Querying NoSQL-based Crowdsourcing Systems Efficiently

Alfredo Cuzzocrea[1], Marcello Di Stefano[2], Paolo Fosci[3], and Giuseppe Psaila[3]

[1] University of Trieste and ICAR-CNR, Italy
alfredo.cuzzocrea@dia.units.it
[2] University of Palermo, Italy
marcello.distefano@unipa.it
[3] University of Bergamo, Italy
paolo.fosci@unibg.it, psaila@unibg.it

Abstract. In this paper, we provide *a novel approach for effectively and efficiently support query processing tasks in novel NoSQL crowdsourcing systems*. The idea of our method is to exploit the *social knowledge* available from reviews about products of any kind, freely provided by customers through specialized web sites. We thus define a *NoSQL* database system for large collections of product reviews, where queries can be expressed in terms of natural language sentences whose answers are modeled as lists of products ranked based on the relevance of reviews w.r.t. the natural language sentences. The best ranked products in the result list can be seen as the best *hints* for the user based on crowd opinions (the reviews). By exploiting the well-known IMDb dataset, which comprises more than 2 million reviews for more than 100,000 movies, we experimentally shows that our prototype obtains good performance in terms of execution time, demonstrating that our approach is feasible.

1 Introduction

Novel *NoSQL database systems* are playing a leading role in latest research, due to their well-understood characteristics that well-marry with actual trends laying under the terms *Cloud Computing* and *Big Data*. On the other hand, *Crowdsourcing* is becoming a very attractive research area, particularly in the context of Web search scenarios. In this paper, we try to cross-layering these two leading contexts, by providing *a novel approach for effectively and efficiently support query processing tasks in novel NoSQL crowdsourcing systems*. The idea of our method is to exploit the *social knowledge* available from reviews about products of any kind, freely provided by customers through specialized web sites. We thus define a *NoSQL* database system for large collections of product reviews, where queries can be expressed in terms of natural language sentences whose answers are modeled as lists of products ranked based on the relevance of reviews w.r.t. the natural language sentences. The best ranked products in the result list can be seen as the best *hints* for the user based on crowd opinions (the reviews).

Reviews about products that customers can freely write on specialized web sites constitute an incredible source of information, by means of which users would like to get useful *hints*. But how could a user obtain them? Typically, the user has some wishes and would like to find products that match those wishes, based on opinions of other users. But to do that, a specialized system is necessary.

Looking at the problem from a database technology point of view, product reviews constitute a text database that has a given structure; user's wishes can be seen as natural language queries over the set of reviews and the user wants to obtain the products whose set of reviews matches the query at the highest degree; the ideal solution, is to get a ranked list, where the best ranked products can be seen as the best hints for the user based on crowd opinions (the reviews). We focus only on reviews, since every single review is a mine of unstructured information that are hard to be queried by classical techniques. In other words, such a system is a *NoSQL* database system, where queries are natural language sentences.

Thus, our prototype is a *NoSQL* database system for large collections of product reviews; the database is queried by expressing a natural language sentence; the result is a list of products ranked based on the relevance of reviews w.r.t. the natural language sentence. Semantic tagging and term expansion (by means of WordNet) are performed, both indexing reviews and querying them. We aim at demonstrating that it is possible to obtain an answer to a query in acceptable time on a large set of reviews. Therefore, we tested the prototype on reviews about movies downloaded from the IMDb.com web site, that includes more than 2 million reviews for more than 100,000 movies. The study about execution times at query time is presented.

2 Related Work

Research in the database area is more and more addressing the concept of NoSQL database. Several attempts to define the concept can be found. Referring to [7], three categories of data stores are considered: *Key-Value stores*, *Document stores* and *Column Family stores*. The first category deals with datasets similar to maps or dictionaries where data are addressed by a unique key. The second category deals with sets of text documents, and our work falls into this category. The third category encompasses column oriented stores, extensible record stores and wide columnar stores. *Graph databases* can be considered as well belonging to the world of NoSQL databases [6].

