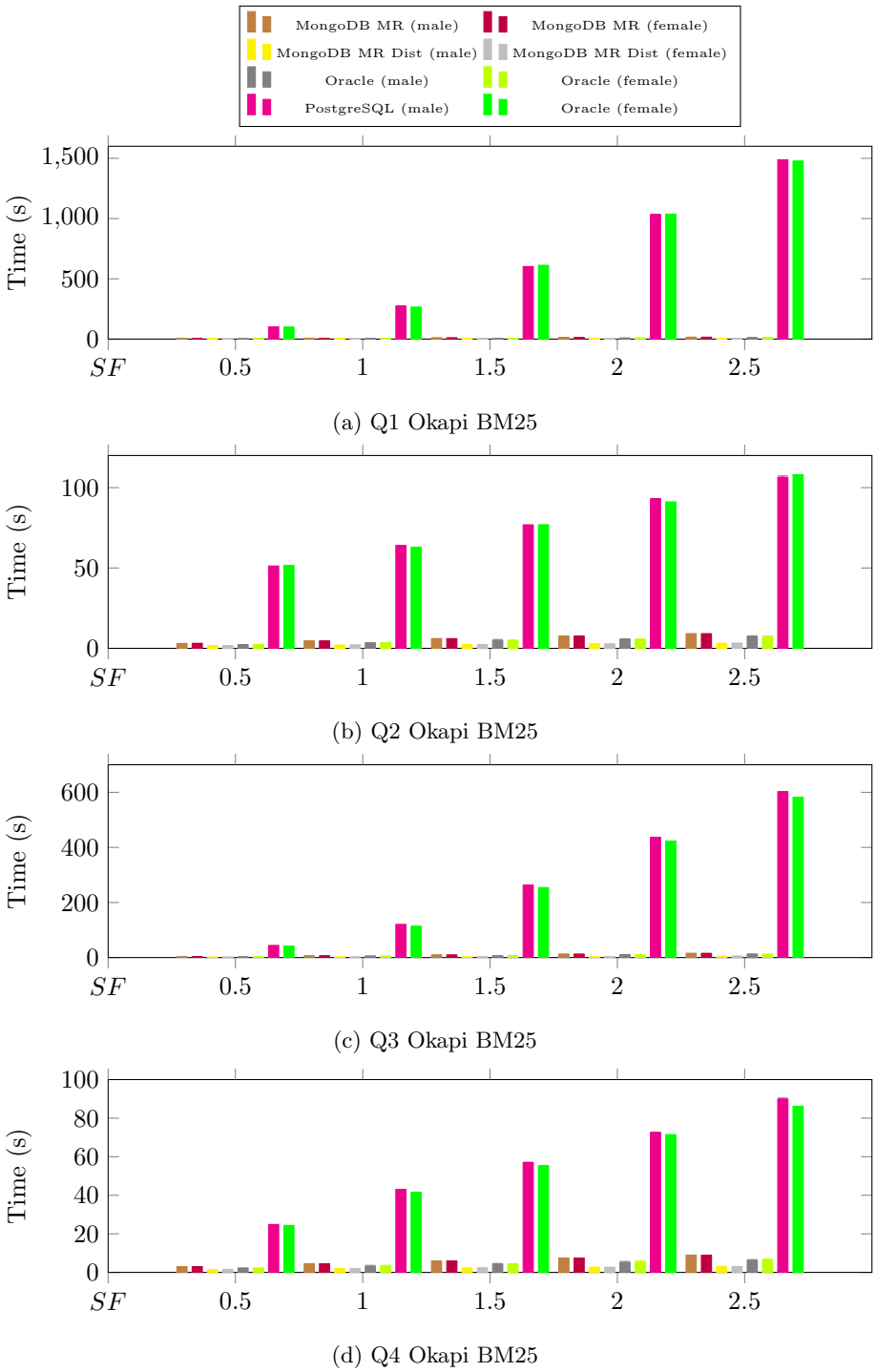
(a) Q1 TF-IDF

(b) Q2 TF-IDF

(c) Q3 TF-IDF

(d) Q4 TF-IDF

Figure 11: Top-k documents query comparison: TF-IDF



(a) Q1 Okapi BM25

(b) Q2 Okapi BM25

(c) Q3 Okapi BM25

(d) Q4 Okapi BM25

Figure 12: Top-k documents query comparison: Okapi BM25

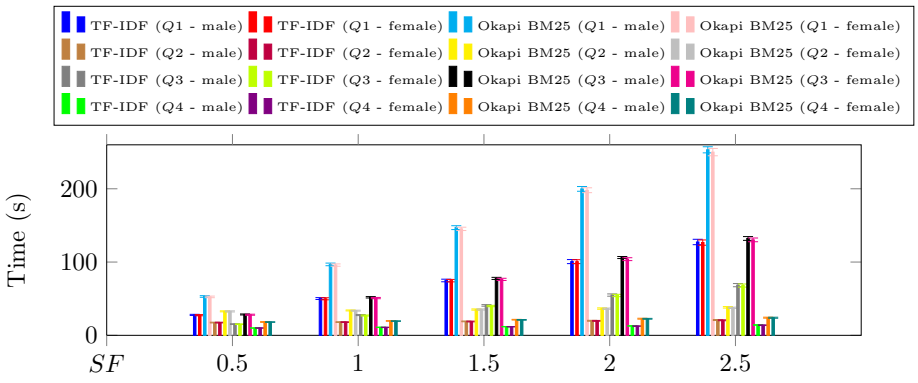
mensional model. The query execution time are stable with a low standard deviation. For top-k documents, $Q1$ has the worst execution time. Moreover, the execution time between the queries that compute the weights with TF-IDF and the ones that compute the weights with Okapi BM25 has a factor of $\sim$1.2 regardless of SF. For top-k documents, $Q1$ execution time is the same regardless of weighting schema. The execution time of $Q3$ with Okapi BM25 is half the time of the execution of the same query with TF-IDF.

