EasyChair Preprint

№ 4194

High-Level Query Languages for Querying Big Data Sets

Christina Kang Xiaoxi, Rhodian Lu Jing Hong,
Muhammad Ramzan Ashraf and Patrice Boursier

September 15, 2020

High-Level Query Languagesfor Querying Big Data Sets

Christina Kang Xiaoxi, Rhodian Lu Jing Hong, Muhammad Ramzan Ashraf
School of Computing & IT
Tayor's University
Selangor, Malaysia
{kangxiaoxi, acknd, jacjksnaca}@sd.taylors.edu.my

Patrice Boursier
School of Computing & IT
Tayor's University
Selangor, Malaysia
Patrice.Boursier@taylors.edu.my

***Abstract*— This paper proposes a comparison of different approaches for querying big data sets with high-level query languages. It also presents the HiFun functional query language. A prototype is currently being developed that will allow to test the usability of the HiFun language.**

Keywords—big data, query languages, functional, HiFun

I. INTRODUCTION

The objective of the work presented in this paper is to develop a system for querying big data sets, with a simple GUI based on a high-level query language. It is based on Prof. Nicolas Spyratos' research on the *HiFun* high-level query language (Spyratos & Sugibuchi, 2018).

With such a system, the user will no longer need to know the database structure and any database programming language in order to query big data sets and easily perform some basic data analytics. The main algorithms that are being used are SQL group-by queries and MapReduce jobs, and it is based on the *HiFun* functional query language which is the main approach. *HiFun* produces expressions that can be encoded either as MapReduce jobs or as SQL group-by queries (Spyratos & Sugibuchi, 2018).

Data analysis is a well-established research field where it's applicable to multiple applications available in several domains. However, the volume of data accumulated by modern applications increased in unprecedented rates. Everything around our daily lives are all potential data, especially for internet platforms such as Google and Facebook which handle more than 2.5 petabytes of data. In order to analyze all these data, the technology has evolved a lot for improvements of data analytic processes.

In order to solve the big data process problem, a lot of new languages have been proposed, such as NoSQL, and such frameworks as the Apache Hadoop Big Data Platform. However, all these platforms need highly-skilled professionals who can handle and manipulate all the data. CapGemini's report discovered that 37% of companies have trouble in finding skilled data analysts to make use of their data (Vanessa Rombaut, 2016). The best bet is to form one common data analyst team for the company, either through retraining your current workers or recruiting new workers specialized in big data and big data analytics. In order to solve this, the *HiFun* platform is a formal framework which helps to analyze data queries based on MapReduce jobs or as an SQL group-by query. The database analyst will no longer need to know the query language but will still able to analyze the various datasets altogether. The system backend code will auto generate the query according to the user selection.

With a traditional SQL database design, entities are connected via relationships, and a call from one entity to another is required in order to find the relationship, and then only the value can be found. This requires a very intensive database study time and if it is a big database, it makes it even harder to implement. Now, a lot of data are unsupervised, and a lot of new values will appear in the databases. Hence for traditional database design, a lot of companies will encounter data integration problems. For example, if a user wants to use SQL to find the total quantity for a branch, he will need to understand the relationships between all entities and use join operations to find the correct values and do the calculations.

To solve this, *HiFun* is using a functional approach to connect all attributes. Each of the attributes will have a query function generator in the backend so that the code will auto generate according to the user selection. Instead of linking attributes via their relationships, the system we are developing is linking all the attributes by using different functions. An example is shown in Figure 1.1, extracted from (Spyratos & Sugibuchi, 2018), where attributes are linked by functions such as *r*, *b*, etc.

The background of the study is presented in section II. A comparison of big data query languages is proposed in section III. Section IV introduces the HiFun functional query language (Spyratos & Sugibuchi, 2018). Conclusion and future work are given in section V.

