Knowledge Driven Query Sharding[⋆]

Adam Krasuski[1] and Marcin Szczuka[2]

[1] Chair of Computer Science, The Main School of Fire Service, Poland
[2] Institute of Mathematics, The University of Warsaw, Poland
krasus@inf.sgsp.edu.pl,szczuka@mimuw.edu.pl

Abstract. We present the idea of an approach to database query sharding that makes use of knowledge about data structure and purpose. It is based on a case study for a database system that contains information about documents. By making use of knowledge about the data structure and the specific top-k queries to be processed we demonstrate a method for avoiding costly and unnecessary steps in query answering. We also demonstrate how the knowledge of data structure may be used to perform sharding and how such sharding may improve performance. We propose generalization of our findings that could lead to self-optimization and self-tuning in RDBMS engines, especially for column-based solutions.

Keywords: Text mining, document grouping, top-k queries, query processing, database sharding, column-oriented.

1 Introduction

The SYNAT project (abbreviation of Polish "**SY**stem **NA**uki i **T**echniki", see [2]) is a large, national R&D program of Polish government aimed at establishment of a unified network platform for storing and serving digital information in widely understood areas of science and technology. The project is composed of nearly 50 modules developed by research teams at 16 leading research institutions in Poland.[3] Within the framework of a larger project we want to design and implement a solution that will make it possible for a user to search within repositories of scientific information (articles, patents, biographical notes, etc.) using their semantic content. Our prospective system for doing that is called SONCA (abbreviation for **S**earch based on **ON**tologies and **C**ompound **A**nalytics, see [6, 8, 9]).

Ultimately, SONCA should be capable of answering the user query by listing and presenting the resources (documents, Web pages, et cetera) that correspond

[⋆] This work was supported by the grant N N516 077837 from the Ministry of Science and Higher Education of the Republic of Poland, the Polish National Science Centre grant 2011/01/B/ST6/03867 and by the Polish National Centre for Research and Development (NCBiR) under SYNAT - Grant No. SP/I/1/77065/10 in frame of the strategic scientific research and experimental development program: "Interdisciplinary System for Interactive Scientific and Scientific-Technical Information".

[3] http://www.synat.pl

to it *semantically*. In other words, the system should have some *understanding* of the intention of the query and of the contents of documents stored in the repository as well as the ability to retrieve relevant information with high efficacy. The system should be able to use various knowledge sources related to the investigated areas of science. It should also allow for independent sources of information about the analyzed objects, such as, e.g., information about scientists who may be identified as the stored articles' authors.

In order to be able to provide semantical relationships between concepts and documents we employ a method called Explicit Semantic Analysis (ESA) [4]. This method associates elementary data entities with concepts coming from knowledge base. In our system the elementary data entities are documents (scientific articles) collected from the PubMed Central database (PMC, see [1]) and as knowledge base we use the Medical Subject Headings (MeSH [4]) – a comprehensive controlled vocabulary for the purpose of indexing journal articles and books in the life sciences. We have field-tested a modified version of the ESA approach on PMC using MeSH (see [5,11]), and found out that while conceptually the method performs really well, the underlying data processing, especially the part performed inside RDBMS, requires introduction of new techniques.

From the database viewpoint the problem we want to solve is one of performing a *top-k* analytic, agglomerative SQL-query that involves joins on very large data tables (see [7]). The characteristic property of our task is the possibility of decomposing it into smaller subtasks (tasks on sub-tables), given some knowledge about the nature and structure of our data set. The fact that the query-answering can be decomposed into largely independent sub-tasks makes it possible to optimize it by using only top-k instead of all sub-results most of the time. Inasmuch as sub-tasks are largely independent from one another, we can also create shards and process them concurrently using, e.g., multiple cores (processors).

While optimizing the execution of queries in our particular case, we have noticed that the problem we are dealing with is of general nature. Our way of solving it, by decomposition to sub-tasks and then scheduling their concurrent execution, also appears to be generally applicable. That led us to formulation of general scheme for processing certain kind of queries with use of knowledge that we have about underlying data and query type. We have observed that while some database engines (e.g. PostgreSQL) partially support the kind of operations we want to perform, others do not. The lack of support for decomposition of query processing was especially problematic for us in case of column-oriented database systems (Infobright, MonetDB), since the column-oriented model is for various reasons recommended for our application.

The paper is organized as follows. We begin by introducing the example data set and query-answering task that has led us to key observations (Section 2). Then we generalize this particular problem to the one of knowledge-based task decomposition and sharding (Section 2.2) and show experimental results that demonstrate possible profits and limitations (3). Finally, we sketch the possible

[4] `http://www.nlm.nih.gov/pubs/factsheets/mesh.html`

improvements that could be introduced into database engines (Section 4) and finish with conclusions and our view on possible further work.