Database implementation comparison.. Oracle has the best execution time regardless of weighting schema in a single instance for both top-k keywords and top-k documents (Figures 15, 16, 17 and 16). Besides the resource allocation optimization used by Oracle in a single instance environment, query complexity also plays an important role in query execution performance, as the number of joins decreases when using a multidimensional model.

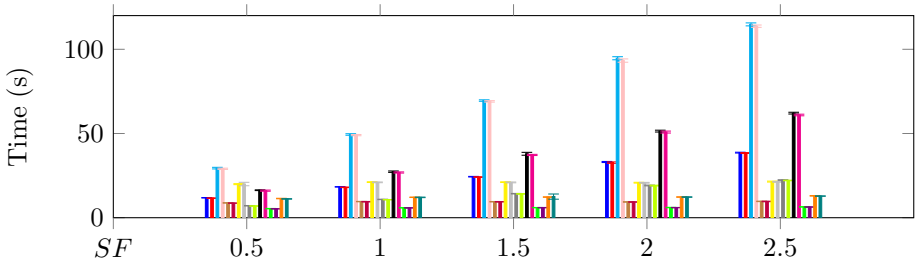
MongoDB in a distributed environment has the overall best performance time for the queries that use the TF-IDF weighting schema, but it is outperformed by both Oracle and PostgreSQL when using Okapi BM25. Because MongoDB uses a schemaless flexible model, the design and implementation of the schema indeed do not really affect query performance results. Furthermore, query complexity, which depends a lot on the weighting scheme used (Tables 5 and 6), plays an important role. In Oracle and PostgreSQL, Okapi BM25 weights are computed through nested queries, but in MongoDB, separate queries are needed. To improve MongoDB query performance, an indexing and sharding key analysis should be made, but such a extensive study is outside the scope of this paper.

MongoDB in a single environment has the overall worst performance between the tested databases for top-k keywords (Figures 15 and 16), while PostgreSQL for top-k documents (Figures 17 and 16).

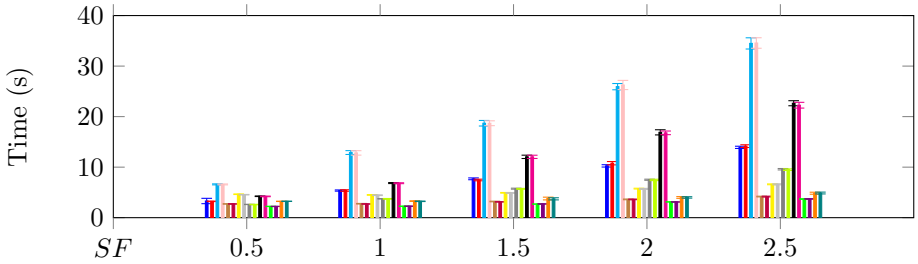
Using a multidimensional model significantly improves query performance in PostgreSQL, because query complexity is decreased. The multidimensional model used in $T^2K^2D^2$ indeed only features one-to-many relationships at the



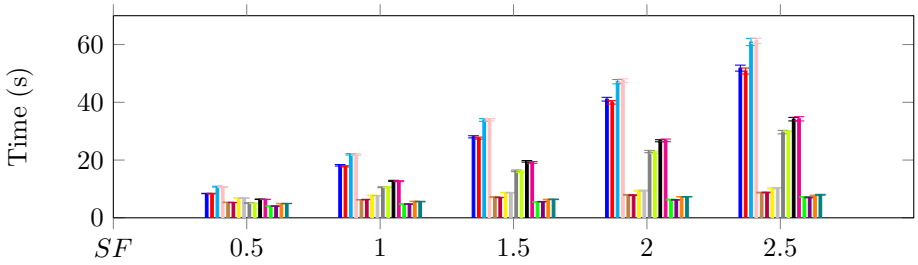
(a) MongoDB MR OLAP: TF-IDF vs. Okapi BM25