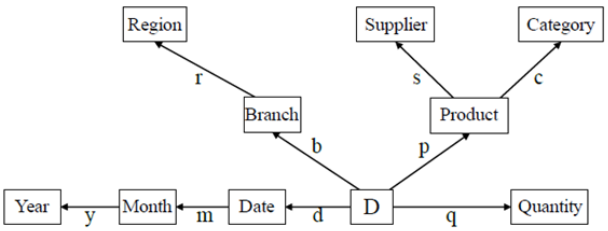

Figure 1.1: Database sample graph

II. Background

Data transactions are around us everywhere in our daily life. Digital devices are producing and collecting all kinds of data. Since the Smart City and Smart Home ideas have spread around the world, more and more data have been collected. According to the International Data Corporation (IDC) who released the 2020 Digital Universe report (Gantz & Reinsel, 2013), it is stated that the total amount of data available will double every two years (Guo, Zhang, & Zhu, 2015). The challenge for processing big data is how to bring big data analytics to a higher level. Big data processing cannot only be done by professional database administrators.

The development of big data technology has grown very rapidly during the last ten years, and companies are tracking information related to their customers. Millions of sensors capture data that are kept into databases (Manyika et al., 2011), and Google is processing more than 24 petabytes of data per day (CACM Staff, 2017).

Data that are being collected can be divided into different categories. From figure 2.1 we can see that the volume of unstructured data such as video or rich media is quite high and the volume of structured or semi-structured data such as social media feeds will be lower.

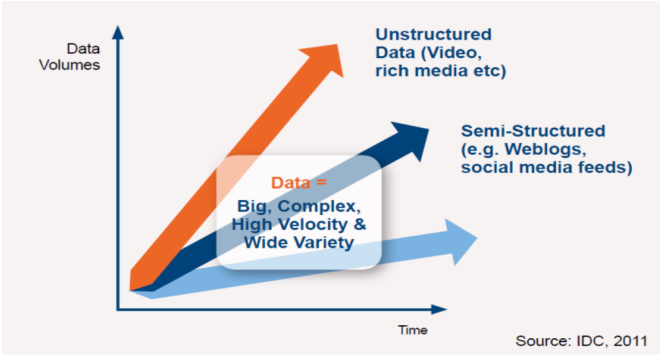

Figure 2.1: Semi-structured vs. unstructured data

Organizations and companies are willing to collect and process big data in order to improve their business profit. However, there are still some challenges when the company wants to go for Big Data. There are indeed a lot of requirements in order to become a professional data analyst. The other challenge comes from the fact that multiple databases need to be merged while staying accessible to the user. From one platform to another, how data can be processed together is another challenge faced by most organizations nowadays.

There are plenty of tools and algorithms that are available for big data processing, the most common ones being Apache Hadoop, Spark, Tableau, etc. Technologies are able to support data sets that come from different systems (Spyratos & Sugibuchi, 2018), especially the Apache Hadoop big data platform, which is based on the MapReduce framework and is very commonly used by many organizations nowadays.

However, there is a need for highly skilled professionals who will be able to handle and make use of the tools for the organization. They should be able to understand the different dimensions of big data modeling, architecture and especially data integration (Wani & Jabin, 2018). According to (Manyika et al., 2011), US might need 140,000 to 190,000 skilled professionals for data analysis as well as more than one million managers and analysts with advanced analytical knowledge and skills to make correct and accurate decisions.

As we can see, there is a very high demand from any organization who has engaged in big data analytics and frameworks. But every organization needs data scientists in order to make more profit from the data that has been collected from their system (Wani & Jabin, 2018), (Kim, Trimi, & Chung, 2014), (Manyika et al., 2011). The demand for professional data analytics is one of the challenges that big data is facing nowadays.

The efficiency and "interactiveness" of big data processing systems, which will allow the user to access different types of databases, will be some of the main challenges for big data processing (Che, Safran, & Peng, 2013). Interactiveness is one of the critical challenges for system designers and data scientists (Wani & Jabin, 2018), and the lack of interactiveness will drop down the performance of the data processing result.

III. Big Data Query Languages

Most data that are stored in a database are processed via a database management system (DBMS) (Schweikardt, Schwentick, & Segoufin, 2010). There are different types of database systems that are usually used with different kinds of query languages depending on the types of data and the size of the database. This section will focus on query languages and their limitations.