One important aspect of NoSQL data stores is performance. Often, NoSQL databases are necessary due to the implicit limitation of relational databases in given application contexts, where the relational structure of data is an obstacle to obtain fast execution times. The work in [1] tries to address this perspective and presents six features of NoSQL data stores. The are: (1) the ability to horizontally scale simple operation throughput over many servers; (2) the ability to replicate and to distribute (partition) data over many servers; (3) a simple call level interface or protocol (in contrast to a SQL binding); (4) a weaker concurrency model than the ACID transactions of most relational (SQL) database systems; (5) efficient use of distributed indexes and RAM for data storage; (6) the ability to dynamically add new attributes to data records.

The ranking metrics we defined as the basis of the proposed system is inspired to the concept of itemset, developed in the area of data mining for mining frequent itemsets and association rules. Several works [4, 5] adopt association rule mining for analyzing customer reviews and extract opinions from them. In [4], association rule mining is used to extract, from within customer reviews, relevant features the characterize opinions of users about products. In [5], a system to compare opinions about products is presented, where product reviews reports PROs and CONs; in particular, association rule mining is exploited to assign a positive or negative polarity to words (namely, adjectives) in product reviews, and use this polarity to rank the opinion about products. The work in [3] extracts, by means of an association rule mining technique, relevant features that summarize product reviews.

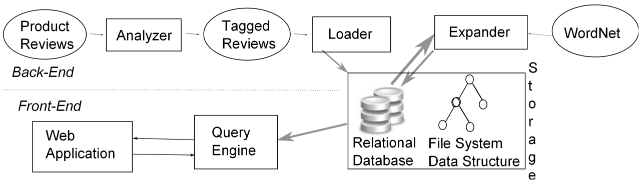


Fig. 1. Architecture of the Proposed System.

3 The Proposed NoSQL System

As stated in Section 1, our proposed prototype is a NoSQL database system that deals with collections of product reviews, that can be queried by expressing a natural language sentence (i.e. *query* in the rest of the paper).

In this Section we shortly mention system architecture and data structure underlying the *Query Engine*, that we describe in Section 4. The proposed system is composed by several components, each one devoted to perform a specific task as shown in Figure 1. In particular, we distinguish between the *back-end* and the *front-end*: the former is responsible for collecting, analyzing and indexing data from product reviews; the latter is the actual user interface accessed by users, that is built on top of the *Query Engine*. Let us describe the architecture in details.

Back-End. In this side of the system, we find the components (rectangles) that prepares the data structure on which queries are executed. These components operate on source data and intermediate results (ovals) and upload data structures in the *Storage* box.

- *Analyzer*. This component is responsible for analyzing product reviews, identifying words and their grammar category (noun, verb, adjective or adverb). This *pos-tagging* operation is performed by the *Stanford Parser*[4]. As a result, reviews are transformed into tagged sentences, composed of *tagged terms*, i.e., a term associated to a tag that denotes the grammar category.
- *Loader*. The goal of this component is to load *Tagged Reviews* into the data structures on which queries are performed.
- *Expander*. After the loading phase has ended, tagged terms are expanded on the base of an ontology (in our case, *WordNet*), so that the *Query Engine* can exploit semantic relationships in order to capture a wider set of results related with the query.

Front-End. From the architectural point of view, the key component of the front-end is the *Query Engine*: it exploits the preliminary work performed by back-end components, and works on data structures describing occurrences of terms in product reviews. The *Web Application* component has been developed to give end users the capability to exploit the system. Figure 2 graphically depicts the logical schema of the underlying NoSQL data-structure. Table `Products` describes each single product, and its schema is context-dependent, in the sense that the attributes are defined based on the application domain. For example, since we use IMDb movie data set to test the prototype, we defined attributes concerning movies, such as title, director, year, and so on. Table `Terms` is the key table, that describes each single tagged word managed by the system. Attributes `products`, `reviews` and `occurrences` counts the number of products and the number of reviews in which a tagged term occurs, and the total number of occurrences, respectively. Notice that, while attribute `taggedword`