(b) MongoDB MR Distributed OLAP: TF-IDF vs. Okapi BM25

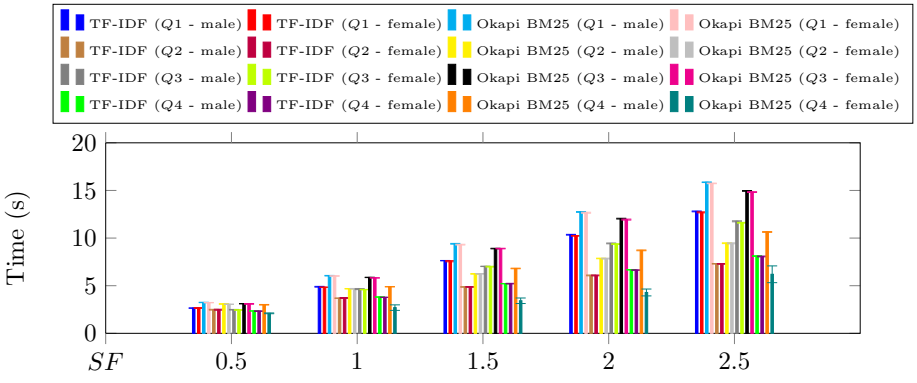


(c) Oracle OLAP: TF-IDF vs. Okapi BM25

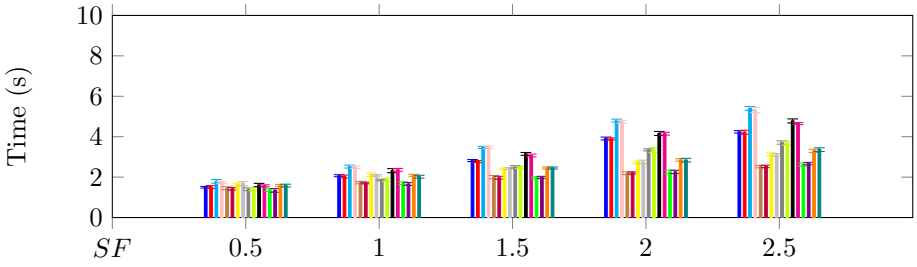


(d) PostgreSQL OLAP: TF-IDF vs. Okapi BM25

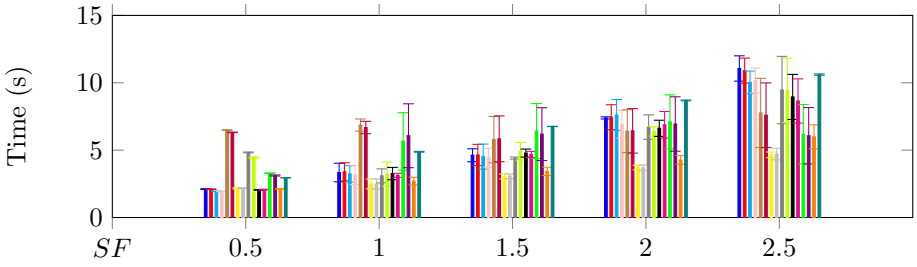
Figure 13: OLAP top-k keywords TF-IDF vs. Okapi BM25 comparison



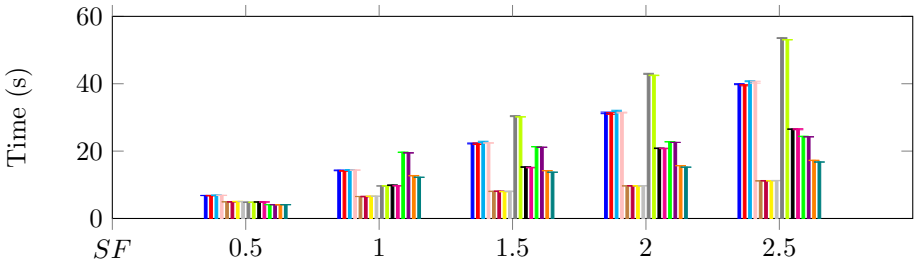
(a) MongoDB MR OLAP: TF-IDF vs. Okapi BM25



(b) MongoDB MR Distributed OLAP: TF-IDF vs. Okapi BM25



(c) Oracle OLAP: TF-IDF vs. Okapi BM25



(d) PostgreSQL OLAP: TF-IDF vs. Okapi BM25

Figure 14: OLAP top-k documents TF-IDF vs. Okapi BM25 comparison

conceptual level, and thus does not include any bridge tables at the logical level for expressing many-to-many relationships, as in T^2K^2. Therefore, the number of joins in queries decreases and so does query complexity.

6. Conclusion

Jim Gray defined four primary criteria to specify a "good" benchmark [36]:

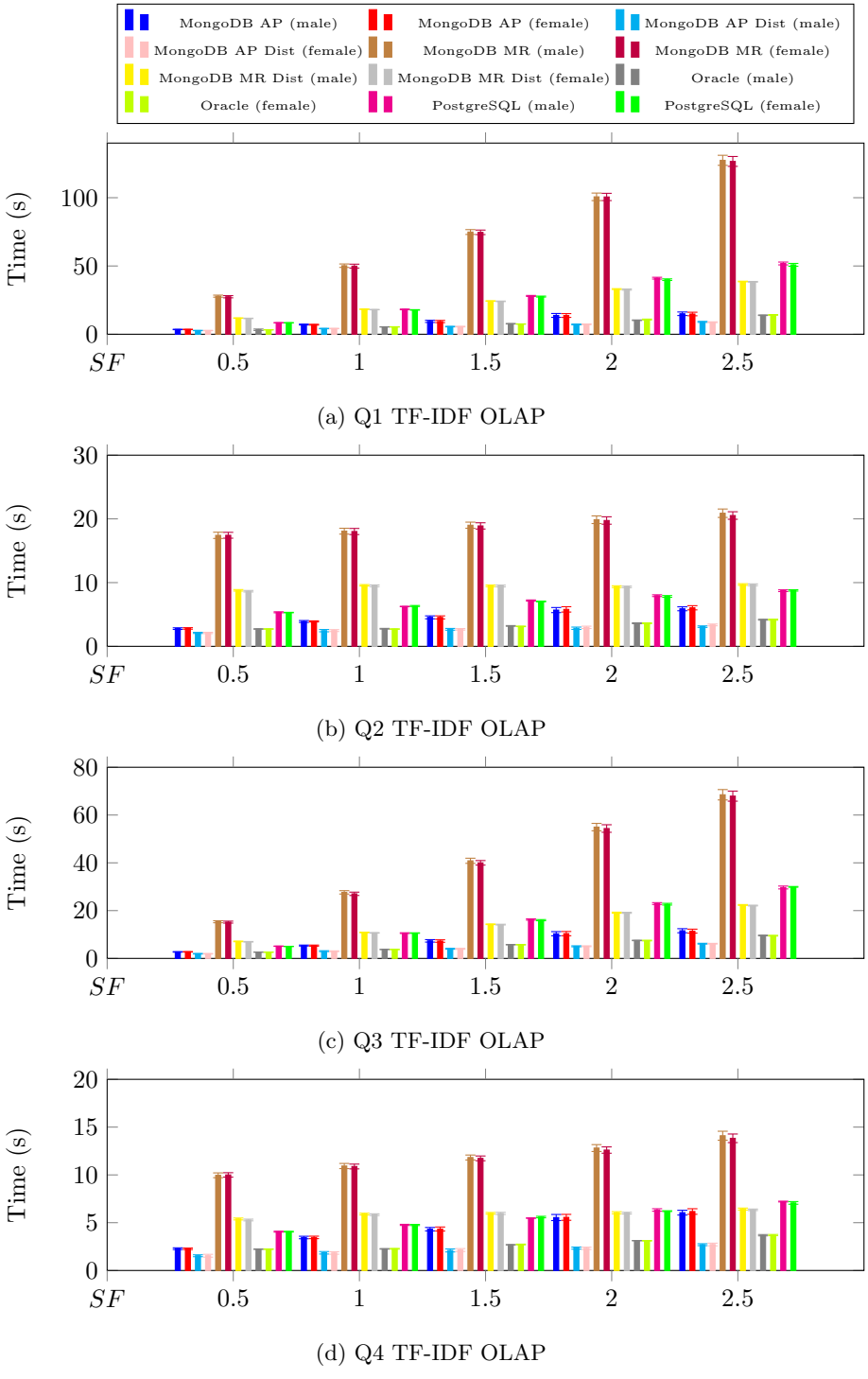
- *Relevance:* The benchmark must deal with aspects of performance that appeal to the largest number of users. Considering the wide usage of top-k keywords and documents queries in various text analytics tasks, we think T^2K^2 and $T^2K^2D^2$ fulfills this criterion. We also show in Section 5 that our benchmark achieves what it is designed for.

- *Portability:* The benchmark must be reusable to test the performances of different database systems. We successfully used T^2K^2 and $T^2K^2D^2$ to compare two types of database systems, namely relational and document-oriented systems.

- *Simplicity:* The benchmark must be feasible and must not require too many resources. We designed T^2K^2 and $T^2K^2D^2$ with this criterion in mind (Section 3), which is particularly important for reproducibility. We notably made up parameters that are easy to setup.

- *Scalability:* The benchmark must adapt to small or large computer architectures. By introducing scale factor SF, we allow users to simply parameterize T^2K^2 and $T^2K^2D^2$ and achieve some scaling, though it could be pushed further in terms of data volume.

Regarding experimental results, for T^2K^2's top-k keywords the best TF-IDF computing time in single-instance mode is obtained by MongoDB using PA. Oracle registers the best execution time among relational databases. Moreover, the best solution for computing TF-IDF is to use MongoDB in a distributed environment because increasing the number of nodes expectingly helps