The main purpose of a database management system is to be able to query data. In general, a query is a mapping which takes a database instance D and maps it into a relation of fixed entity. The query language is what will allow users to pose queries in a semantically unambiguous way. SQL and NoSQL are common query languages that are used with relational and non-relational databases, respectively. SQL databases handle structured data and have a predefined

schema whereas NoSQL databases handle unstructured data and have a dynamic schema.

The limitations of traditional query languages are related to the complexity and the integration from one language to another. For example, SQL can be in some cases a very complex language (Schweikardt et al., 2010). Join operations can also be very costly, and it will not be very suitable for querying large data sets.

In addition, all these query languages are low-level, and they require professionals with database programming skills in order to process and query the data.

High Level Query Languages

Several high-level query languages have been built on top of Hadoop, for example Pig, Hive or JAQL. The relationships between high-level query languages and Hadoop are shown in Figure 3.1 (Stewart et al., 2011). High-level query languages are able to be compiled into a sequence of MapReduce jobs and also executed in different environments.

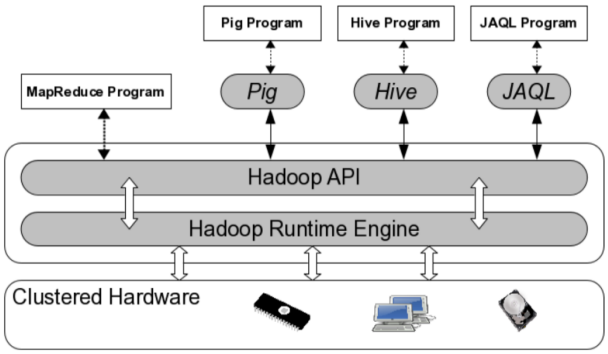

Figure 3.1: High Level Query Language Implementation Stack

Figure 3.2 illustrates how high-level query languages can help increasing computational power.

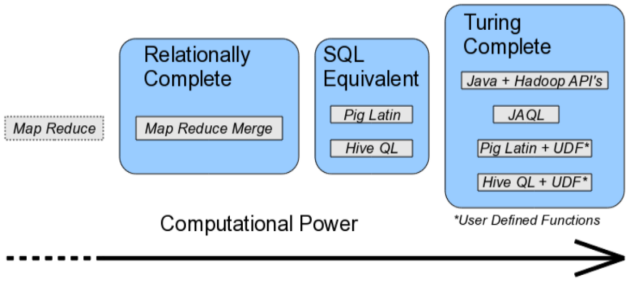

Figure 3.2: Computational Power Comparison (Stewart et al., 2011)

Relational completeness is a mathematical notation which defines the relationship which the operators and functions are required for the relationship between each of the entities. The SQL language is relationally complete, and it is able to provide for all operations through the *relational algebra*. Some additional aggregate functions can also be done by SQL such as average, count and sum.

Turning Completeness defines a language that has conditional constructs which can define the recursion for each iteration, memory architecture and emulate an infinite memory model which is suitable for unsupervised data.

User Defined Functions means that the programs are customized for the users with data formats and bespoke functions. Users are able to define functions that provide them with a higher level of data processing.

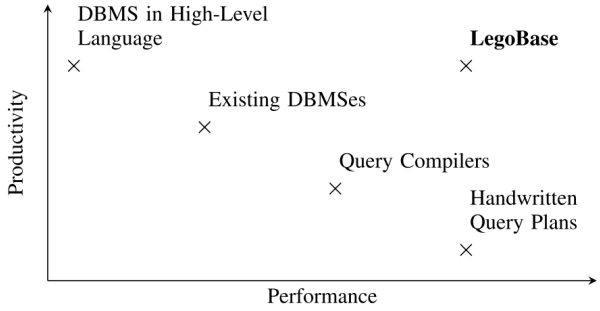

Figure 3.3: Comparison of performance/productivity trade-off for different approaches

Figure 3.3 shows a comparison between types of queries and processing methods. A DBMS with a high level language gives the highest performance (Shaikhha et al., 2016). The DMBS will produce high performance and the developers will be more efficient at the high-level of abstraction without being affected by the negative performance impact. In addition, a high-level query language can also quickly define the system modules. The system will have more flexibility and enable the user to choose and experience with a number of possibilities when building the query engines.