[4] from the *Stanford NLP Group*: http://nlp.stanford.edu/index.shtml

```
Terms(id, word, tag, taggedword,
products, reviews, occurrences)
Term2Expansion(termId, expandedWordId, relation)
Occurrences(id, productId, termId, review,
position)
Product(id, domain specific attributes)
```

Fig. 2. Schema of the NoSQL Database.

denotes a unique element in the table, since a simple word can be associated to more than one grammar category (i.e. word *book* can be either a *noun* or a *verb*), it can appear several times in the table. Table `Term2Expansion` represent the relations of a tagged term (by means of attribute `termId`) with another tagged term (by means attribute `expandedTermId`). Attribute `relation` denotes the typology of expansion[5], i.e., synonym, hypernym, heponym to name a few. Notice that tagging a term with its grammar category (for instance *noun*), allows us to delimit word expansion only to the relations implied by the grammar category (i.e. the concept of *meronym* can be applied to a *verb*, but not to a *noun*). Finally, table `Occurrences` describes all occurrences of tagged terms in product reviews; in particular, notice attribute `position`, that indicates the position of the occurrence in the review.

Data Storing. The previous data structure could be totally implemented on a traditional relational database, with indexes on the main searching attributes of tables. But after we implemented this solution, since our data-set is quite big (see 5), when we submitted the first query-test the prototype still had to answer after one hour! So due to performance issues, part of the data resides as tables on a relational (*Postgres*) database (tables `Product`, `Terms` and `Term2Expansion`), and part on the file system (tables `Occurrences`). Specifically, as table `Occurrences` is likely to be huge[6], it has been split in single occurrences file for each term. Each file, containing occurrences of a single term, is identified by the term id. due to the very large number of terms, files are distributed in a subdirectory tree to avoid to saturate file system limits of files per directory. Internally, each occurrences file is organized as a binary file, where a fixed length data structure represents a term occurrence; this data structure is a 12-bytes triple *(ProductId, ReviewId, Position).* Furthermore, for the sake of performance study, we also have a 2nd version of the file system data structure, where occurrences are partitioned in 5 orthogonal subtrees, and each subtree describes occurrences for 1/5 of the products. This second version allows us to implement a multi-thread query engine, with 5 threads running in a parallel way (see 5).

4 Query Processing Support

We now describe the key component, i.e., the query engine. Based on a natural language sentence (the query) it extracts those products whose reviews are mostly relevant for the query. Relevance is evaluated by means of a *ranking metric*; retrieved products are returned as a list sorted in reverse order of relevance. Hereafter, we describe how the ranking metric is defined.

[5] WordNet provides a set of 15 different possible expanding relations depending on word grammar category

[6] In our test case, the size is more than 12Gb

l	*#I_l*	*weight*	*termsets (I)*
4	1	0.5000	{funny, great, hilarious, jokes}
3	4	0.1000	{funny, great, hilarious} {funny, great, jokes} {funny, hilarious, jokes} {great, hilarious, jokes}

2	6	0.0167	{funny, great} {funny, hilarious}
			{great, hilarious} {great, jokes}
			{funny, jokes} {hilarious, jokes}

Fig. 3. Termsets levels for query *great funny hilarious jokes* and corresponding weights.

4.1 Termsets

In this paper we consider a query q as a *set of terms* (or briefly, a *termset*). Thus, we describe a query containing a number n of terms as $q = \{t_1, \ldots, t_n\}$[7], and we investigate only those queries where $n > 1$ or, in other words, $|q| > 1$. With I, we denote a generic termset that is a subset of q for which applies $|I| > 1$. With D_q, we denote the set of termsets I derived from q. Notice that the cardinality of D_q is $|D_q| = 2^n - (n + 1)$, i.e. D_q is the power set of q without the empty set and the n single terms that compose q. With I_l we denote an l-termset of q, that is a termset composed by l terms, i.e. $|I_l| = l$. With $D_{q,l}$ we denote the set of l-termsets I_l. Notice that the cardinality of $D_{g,l}$ is $|D_{q,l}| = \binom{n}{l}$.

4.2 Termset Weight

We now define the concept of *weight* for a termset.