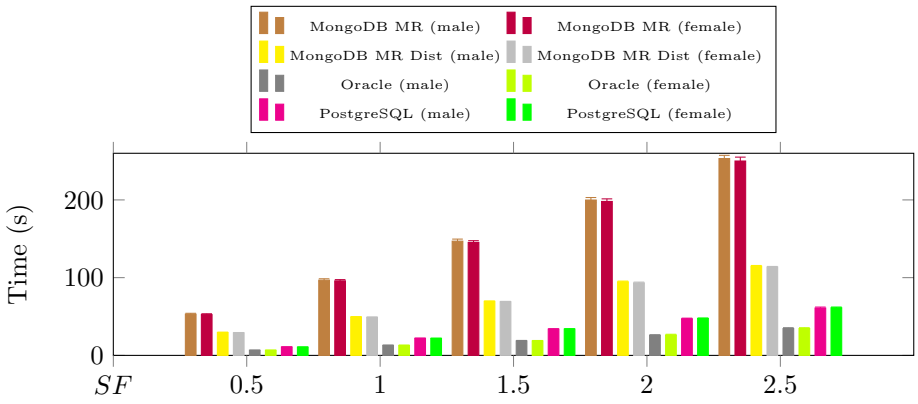
(a) Q1 TF-IDF OLAP

(b) Q2 TF-IDF OLAP

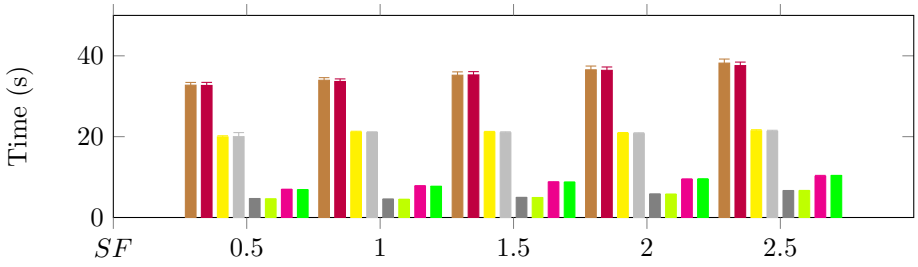
(c) Q3 TF-IDF OLAP

(d) Q4 TF-IDF OLAP

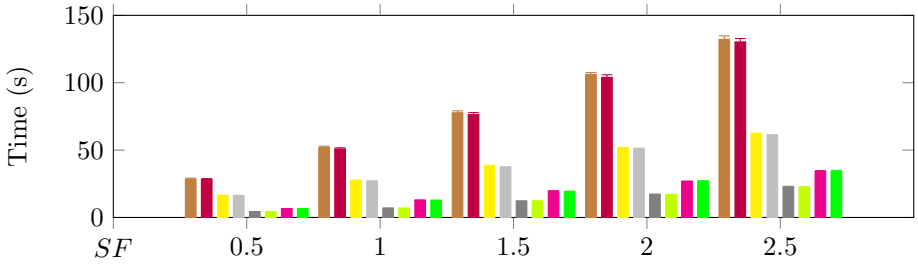
Figure 15: Top-k keywords OLAP query comparison: TF-IDF



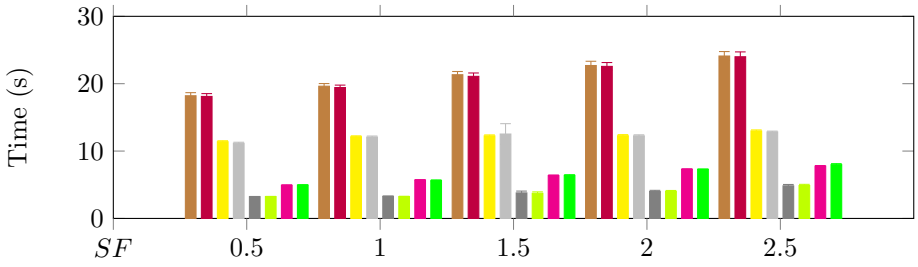
(a) Q1 Okapi BM25 OLAP



(b) Q2 Okapi BM25 OLAP

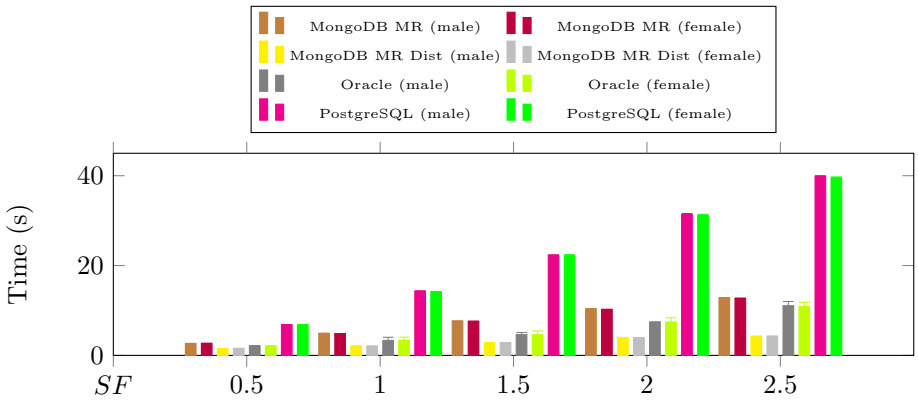


(c) Q3 Okapi BM 25 OLAP

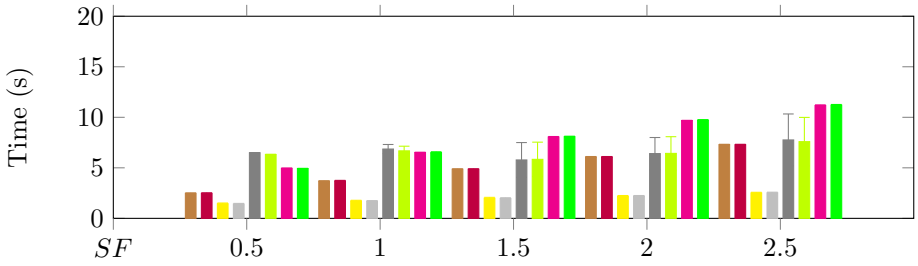


(d) Q4 Okapi BM25 OLAP

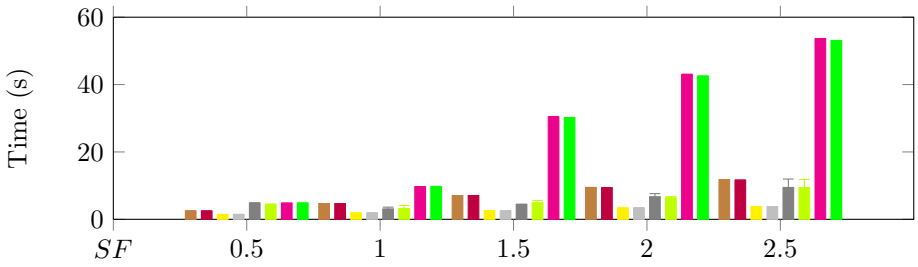
Figure 16: Top-k keywords OLAP query comparison: Okapi BM25