Below are given examples of high-level query languages for querying big data sets.

Pig – A high level data flow interface for Hadoop (Stewart et al., 2011).

Pig is a high-level dataflow system that aims to combine SQL and MapReduce by having high-level data manipulation constructs which can be assembled in an explicit dataflow which interleaves SQL queries with MapReduce functions. Pig programs will first parse for syntactic and instance checking then produce the logical plan and arrange it in a directed acyclic graph. The logical plan is to be compiled in the logical plan compiler and then optimized once more by the MapReduce optimizer by using the map-reduce combiner function. The MapReduce program will be executed in the Hadoop application.

Pig provides simple data types such as *int* and *double* but also non-normalized data models. Various data types are supported by Pig collections such as cover maps, tuples and bags. A sample of Pig word count query is given is Figure 3.4.

```
myinput = LOAD 'input.dat' USING PigStorage();
grouped = GROUP myinput BY \$0;
counts = FOREACH grouped GENERATE group,
         COUNT(myinput) AS total;
STORE counts INTO 'PigOutput.dat' USING PigStorage();
```

Figure 3.4: Pig Word Count Benchmark

Hive – A data warehouse infrastructure for Hadoop (Stewart et al., 2011).

Hive is a query language which provides entry points for data analysts, minimizing the pain to migrate to the Hadoop infrastructure for distributed data storage and parallel query processing. Hive also supports SQL and declarative HiveQL and combines MapReduce jobs such as Hadoop HLQLs. Hive also includes SQL features such as *join*, *group-by*, aggregations and *create table as select all,* which make HiveQL very much SQL-like.

Data structures in Hive are like tables, columns, rows and partitions which are easily understood database concepts. Hive supports all primitive types of data such as integers, floats, doubles and strings and as well as maps, lists and struct. Hive also has a system catalogue which is a *meta store* that contains schemas and statistics which are very useful in data exploration, query optimization and query compilation. Like mentioned before for Pig, Hive also includes a query compiler that compiles a Hive query into as acyclic graph of MapReduce tasks. A sample Hive word count query is shown in Figure 3.5.

```
CREATE EXTERNAL TABLE Text(words STRING)
LOCATION 'input.dat';
FROM Text INSERT OVERWRITE DIRECTORY 'HiveOutput.dat'
SELECT words, COUNT(words) as totals
GROUP BY words;
```

Figure 3.5: Hive Word Count Query

JAQL – A JSON Interface to MapReduce

JAQL is a functional big data query language which is built upon the JavaScript Object Notation Language (Stewart et al., 2011). JAQL is a dataflow language that combines structured and non-structured database information and transfer it into JSON values. The framework of JAQL is able to read and write data in custom formats and provides support for common input/output formats. In a similar way as Pig and Hive, JAQL is also able to operate filtering, transformations, sort, group-by, aggregation and join.

JSON operates with different data types such as numbers, strings, arrays etc. For the mismatch of different data models, JSON is able to provide easy migration of data from one to another. A sample query is shown in Figure 3.6.

```
$input = read(lines('input.dat'));
$input -> group by $word = $
into { $word, num: count($) }
-> write(del('JAQLOutput.dat',{fields:[''words'',''num'']}));
```

Figure 3.6: JAQL Word Count Benchmark $input

Lego Base Lightweight Modular Staging Compiler (Shaikhha et al., 2016).

The Lego Base system modular architecture is shown in Figure 3.7 which is the system architecture. Programmers will be able to write queries as high-level Scala programs that will generate and process the query.

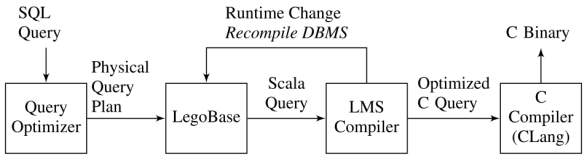

Figure 3.7: Overall Lego Base system architecture

The Lego Base system offers user-friendly concepts by providing database specification and optimization as a library and by generating the query engine. LMS (Lightweight modular staging) will perform all possible user-defined queries and generate a closer and final code for queries. Lego Base can compile expressions at runtime.