2 Description of the Problem

We present the general task that we are dealing with by means of a case study which is a version of the actual analytic task that we face while constructing the SONCA system (see [6, 8, 9, 11]).

For the sake of clarity of the presentation, we will first make some simplifying assumptions. As mentioned in Introduction, the analytical part of the SONCA database stores the information about different types of objects such as authors, publications, institutions and so on. In this case study we consider only objects of type *publication*, i.e., documents. Let us assume that the information about these objects is stored in a table called *document_word*. The table contains columns as follows: *doc_id* – identifier of the document, *word_pos* – an ordinal number of the given word in document, and *word* – a word from the document. Thus, to store a document we need as many rows as there are words in it. In the relational algebra formulæ below we will refer to this table as R.

The second table which is involved in our calculations is called *word_stem*, and denoted by S. The table contains two columns *word* and *stem* – which represent the stem for the given word. A stem is the root or roots of a word, together with any derivational affixes, to which inflectional affixes are added. The table stores the information about the stem and the stemming process, which was performed earlier using a standard Porter stemmer[5].

The last table needed to present a sample of analytic querying performed on documents, is called *stem_concept* and denoted by T. The table contains three columns, as follows: *stem*, *concept* – the name of the concept from MeSH controlled vocabulary and *weight* which associates quantitatively the concept with a given stem. The *stem_concept* table represents an inverted index for the ESA method.

Figure 1 outlines the tables introduced. The tables will be used to explain the ESA method, which aims at determining the semantical relationships between documents and (MeSH) concepts.

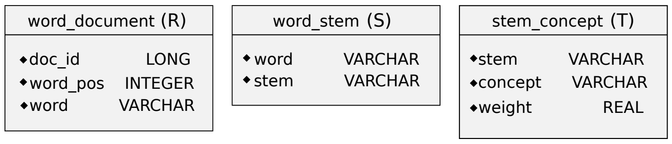

Fig. 1. The tables used in case study.

[5] http://tartarus.org/~martin/PorterStemmer/

2.1 Determining Semantical Relationships between Documents

In our (SONCA) system we would like to associate each document with a list of concepts from an ontology or a knowledge base, such as MeSH [11], Wikipedia [12], and so on. This, technically speaking, corresponds to creation of a vector of ontology concepts associated with the document. The vector is constructed in such a way, that each position corresponds to an ontology concept and the numerical value at this position represents the strength of the association. The strength is derived using the Explicit Semantic Analysis (ESA, see [4]). In a nutshell, the calculation of the association between concept(s) and the document in ESA comprises of three steps: stemming, stem frequency calculation, and the retrieval of the set of concepts relevant for (strongly associated with) stems corresponding to the document. We describe this calculation with relational calculus formulæ corresponding to SQL queries.

First, the inflected or derived words are reduced to their stem. We create new table called *word_doc_stemmed* – *R2* as the result of the join of tables *document_word* – *R* and *word_stem* – *S*.

$$R2 \leftarrow \Pi_{(R.doc\_id, S.stem)}(R \underset{word=word}{\bowtie} S) \tag{1}$$

The next task is the calculation of the stem frequency within the documents. We perform this using one table *R2*. The term (stem) frequency is calculated as:

$$\begin{aligned} R3 \leftarrow \Pi_{(U1.doc\_id, U1.stem, (U1.cnt/U2.cnt\_all) \rightarrow tf)} \Bigg(\\ \rho_{U1} \Big(\gamma_{(R2.doc\_id, R2.stem, R2.count(*) \rightarrow cnt)}(R2) \Big) \\ \underset{doc\_id=doc\_id}{\bowtie} \\ \rho_{U2} \Big(\gamma_{(R2.doc\_id, R2.count(*) \rightarrow cnt\_all)}(R2) \Big) \Bigg) \end{aligned} \tag{2}$$

The final step is the calculation of the vector of concepts associated with the document and the strength of the association.

$$\begin{aligned} R4 \leftarrow \Pi_{(R3.doc\_id, T.concept, assoc)} \Bigg(\tau_{(assocDESC)} \Bigg(\\ \gamma_{(R3.doc\_id, T.concept, SUM(R3.tf * T.weight) \rightarrow assoc)}(R3 \underset{stem=stem}{\bowtie} T) \Bigg) \Bigg) \end{aligned} \tag{3}$$

The queries presented above return the complete information, i.e., for each document they give us the levels of association with each and every concept in our ontology (knowledge base). This is both unnecessary and unwanted in practical applications. Empirical experiments show that if we are to present the results to the user, we shall present no more than top-k most associated concepts, with $k \leq 30$. Anything above 30 produces perceptual noise. So, as a last step in calculation we shall prune the result, leaving only top-30 most associated concepts in each documents' representation.