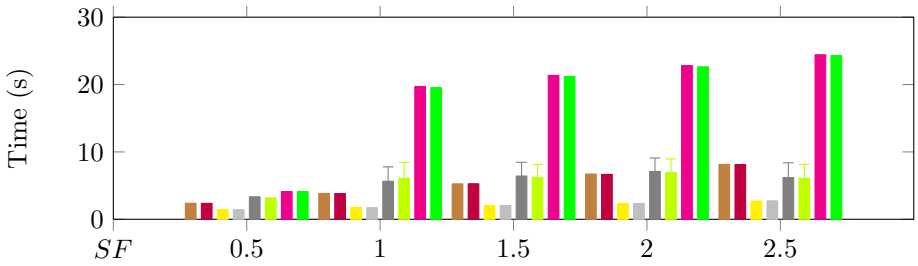
(a) Q1 TF-IDF OLAP



(b) Q2 TF-IDF OLAP

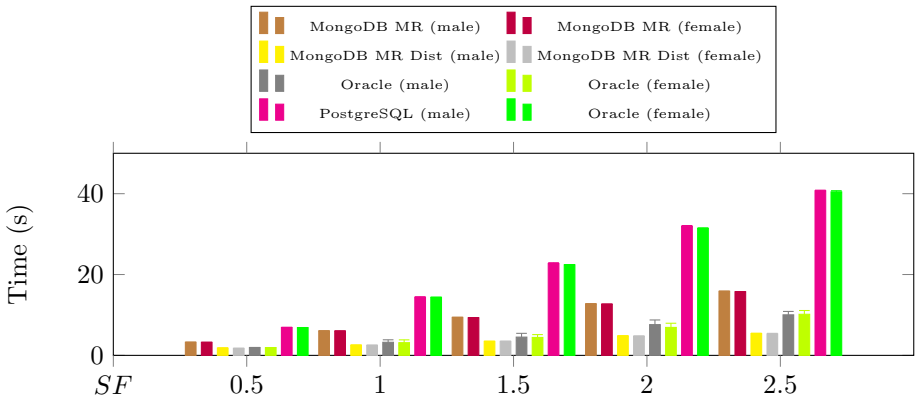


(c) Q3 TF-IDF OLAP

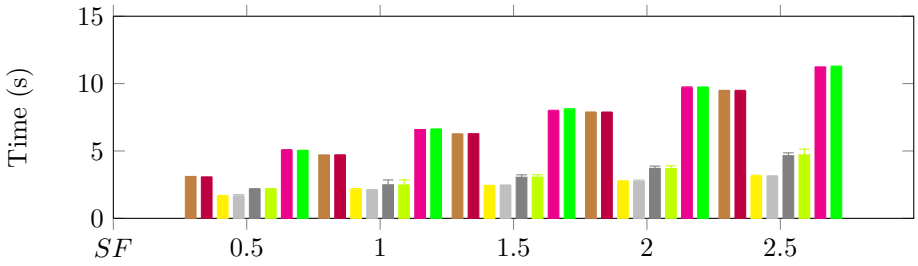


(d) Q4 TF-IDF OLAP

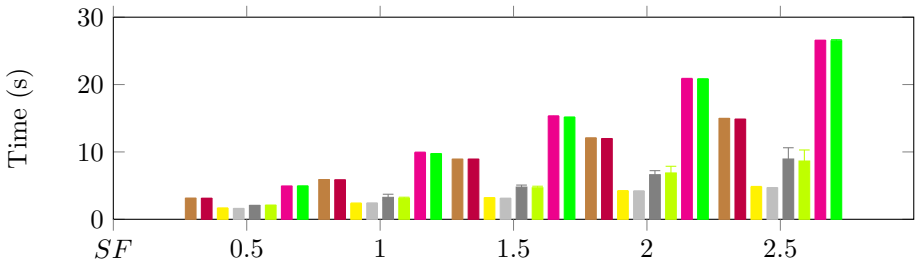
Figure 17: Top-k documents OLAP query comparison: TF-IDF



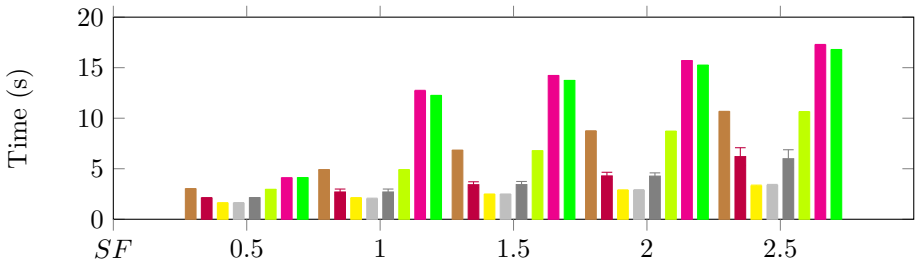
(a) Q1 Okapi BM25 OLAP



(b) Q2 Okapi BM25 OLAP



(c) Q3 Okapi BM 25 OLAP



(d) Q4 Okapi BM25 OLAP

Figure 18: Top-k documents OLAP query comparison: Okapi BM25

decreasing query execution time. Oracle outperforms the other database systems when computing Okapi BM25. Moreover, MongoDB in single-instance mode has the worst execution time for computing Okapi BM25 with queries $Q1$ and $Q3$, while PostgreSQL has the worst execution time for queries $Q2$ and $Q4$. This is unexpected, because the trends should have remained the same for these two database systems. When computing the top-k documents with T^2K^2, the best performance time is obtained by MongoDB in a distributed environment and by Oracle in a single instance regardless of the schema, while PostgreSQL has the worst performance.