LMS provides programmers with many optimizing functions such as code elimination, constant propagation, loop fusion, deforestation and code motion.

According to (Stewart et al., 2011), there are some common features shared by Pig, Hive and JAQL, and because of this they are not only limited to their own core functionality but also have increased their power according to the user definition.

IV. The HiFun Approach

HiFun is a high-level functional query language which is used for defining analytic queries for big data sets analytics (Spyratos & Sugibuchi, 2018). As defined in the previous section, a high query language is more flexible and easier to use for data analytics.

HiFun is based on well-formed functional queries, and it is very similar to relational queries (Spyratos & Sugibuchi, 2018). HiFun expressions can include SQL group-by queries but they can also include MapReduce jobs.

HiFun can also include group-by queries for NoSQL. It is able to independently evaluate lower level mechanisms for the higher-level query.

In the following, we will explain the HiFun approach and the advantages and concerns of using HiFun for big data analytics.

The most import concepts proposed by HiFun are Grouping, Measuring and Reduction. A HiFun expression can be shown as: *Q = (g, m, op)*
Where g stands for Grouping, m stands for Measuring and op stands for operations.

Let's take an example to see how expressions work. Suppose D is the set of invoices in a supermarket over a year. An invoice has information about the delivery date, branch,

product name and quantity of the product. Figure 4.1 shows a sample dataset.

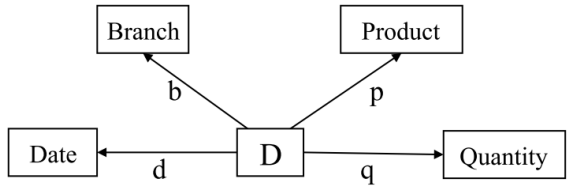

Figure 4.1: Sample dataset for HiFun

Suppose we want to know the total quantity for each branch. HiFun will use grouping, measuring and op in order to get the result.

As shown in Figure 4.2, the HiFun query will be viewed as a set of four functions which we name d, b, p, and q. D is the set of the invoice. From Branch, we find the invoice and find the quantity and do the operation of sum in order to find each quantity for that branch.

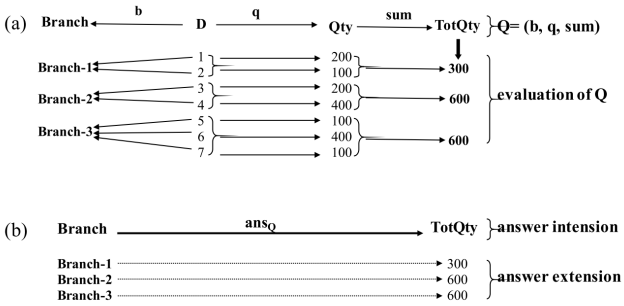

Figure 4.2: A HiFun analytic query and its answer

Grouping. The first step that a HiFun expression will do is Grouping. In the above example, the operation is b. This step will group all the invoices for that branch. Invoices will first be grouped as branches. In the example, the invoice will be grouped from branch1, branch2 and branch 3. The result of this expression will be like:

- Branch 1: 1, 2
- Branch 2: 3, 4
- Branch 3: 5, 6, 7

Measuring. The second step consists in finding the values. In our case, we want to find the quantities. The result will be like:

- Branch 1: 200,100
- Branch 2: 200,400
- Branch 3: 100,400,100

Reduction. The reduction in the expression is done through operators. In our example, the user wants to see the total quantity for each branch, then the operator will be *sum*. Then, the result will be like:

- Branch 1: 200 +100 = 300
- Branch 2: 200+400 = 600
- Branch 3: 100+400+100 = 600

After these 3 steps, we are able to get the desired result. The HiFun expression will be: *Q = (b, q, sum)*

Figure 4.3 illustrates the execution of a HiFun query.