Definition 1: The weight of a $l-$termset is a function of its length and the length of the query q ($|q| = n$) and it is denoted as $w_q(l)$.

For $n = 2$ there is only one 2-termset and its weight is $w_q(2) = 1$ by definition.

For $n > 2$ the weight of the single $n-$termset q is, by definition, $w_q(n) = 0.5$, while for $2 < l < n$ it is $w_q(l) = w_q(l + 1)/(\binom{n}{l} + 1)$ and for $l = 2$ it is $w_q(2) = w_q(3)/\binom{n}{2}$. □

The rationale behind Definition 1 is the following. The topmost termset, corresponding to the whole query, is the most important one, and its weight is equal to the overall weight of all the shorter termsets. The same principle is valid for any generic termset I_l (with $2 < l < n$), whose weight is equal to the overall weight of all lower levels termsets (even those that are not subset of I_l). In this way, reducing the size of termsets, the contribution of each level quickly decreases. Notice, that the overall weight of all termsets is exactly 1 ($\sum_{I \in D_q} w_q(|I|) = 1$). Figure 3 shows the termsets levels with an example query.

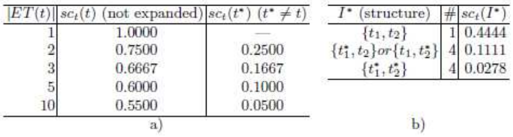
4.3 Query Expansion and Semantic Coefficient

As stated above, reviews are processed performing several operations. Similar operations are performed on a user query in natural language as well.

Pos-Tagging. By means of *Stanford Parser*, each word of a user query is tagged with an attribute that denotes its grammar role (*verb*, *noun*, *adjective* to name a few) in the query.

Stopwords Filtering. Stopwords are those words that are too common in reviews (such as *articles*, *conjunctions*); furthermore, common verbal forms like *is* or *have* (just to name a few) are treated as stopwords. Stopwords include also some very context-dependent words such as the word *actor* in a movie context. These words hold a small semantic meaning,

[7] At moment, in this stage of the project we do not consider word order or repetitions



$\|ET(t)\|$	$sc_t(t)$ (not expanded)	$sc_t(t^*)$ $(t^* \neq t)$
1	1.0000	—
2	0.7500	0.2500
3	0.6667	0.1667
5	0.6000	0.1000
10	0.5500	0.0500

a)

I^* (structure)	#	$sc_t(I^*)$
$\{t_1, t_2\}$	1	0.4444
$\{t_1^*, t_2\} or \{t_1, t_2^*\}$	4	0.1111
$\{t_1^*, t_2^*\}$	4	0.0278

b)

Table 1. a) Trend of Semantic coefficient w.r.t. the cardinality of $ET(t)$. b) Semantic coefficient for a generic 2-termset where each term has two expansions.

so after pos-tagging operation *stopwords* are discarded from the query. Thus, denoting with SW the set of possible stopwords, in the rest of the paper the notation:

$$q = \{t_1, \ldots, t_n\}$$

includes only those terms $t_i \notin SW$, and, as stated in Section 4.1, we consider only those queries q such that $|q| > 1$ (actual length without stopwords).

Term Expansion. By means of *WordNet* ontology, each tagged term $t_i \in q$ is expanded with all those terms directly associated to t_i depending by its grammar tag. Thus, for example a *noun* is expanded with all its *synonyms*, *hypernyms* or *hyponyms* and so on, while a *verb* is expanded with all its *synonyms* or *meronyms*, to name a few. There are actually a total of 15 possible different relations between a tagged term and its expanded words. We denote with t_i^* the generic expanded term of t_i, and with $ET(t_i)$ the set of all expanded terms of t_i. By definition, $t_i \in ET(t_i)$ with an *identity* relation, thus, $|ET(t_i)| \geq 1$. Notice that, given a generic expanded term t^*, it can happen that $t^* \in ET(t_i)$ and $t^* \in ET(t_j)$ with $i \neq j$. In other words, we cannot state a-priori that $ET(t_i) \cap ET(t_j) = \oslash$ with $i \neq j$. As an example, the term *colour* can be an *hypernym* expansion for both terms *red* and *black*.