2.2 The Idea behind Query Sharding

As explained in the previous section, we are not interested in calculating all possible answers to a query. Hence, we propose to split the table and process it piece-wise. Each piece (shard) would contain information about an object in the data, such that it can be processed independently from other objects without distorting the global outcome. Once we have this sharding (task decomposition), we can distribute calculation among multiple threads on a single (multicore) processor or among several machines in networked environment.

The key to success is the knowledge about the composition of and relationships between the objects. If we possess the information (knowledge) that the objects are largely independent and can be processed in parallel, each shard separately. In our current approach this knowledge is derived from the domain by hand. However, it is imaginable that in the future an automated data (structure) analysis tool would make it possible to discover rules (criteria) for detecting situations we discuss here, and implement these rules using database triggers.

It is crucial to note, that the approach to processing queries using sharding which we propose, does not require a rewrite of the existing query optimizers. We propose a rewrite of the large query into a collection of smaller ones, that can be executed in parallel. We do not interfere with intra-query parallelization implemented in most RDBMS. Instead we apply a trick, creating a collection of virtual clients that send very simple queries to the database, instead of processing one global query that may be very time-consuming to answer.

By running queries for each of the pieces (documents) separately we achieve additional profit. We are able to process queries that require application of `LIMIT` operator within `GROUP BY` statement. This functionality was added in SQL:1999 and SQL:2003 standards [3] by introducing *windowing functions* and elements of *procedural languages*. Unfortunately, these functionalities are not supported in most column-based systems, such us Infobright[6] and MonetDB[7]. The ability to limit processing to top-k objects (documents) only can make a big difference in execution time, as demonstrated experimentally in Section 3.

```
1 $N$ := SELECT DISTINCT doc_id from TABLE
2 for $doc\_id \in N$ do
3     run SELECT ... WHERE DOC_ID = $doc\_id$ in K threads concurrently
4 end
```

Algorithm 1: Query sharding

In the case of example presented in Section 2.1 sharding corresponds to creation a separate query for each of the objects, since we have knowledge that there is no interference with other objects along the calculation. Objects correspond to documents, and the boundary of an object can be easily determined by

[6] `http://www.infobright.org/`
[7] `http://www.monetdb.org/`

detecting the change of *id* in column *doc_id*. Now, each of the involved queries presented in the Section 2.1, can be decomposed to a series of simple ones using the scheme presented in Algoritm 1.

3 Experimental Verification

In order to validate the usefulness of the proposed approach we have performed a series of experiments. In the experiments, we compared the processing of queries with and without the use of sharding. To have better overview we have included in the experiment the representatives of three major types of database technologies:

a) Infobright, which combines column-oriented architecture with Knowledge Grid;
b) PostgreSQL[8] which represents a row-oriented object-relational architecture with PL/pgSQL – a procedural language extensions of SQL.
c) MonetDB which represents purely column-oriented database architecture.

The results are summarized in Table 1 and Fig. 2.

Table 1. Results of the performed experiments.

Stemming (Formula (1))			
	Database	No shardnig	Sharding
1	Infobright	0 h 26 m 50.01 s	0 h 58 m 52.23 s
2	PostgreSQL	0 h 10 m 20.21 s	0 h 26 m 38.65 s
3	MonetDB	0 h 1 m 18.37 s	0 h 24 m 53.78 s
Stem frequency calculation (Formula (2))			
	Database	No shardnig	Sharding
1	Infobright	0 h 3 m 59.86 s	0 h 4 m 17.80 s
2	PostgreSQL	0 h 9 m 48.05 s	0 h 16 m 02.74 s
3	MonetDB	0 h 1 m 55.90 s	0 h 19 m 57.63 s
Vector of concepts calculation (Formula (3))			
	Database	No shardnig	Sharding
1	Infobright	22 h 22 m 0.39 s	8 h 42 m 6.74 s
2	PostgreSQL	24 h – no results	7 h 3 m 1.74 s
3	MonetDB	*MALException error*	8 h 17 m 30.50 s
Vector of concepts calculation (Formula (3) with LIMIT $k = 30$)			
	Database	No shardnig	Sharding
1	Infobright	*NA*	0 h 29 m 11.98 s
2	PostgreSQL `LOOP` within PL/pgSQL	16 h 58 m 28.03 s	1 h 27 m 51.64 s
3	PostgreSQL WINDOWING FUNCTION	17 h 22 m 30 s	1 h 27 m 51.64 s
4	MonetDB	*NA*	0 h 35 m 31.34 s