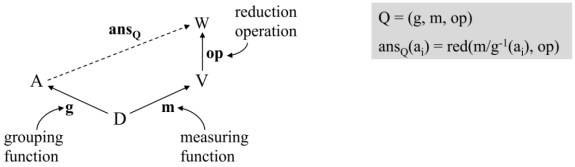

Figure 4.3: A query Q and its answer, ansQ

As said earlier, a HiFun query Q can combine SQL and MapReduce. No matter for what kind of database is used there are two steps needed for a HiFun query. The first step is the mapping step, which means we need to find the key-value pairs. The second step is the reduction step which means the operator that we need to apply to the key-value pairs that have been found in the first step.

A HiFun query Q can be represented as Q = (e, e', op). Figure 4.4 shows a running example for a Hadoop query.

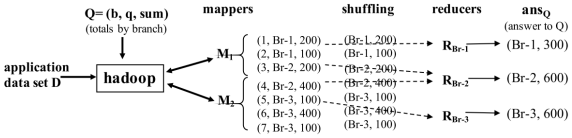

Figure 4.4: Evaluating the query of our running example in Hadoop

We can see ((k(i), vi)) as an example for the key-value pairs from the mapping step for all items (i) in the set D. In this case, there are 2 functions: k: D-> K and v: D-> V. The K and V are domain values. Then op will be the operator for the Reduce job and the query for this expression will be Q = (k, v, op). If the data set is a relational database, we can use SQL to generate the HiFun query Q as well:

Select target(e), op(target(e')) As Result
From T
Group by target(e)

In the example shown above, we can encode the HiFun expression Q in either MapReduce job for a non-relational database but also for a relational database.

HiFun is considered as a powerful language because of the rewrite ability in the SQL and NoSQL or MapReduce by the query writing. The basic writing rule for the language is:

$(g \circ f,m,op) = (g, (f,m,op), op)$

Figure 4.5 illustrates the evaluation process for a HiFun expression. The first is unfolding the query of each expression. The Q expression will be then divided into Q_1 and Q_0. Then, based on the query, we generate the answer for Q. In this example, Q1 has already had its answer stored in the

expression. On the right-hand side appears the recursion of the giving example. The base of the recursion is 1! and since we have three values in the expression then the recursion of the giving expression will be 3!

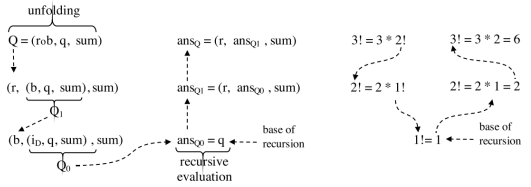

Figure 4.5: Recursive evaluation of the analytic query "totals by region"

Advantages of the HiFun approach

HiFun as a high-level query language for expressing analytic queries over big data sets that are easy to evaluate into group by or MapReduce queries It has a clear separation of the conceptual level and formal rewriting approach which directly access group by and MapReduce jobs.

HiFun uses functional algebra approaches for each of the attributes in the data set and generates query expressions for the attributes. It is able to evaluate user requirements into the query to determine if it is a MapReduce job or a SQL group-by query. The system is able to determine the dataset attributes and generate the query according to the nature of the data attributes.

It is high-level query language in the sense that it is conceptual level. The data set will be shown in the graph no matter if it processes relational or non-relational data attributes. Users no longer need to understand the relationships between all the attributes, they only need to know the number of attributes in the given dataset. The requirements of the user having to understand and have database knowledge is much lower compared to another query language.

HiFun has the ability to rewrite the query language in order to get the result. The data attributes may be stored in different types of databases. In order to get the result, we need to retrieve values in the HiFun language and write the query in order to be able to access different databases.

Concerns of HiFun

There are two concerns that were bought up in (Spyratos & Sugibuchi, 2018) about the HiFun language. The first concern is related to the changing of the dataset D and another concern will be the extension of the functional algebra.