Query Expansion. An *expanded query* q^* is each combination of $\{t_1^*, \ldots, t_n^*\}$. We consider *valid* a combination $q^* = \{t_1^*, \ldots, t_n^*\}$ only if $t_i^* \neq t_j^*$ $\forall i \neq j$ Notice that the original query q is a particular q^* itself, and it is *valid* by definition.

Expanded Termsets. Previous considerations about query q and its expansions, are applicable to each termset I_l. With I_l^* we denote an *expanded termset* $I_l^* = \{t_1^*, \ldots, t_l^*\}$, and similarly I_l^* is *valid* only if $t_i^* \neq t_j^*$ $\forall i \neq j$. With $EI(I)$ we denote the set of all possible expanded termset I^* that can be derived from I. The cardinality of $EI(I) = \prod_{t \in I} |ET(t)|$, that is the number of all possible combinations of the expanded terms of those terms that compose I. Finally, with D_q^*, we denote the set of all valid expanded termsets that are included in q and all its valid expansions q^*.

Semantic Coefficient. Each $t^* \in ET(t)$ has a *semantic coefficient* $sc_t(t^*)$, with $0 < sc_t(t^*) \leq 1$, that depends on the cardinality of $ET(t_i)$.

Definition 2: For each $t^* \in ET(t)$ except t, $sc_t(t^*) = 0.5/|ET(t)|$, and $sc_t(t) = 0.5 + 0.5/|ET(t)|$. □

The rationale of semantic coefficient, is the following. A term describes a semantic concept that is mostly expressed by the term itself, but receives a small contribution from expanded terms: the greater the number of expansion, the smaller the semantic contribution of a single expanded term. Notice that $\sum_{t^* \in ET(t)} sc_t(t^*) = 1$. Table 1.a shows how sc_t varies with $ET(t)$ cardinality.

With $sc_I(I^*)$ we denote the semantic coefficient for an expanded termset I^* derived from I.

Definition 3: Given an expanded termset $I^* = \{t_1^* \ldots t_l^*\}$ derived for a termset $I = \{t_1 \ldots t_l\}$, it is $sc_I(I^*) = \prod_{t_i^* \in I_l^*} sc_{t_i}(t_i^*)$. □

This way, a termset that contains only original terms gives the highest semantic contribution, while augmenting the number of expanded terms in the termset, the semantic contribution decreases. Table 1.b shows the trend of the semantic coefficient for a 2-termset where each original term has two expansions. Notice that, according to the above definition, $\sum_{I^* \in EI(I)} sc_I(I^*) = 1$.

4.4 Product Reviews and Termsets

Consider a product p (a movie, a camera, etc.), its set of reviews is denoted by $R(p) = \{r_1, \ldots, r_k\}$. Each review is a text, i.e., a sequence of term occurrences $r_i =< t_1, \ldots, t_s >$. With $T(R(p))$ we denote the set of terms appearing in reviews for product p, and with $T(r_i)$ the set of terms appearing in review $r_i \in R(p)$.

Definition 4: A termset I is said *relevant* for product p if $\exists r_i | I \subseteq T(r_i)$. □

The set of relevant termsets for product p is denoted as $RD_{p,q}$. In an analogous way, $RD^*_{p,q}$ is the set of all relevant expanded termsets for product p. Notice that $RD_{p,q} \subseteq D_q$, and also $RD^*_{p,q} \subseteq D^*_q$.

4.5 Termset Average Density

In a preliminary work [2], we assumed that every termset occurrence in product reviews contribute to the *support* of the termset with the same weight, i.e. 1, since the support, by definition, is the number of reviews containing the termset on the total amount of reviews. Given a termset I, in a single review, terms in I can be very dense or, on the opposite case, very sparse. We consider a review in which the occurrences of terms in I are dense being more relevant for the query than a review where occurrences are sparse. Thus, we introduce the concept of *Termset Density* of an termset I for a single review.