[8] `http://www.postgresql.org/docs/current/static/plpgsql.html`

Due to the fact that in column-oriented architectures that we use it is not possible to run query with `LIMIT` within `GROUP BY`, the comparison with performance of windowing functions and elements of procedural language (`LOOP` within PL/pgSQL) was performed only with PostgreSQL database. The experiments are based on an external implementation in Java with Apache DBCP[9] used for connection pooling that was required in parallel query processing. In all experiments we have used tables with the following number of rows: *word_document* – 370 878 730, *word_stem* – 76 108, *stem_concept* 517 729. All experiments were done on a server with Intel® Xeon® CPU X5650 @ 2.67GHz - 24 cores, with 64 GB memory and 600 GB SAS 6 GB/s 15 000 RPM disks.

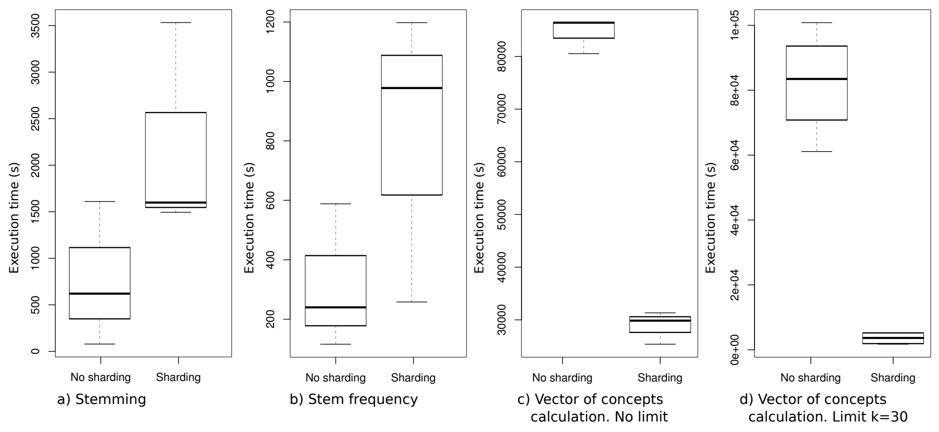

Fig. 2. Comparison of execution times for the experiment. Each of the boxes represents distribution of query execution time for three types of databases within specific task.

4 Discussion and Conclusions

The experiments clearly demonstrate that joining query sharding with parallel execution of sub-tasks has a potential. In some cases, the queries processed with using query sharding were executed from 3 to 23 times faster (Fig. 2c,d). Also, in column-oriented databases sharding was the method to get around the problem with enforcing limit `LIMIT` inside `GROUP BY`. The experiments, however, also demonstrate that specific conditions must be met in order for query sharding to be beneficial. Two out of four tasks tested experimentally exhibited significant decrease in performance – from 2.6 to 2.9 times slower (Fig. 2a,b). This is due to imbalance between the computational overhead created by the parallelization of the task and the complexity of the task itself. In case of both stemming and stem frequency calculation the cost of creating virtual clients that ask distributed

[9] `http://commons.apache.org/dbcp/`

queries was much higher than the collective profit obtained from processing only simpler queries on smaller data tables.

There is additional issue that we decided to address by performing additional experiment. As mentioned before, we have restricted ourselves to top 30 results because of the kind of objects we are dealing with. However, for other types of data the specific limit of 30 top results may be meaningless. Therefore, we have also conducted a smaller experiment, using only the Infobright instance (part of the current SONCA implementation) to check how the query execution changes with the increase of k in top-k limit. With $k = 100$ the vector of concepts (ESA) calculation took 29 min. 18.22 s. and for $k = 1000$ the execution time was 35 min. 25.09 s. We should also mention that good efficiency achieved by MonetDB on non-sharded, simple queries could not be extended to a sharded case due to problems that MonetDB has with facilitating multiple queries in parallel.

The conclusion from the experimental verification is the set of guidelines that have to be followed in order for the sharding to be effective. These guidelines are expansion of general ideas stated in Section 2.2. These are:

1. The query to be decomposed must contain a central, complex, agglomerative task, which involves joins on very large data tables. Typically, we would decide to use sharding if the auxiliary structures used to store `GROUP BY` data exceed the RAM allocation capabilities.
2. Secondly, all arithmetic operations must occur inside the join operation.