When using a multidimensional model with $T^2K^2D^2$, MongoDB registers the same performance or both top-k keywords and top-k documents as with T^2K^2. Oracle in a single instance has the overall best performance for both tasks. For top-k keywords MongoDB in a single instance has the worst performance while for top-k documents, PostgreSQL has the worst performance.

In future work, we plan to expand T^2K^2 and $T^2K^2D^2$ dataset significantly to aim at big data-scale volume (scalability). We also plan to adapt our benchmarks so that they run in the Hadoop and Spark environments. Moreover, we intend to further our proof of concept and validation efforts by benchmarking other NoSQL database systems and gain insight regarding their capabilities and shortcomings (relevance). Eventually, we considered in this paper that TF-IDF and Okapi BM25 were enough for machine learning tasks. Yet, the next version of our benchmarks should include other weighting schemes, such as KL-divergence [37], also to improve their relevance.

References

[1] F. Ravat, O. Teste, R. Tournier, G. Zurfluh, Top_keyword: an aggregation function for textual document olap, in: 10th International Conference on Data Warehousing and Knowledge Discovery (DaWaK), 2008, pp. 55–64. doi:10.1007/978-3-540-85836-2_6.

[2] S. Bringay, N. Béchet, F. Bouillot, P. Poncelet, M. Roche, M. Teisseire,

Towards an on-line analysis of tweets processing, in: International Conference on Database and Expert Systems Applications (DEXA), 2011, pp. 154–161. doi:10.1007/978-3-642-23091-2_15.

[3] A. Guille, C. Favre, Event detection, tracking, and visualization in twitter: a mention-anomaly-based approach, Social Network Analysis and Mining 5 (1) (2015) 18. doi:10.1007/s13278-015-0258-0.

[4] R. Alghamdi, K. Alfalqi, A survey of topic modeling in text mining, International Journal of Advanced Computer Science and Applications 6 (1) (2015) 147–153. doi:10.14569/IJACSA.2015.060121.

[5] C. C. Aggarwal, C. Zhai, A Survey of Text Clustering Algorithms, in: Mining Text Data, Springer, 2012, pp. 77–128. doi:10.1007/978-1-4614-3223-4_4.

[6] S. Huang, J. Huang, J. Dai, T. Xie, B. Huang, The HiBench benchmark suite: Characterization of the MapReduce-based data analysis, in: Workshops Proceedings of the 26th International Conference on Data Engineering (ICDE 2010), 2010, pp. 41–51. doi:10.1109/ICDEW.2010.5452747.

[7] L. Wang, J. Zhan, C. Luo, Y. Zhu, Q. Yang, Y. He, W. Gao, Z. Jia, Y. Shi, S. Zhang, C. Zheng, G. Lu, K. Zhan, X. Li, B. Qiu, BigDataBench: A big data benchmark suite from internet services, in: 20th IEEE International Symposium on High Performance Computer Architecture (HPCA 2014), 2014, pp. 488–499. doi:10.1109/HPCA.2014.6835958.

[8] Transaction Processing Performance Council (TPC), TPC Express Benchmark HS Standard Specification Version 1.4.2 (Nov. 2016).
URL http://www.tpc.org

[9] A. E. Gattiker, F. H. Gebara, H. P. Hofstee, J. D. Hayes, A. Hylick, Big data text-oriented benchmark creation for Hadoop, IBM Journal of Research and Development 57 (3/4) (2013) 10:1–10:6. doi:10.1147/JRD.2013.2240732.

[10] J. Ferrarons, M. Adhana, C. Colmenares, S. Pietrowska, F. Bentayeb, J. Darmont, Primeball: a parallel processing framework benchmark for big data applications in the cloud, in: 5th TPC Technology Conference on Performance Evaluation and Benchmarking (TPCTC 2013), Vol. 839 of LNCS1, 2014, pp. 109–124. doi:10.1007/978-3-319-04936-6_8.

[11] C.-O. Truică, J. Darmont, T^2k^2: The twitter top-k keywords benchmark, in: 21st European Conference on Advances in Databases and Information Systems (ADBIS 17), CCIS, 2017, pp. 21–28. doi:10.1007/978-3-319-67162-8_3.

[12] C.-O. Truică, A. Guille, M. Gauthier, Cats: Collection and analysis of tweets made simple, in: ACM Conference on Computer Supported Cooperative Work and Social Computing (CSCW2016), ACM, 2016, pp. 41–44. doi:10.1145/2818052.2874320.

[13] C.-O. Truică, J. Darmont, J. Velcine, A scalable document-based architecture for text analysis, in: International Conference on Advanced Data Mining and Applications (ADMA2016), Vol. 10086, Springer, 2016, pp. 481–494. doi:10.1007/978-3-319-49586-6_33.

[14] J. Lin, M. Crane, A. Trotman, J. Callan, I. Chattopadhyaya, J. Foley, G. Ingersoll, C. Macdonald, S. Vigna, Toward reproducible baselines: The open-source ir reproducibility challenge, in: Advances in Information Retrieval, Springer International Publishing, 2016, pp. 408–420. doi:10.1007/978-3-319-30671-1_30.

[15] M. Crane, J. S. Culpepper, J. Lin, J. Mackenzie, A. Trotman, A comparison of document-at-a-time and score-at-a-time query evaluation, in: Proceedings of the Tenth ACM International Conference on Web Search and Data Mining, ACM, 2017, pp. 201–210. doi:10.1145/3018661.3018726.