Definition 5: Consider a product p, a review $r \in R(p)$, and a termset I_l. The *Termset Review Density* $d_r(I_l)$ is defined as

$$d_r(I_l) = l / minWin_r(I_l)$$

where $minWin_r(I_l)$ is the size of the minimal window in review r that includes all the terms of termset I_l. □

Notice that for *Termset Review Density*, it holds that $0 < d(I_l, r) \leq 1$. The next step is to define a *Termset Average Density* for a generic termset I (we omit the subscript l not to burden notation) w.r.t. a product p.

Definition 6: Consider a product p and its set of reviews $R(p)$. With $R_I(p)$ we call the subset of $R(p)$ of those reviews containing termset I. The *Termset Average Density* for product p, denoted as $ad_p(I)$, is defined as:

$$ad_p(I) = (\textstyle\sum_{r \in R_I(p)} d_r(I)) / |R(p)|$$

□

The Termset Average Density is analogous to termset support, with the difference that the contribution of the occurrence of a termset I in a review r is not 1 but its density $d_r(I)$. Notice, thus, that $ad_p(I) \leq s_p(I) \leq 1$, where with $s_p(I)$ we denote the support of a termset I for a product p.

4.6 Product Ranking Metric

Finally, we can now define the *Product Ranking Metric PRM.*

Definition 7: Consider a query q, the set of termsets D^*_q derived from q, the system of the weights $w_q(|I^*|)$ and semantic coefficients $sc_q(I^*)$ for each expanded termset $I^* \in D^*_q$. Consider a product p, the set of reviews $R(p)$ and the set of relevant expanded termsets $RD^*_{p,q}$ that can be actually extracted from $R(p)$. Given for each $I^* \in RD^*_{p,q}$ the average termset density $ad_p(I^*)$, the *Product Relevance Value* for product p is defined as

Schema	A	B	Diff %
Pos-Tagger	active	inactive	
Distinct tagged terms	1,151,827	776,852	-32.55%
Occurrences	216,345,522	216,345,522	0.00%
Analysis Time (A = Ps+Pt)	2226.80h	3.82h	-99.83%
Parsing Time (Ps)	2.11h	2.42h	+14.74%
Pos-tagging Time (Pt)	2224.69h	1.40h	-99.94%
Db Loading Time (D)	56.05h	49.76h	-11.23%
Term Expansion Time (E)	3.73h	2.67h	-28.49%
Total Time (T = A+D+E)	2286.58h	56.25h	-97.54%

Table 2. Indexed schemes

$$PRM_q(p) = \sum_{I^* \in RD^*_{p,q}} (w_q(|I^*|) \times ad_p(I^*) \times sc_q(I^*))$$

□

The rationale of the above definition is the following. For each termset I^* included in the query q and actually relevant in the reviews, its contribution to the overall relevance value is given by its weight $w_q(|I^*|)$ (that depends on its size) multiplied by its *average density* $ad_{p,q}(I^*)$ and its semantic coefficient $sc_q(I^*)$. The system of weights and semantic coefficients has been designed to obtain a $PRM_q(p) = 1$ for an *ideal* set of reviews for product p, where each review contains every expanded termset I^* that can be derived from q with a density $d_r(I^*) = 1$, and every expanded termset I^* is *valid*.

5 Experimental Assessment and Analysis

Our dataset is composed by a total of *2,207,678* user reviews for *109,221* movies downloaded from the IMDb.com web site[8]. The size of the text we downloaded is approximatively *3,091Mb*. Each movie has a number of reviews included between *1* and *4,876*, and the average number of reviews per movie is *20*.

Experiments has been run on a PC with two Intel Xeon Quad-core 2.0GHz/L3-4MB processors, 12GB RAM, four 1-Tbyte disks and Linux operating system.

While indexing our data set, as described in the *back-end* side of proposed architecture in 3, we figured out how *pos-tagging* affects the proposed system. Disabling pos-tagging means tagging each term with a unique trivial tag, and considering for each term every possible expansion regardless of its role inside the query; in other words, disabling pos-tagging means a significant reduction of the number of managed terms because words are distinguished on the basis of their grammar category (for instance word *colour* could be both a noun and a verb); however, the counter effect is that the possible number of expansions for a termset combinatorially increases.