We strongly believe that these guidelines can be used to formulate a set of rules for automatic query execution tuning in database engines. That is, if certain conditions are met, the database engine would transform the query processing from traditional to sharding model. The key to success is the knowledge about the data structure and purpose, which makes it possible to avoid unnecessary calculations.

The proposed approach has one more advantage, which was especially valid for us in the context of our SONCA system. The set of smaller queries obtained the result of sharding may be executed independently and concurrently. Thanks to this, we can regulate the number of threads (machines, processors, cores) involved in the calculation at any given point. Since the results of each sub-query execution are stored and do not need to be accessed by others, the entire calculation can be interrupted and then picked up without any loss of information. This ability is usually hard to achieve in database systems that use multi-threading in query processing (see [10]). In our implementation we have achieved good control over load-balancing by performing the scheduling outside of database, using our own code. However, we strongly believe that a similar result can (and should) be achieved by implementing sharding inside the database engine. For the moment, we benefit from query sharding in the SONCA system. It gives us the ability to plan ahead tasks and perform them with optimal use of computing resources. This is not so crucial for simpler tasks, such as document processing (stemming, stem association), which normally take less than an hour, but for finding semantical relationships between concepts and sections, sentences or snippets it is of paramount importance, as these calculations may last for several days.

References

1. Beck, J., Sequeira, E.: PubMed Central (PMC): An archive for literature from life sciences journals. In: McEntyre, J., Ostell, J. (eds.) The NCBI Handbook, chap. 9. National Center for Biotechnology Information, Bethesda (2003), `http://www.ncbi.nlm.nih.gov/books/NBK21087/`
2. Bembenik, R., Skonieczny, Ł., Rybiński, H., Niezgódka, M. (eds.): Intelligent Tools for Building a Scientific Information Platform, Studies in Computational Intelligence, vol. 390. Springer, Berlin / Heidelberg (2012)
3. Eisenberg, A., Melton, J., Kulkarni, K., Michels, J.E., Zemke, F.: SQL:2003 has been published. SIGMOD Rec. 33(1), 119–126 (Mar 2004), `http://doi.acm.org/10.1145/974121.974142`
4. Gabrilovich, E., Markovitch, S.: Computing semantic relatedness using Wikipedia-based explicit semantic analysis. In: Proceedings of the 20th International Joint Conference on Artificial Intelligence. pp. 6–12 (2007)
5. Janusz, A., Świeboda, W., Krasuski, A., Nguyen, H.S.: Interactive document indexing method based on explicit semantic analysis. In: Proceedings of the Joint Rough Sets Symposium (JRS 2012), Chengdu, China, August 17-20, 2012. Lecture Notes in Computer Science, Springer (2012)
6. Kowalski, M., Ślęzak, D., Stencel, K., Pardel, P., Grzegorowski, M., Kijowski, M.: RDBMS model for scientific articles analytics. In: Bembenik et al. [2], chap. 4, pp. 49–60
7. Michel, S., Triantafillou, P., Weikum, G.: KLEE: a framework for distributed top-k query algorithms. In: Proceedings of the 31st international conference on Very large data bases. pp. 637–648. VLDB '05, VLDB Endowment (2005)
8. Nguyen, A.L., Nguyen, H.S.: On designing the SONCA system. In: Bembenik et al. [2], chap. 2, pp. 9–35
9. Nguyen, H.S., Ślęzak, D., Skowron, A., Bazan, J.: Semantic search and analytics over large repository of scientific articles. In: Bembenik et al. [2], chap. 1, pp. 1–8
10. Pankratius, V., Heneka, M.: Parallel SQL query auto-tuning on multicore. Karlsruhe Reports in Informatics 2011-5, Karlsruhe Institute of Technology, Faculty of Informatics (2011), `http://digbib.ubka.uni-karlsruhe.de/volltexte/documents/1978109`
11. Ślęzak, D., Janusz, A., Świeboda, W., Nguyen, H.S., Bazan, J.G., Skowron, A.: Semantic analytics of PubMed content. In: Holzinger, A., Simonic, K.M. (eds.) Information Quality in e-Health - 7th Conference of the Workgroup Human-Computer Interaction and Usability Engineering of the Austrian Computer Society, USAB 2011, Graz, Austria, November 25-26, 2011. Proceedings. Lecture Notes in Computer Science, vol. 7058, pp. 63–74. Springer (2011)
12. Szczuka, M., Janusz, A., Herba, K.: Clustering of rough set related documents with use of knowledge from DBpedia. In: Proceedings of the 6th International Conference on Rough Sets and Knowledge Technology (RSKT 2011), Banff, Canada, October 9-12, 2011. Lecture Notes in Computer Science, vol. 6954, pp. 394–403. Springer (2011)