[16] P. Bellot, A. Doucet, S. Geva, S. Gurajada, J. Kamps, G. Kazai, M. Koolen, A. Mishra, V. Moriceau, J. Mothe, M. Preminger, E. SanJuan, R. Schenkel,

X. Tannier, M. Theobald, M. Trappett, A. Trotman, M. Sanderson, F. Scholer, Q. Wang, Report on inex 2013, SIGIR Forum 47 (2) (2013) 21–32. doi:10.1145/2568388.2568393.

[17] A. Bifet, E. Frank, Sentiment knowledge discovery in twitter streaming data, in: Discovery Science, Springer Berlin Heidelberg, 2010, pp. 1–15. doi:10.1007/978-3-642-16184-1_1.

[18] A. J. Reagan, B. F. Tivnan, J. R. Williams, C. M. Danforth, P. S. Dodds, Benchmarking sentiment analysis methods for large-scale texts: A case for using continuum-scored words and word shift graphs, CoRR abs/1512.00531.

[19] J. O'Shea, Z. Bandar, K. A. Crockett, D. McLean, Benchmarking short text semantic similarity, International Journal of Intelligent Information and Database Systems 4 (2) (2010) 103–120. doi:10.1504/IJIIDS.2010.032437.

[20] D. D. Lewis, Y. Yang, T. G. Rose, F. Li, Rcv1: A new benchmark collection for text categorization research, Journal of Machine Learning Research 5 (2004) 361–397.
URL http://www.jmlr.org/papers/v5/lewis04a.html

[21] I. Partalas, A. Kosmopoulos, N. Baskiotis, T. Artières, G. Paliouras, É. Gaussier, I. Androutsopoulos, M.-R. Amini, P. Gallinari, Lshtc: A benchmark for large-scale text classification, CoRR.
URL http://arxiv.org/abs/1503.08581

[22] D. Kılınç, A. Özçift, F. Bozyigit, P. Yildirim, F. Yücalar, E. Borandag, Ttc-3600: A new benchmark dataset for turkish text categorization, Journal of Information Science 43 (2) (2017) 174–185. doi:10.1177/0165551515620551.

[23] L. Wang, X. Dong, X. Zhang, Y. Wang, T. Ju, G. Feng, Textgen: a realistic text data content generation method for modern storage system

benchmarks, Frontiers of Information Technology & Electronic Engineering 17 (10) (2016) 982–993. doi:10.1631/FITEE.1500332.

[24] J. Darmont, Data-Centric Benchmarking, Encyclopedia of Information Science and Technology (4th Edition), IGI Global, 2017, Ch. 154, pp. 1772–1782. doi:10.4018/978-1-5225-2255-3.ch154.

[25] E. Haddi, X. Liu, YongShi, The role of text pre-processing in sentiment analysis, Procedia Computer Science 17 (2013) 26–32. doi:10.1016/j.procs.2013.05.005.

[26] J. D. Cooper, M. D. Robinson, J. A. Slansky, N. D. Kiger, Literacy: Helping students construct meaning, Cengage Learning, 2014.

[27] G. Paltoglou, M. Thelwall, A study of information retrieval weighting schemes for sentiment analysis, in: 48th Annual Meeting of the Association for Computational Linguistics, 2010, pp. 1386–1395.
URL http://dl.acm.org/citation.cfm?id=1858681.1858822

[28] C. D. Manning, P. Raghavan, H. Schütze, Introduction to information retrieval, Cambridge University Press, 2008.

[29] K. Spärck Jones, S. Walker, S. E. Robertson, A probabilistic model of information retrieval: development and comparative experiments: Part 1, Information Processing & Management 36 (6) (2000) 779 – 808. doi:10.1016/S0306-4573(00)00015-7.

[30] K. Spärck Jones, S. Walker, S. E. Robertson, A probabilistic model of information retrieval: development and comparative experiments: Part 2, Information Processing & Management 36 (6) (2000) 809 – 840. doi:10.1016/S0306-4573(00)00016-9.

[31] M. Chevalier, M. El Malki, A. Kopliku, O. Teste, R. Tournier, Implementation of multidimensional databases with document-oriented nosql, in: Big Data Analytics and Knowledge Discovery (DaWaK2015), 2015, pp. 379–390. doi:10.1007/978-3-319-22729-0_29.

[32] L. Bonnet, A. Laurent, M. Sala, B. Laurent, N. Sicard, Reduce, you say: What nosql can do for data aggregation and bi in large repositories, in: International Workshop on Database and Expert Systems Applications, 2011, pp. 483–488. doi:10.1109/DEXA.2011.71.

[33] MongoDB, Inc., The MongoDB Manual v3.4 (2017).
URL https://docs.mongodb.com/manual

[34] K. Chodorow, MongoDB: The Definitive Guide: Powerful and Scalable Data Storage (2rd Edition), O'Reilly Media, Inc., 2013.

[35] R. Bhatiya, I. Chan, L. Ashdown, Oracle Database Performance Tuning Guide, 12c Release 1 (12.1), Oracle, Inc., 2017.

[36] J. Gray, The Benchmark Handbook for Database and Transaction Systems (2nd Edition), Morgan Kaufmann, 1993.

[37] F. Raiber, O. Kurland, Kullback-leibler divergence revisited, in: Proceedings of the ACM SIGIR International Conference on Theory of Information Retrieval, ICTIR '17, ACM, 2017, pp. 117–124. doi:10.1145/3121050.3121062.