Table 2 reports data collected during dataset indexing. Column A shows data regarding indexing with pos-tagging activated (*Schema A*), while Column B shows data regarding indexing with pos-tagging deactivated (*Schema B*). Column *Diff %* shows the percentage variation from data of Schema A to data of Schema B (where applicable).

For our query performance tests we prepared a set of 25 standard user queries like *I want to know more about the history of Greece and the Persian wars*, or *All those moments will be lost in time, like tears in rain*[9].

[8] we focus on *movies* as Imdb data-set as been the first freely accesible big-data set we've found. Anyway the same approach is suitable for any set of product reviews.

[9] from *Blade Runner* movie

	Single-thread	5-threads	Diff %
Average Time (T=QE+TG+TE+TM+S)	2,501.12 ms	1,994.66 ms	-20.25%
Query Expansion (QE	286.44 ms	286.40 ms	-0.01%
Thread generation (TG)	0.40 ms	1.88 ms	370.00%
Thread execution (TE $\leq$ O+R)	2,199.64 ms	1,691.60 ms	-23.10%
Occurrences Loading (O)	1,962.52 ms	1,639.84 ms	-16.44%
Ranking (R)	237.12 ms	75.12 ms	-68.32%
Thread merging (TM)	1.64 ms	1.80 ms	9.76%
Sorting (S)	13.00 ms	12.98 ms	-0.17%

Table 3. Single-thread search engine Vs 5-threads search engine

Our test compares the variation of performance of the query engine working on *Schema A* (ad described in Section 5) in a *single-searching-thread* version versus a *5-searching-threads* version. Table 3 shows the average results of the test performed on the set of 25 standard queries mentioned before. Column *Single-thread* shows performance of the single-thread search engine, while Column *5-threads* shows performance of the 5-threads search engine, and column *Diff %* shows the percentage variation from single-thread w.r.t. 5-threads search engine. For each search engine version, the average execution time per query is provided in row *Average Time.*

6 Conclusions and Future Work

The scope of this paper was to present the architecture and the query engine of a *NoSQL* database system. Although performance of the system can be further be improved, the considerations in 5 show that the approach is feasible in terms of query response time. We are aware we did not discuss about system effectiveness, but it was beyond the scope of the paper. However the web-interface we developed is designed to collect users opinions about the system, and by means of that, in the future work we intend to deeper investigate effectiveness of the system. Moreover, as far as effectiveness is concerned, in the future work we intend to integrate term expansion with *linked-data* as a source for semantic ontology about terms, and also considering *word order* and *word repetition* in queries in our ranking model. In the end, we are also aware we have to deal with advanced semantic issues such as *negative sentences.*

References

1. R. Cattell. Scalable sql and nosql data stores. *SIGMOD Record*, 39 (4):12–27, 2011.
2. Paolo Fosci and Giuseppe Psaila. Toward a product search engine based on user reviews. In *DATA-2012 Int. Conf. on Data Technologies and Applications, Rome (Italy)*, July 2012.
3. Minqing Hu and Bing Liu. Mining and summarizing customer reviews. In Won Kim, Ron Kohavi, Johannes Gehrke, and William DuMouchel, editors, *KDD*, pages 168–177. ACM, 2004.
4. Minqing Hu and Bing Liu. Mining opinion features in customer reviews. In Deborah L. McGuinness and George Ferguson, editors, *AAAI*, pages 755–760. AAAI Press / The MIT Press, 2004.
5. Bing Liu, Minqing Hu, and Junsheng Cheng. Opinion observer: analyzing and comparing opinions on the web. In Allan Ellis and Tatsuya Hagino, editors, *WWW*, pages 342–351. ACM, 2005.
6. Hecht Robin and Stefan Jablonski. Nosql evaluation: A use case oriented survey. In *CSC-2011 International Conference on Cloud and Service Computing, Hong Kong, China*, pages 336–341, December 2011.
7. C. Strauch. Nosql databases. *http://www.christof-strauch.de/nosqldbs.pdf*, 2011.