Throughout the research about the HiFun language, the data set that has been implemented is very static. However, for different application running environments, the data set may also change frequently. Due to the change of the data set, the query has to be updated frequently in order to get more accurate results. For example, the dataset that we have used is all about invoices, and the functional queries only target attributes of invoices. However, if the invoices dataset is changed, some of the function queries also need to adapt to the changes. In order to solve this concern, HiFun will propose incremental algorithms in order to generate incremental queries as well.

Another concern that has been discussed in the HiFun research relates to the extension of functional expressions (Spyratos & Sugibuchi, 2018). It is very important that HiFun be able to unify the expressions in SQL and MapReduce in the formal framework. Currently there is no formal common framework for SQL and MapReduce in the HiFun language. In order to solve this, HiFun could include a formal framework for all select, group by and MapReduce jobs. The data will be unified under the data management platform. The formal data management platform will be able to unify all the expressions in SQL and NoSQL. Queries will be generated in a unified system, which in that case will be more standardized.

V. CONLUSION AND FUTURE WORK

In this paper, we have compared different approaches for high-level query languages for querying big data sets.

We have also presented the HiFun query language that has been proposed by (Spyratos & Sugibuchi, 2018).

We are currently implementing a prototype that will allow to test the usability of the HiFun language. It includes (i) a graphical user interface that will allow the user to prepare the query and (ii) a backend system that will process the graphical query and generate access to the data sets, and finally send back the results to the user.

REFERENCES

[1] Bry, F., & Eckert, M. (2006). A high-level query language for events. *Proceedings - SCW 2006: IEEE Services Computing Workshops*, 31–38. https://doi.org/10.1109/SCW.2006.2

[2] CACM Staff. (2017). Big data. *Communications of the ACM*, *60*(6), 24–25. https://doi.org/10.1145/3079064

[3] Che, D., Safran, M., & Peng, Z. (2013). From Big Data to Big Data Mining: Challenges, Issues, and Opportunities. In *Database Systems for Advanced Applications* (pp. 1–15). https://doi.org/10.1007/978-3-642-40270-8_1

[4] Gantz, J., & Reinsel, D. (2013). The Digital Universe IN 2020: Big Data, Bigger Digital Shadows, and Biggest Growth in the Far East. Retrieved from http://www.emc.com/leadership/digital-universe/iview/index.htm.

[5] Guo, H.-D., Zhang, L., & Zhu, L.-W. (2015). Earth observation big data for climate change research. *Advances in Climate Change Research*, *6*(2), 108–117. https://doi.org/10.1016/J.ACCRE.2015.09.007

[6] Kim, G.-H., Trimi, S., & Chung, J.-H. (2014). Big-data applications in the government sector. *Communications of the ACM*, *57*(3), 78–85. https://doi.org/10.1145/2500873

[7] Manyika, J., Chui, M., Brown, B., Bughin, J., Dobbs, R., Roxburgh, C., & Byers, A. H. (2011). Big data: The next frontier for innovation, competition, and productivity. *McKinsey Global Institute The McKinsey Global Institute*. Retrieved from https://bigdatawg.nist.gov/pdf/MGI_big_data_full_report.pdf

[8] Schweikardt, N., Schwentick, T., & Segoufin, L. (2010). *Database theory: Query languages*. *Algorithms and theory of computation handbook*. CRC Press LLC. Retrieved from http://www.lsv.fr/~segoufin/Papers/Mypapers/DB-chapter.pdf

[9] Shaikhha, A., Klonatos, Y., & Koch, C. (2016). Building Efficient Query Engines in a High-Level Language, 853–864. https://doi.org/10.14778/2732951.2732959

[10] Spyratos, N., & Sugibuchi, T. (2018). HIFUN - a high level functional query language for big data analytics. *Journal of Intelligent Information Systems*. https://doi.org/10.1007/s10844-018-0495-6

[11] Stewart, R. J., Trinder, P. W., & Loidl, H. W. (2011). Comparing high level MapReduce query languages. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *6965 LNCS*(Section 6), 58–72. https://doi.org/10.1007/978-3-642-24151-2_5

[12] Strauch, C., & Kriha, W. (n.d.). NoSQL Databases Lecture Selected Topics on Software-Technology Ultra-Large Scale Sites. Retrieved from http://www.christof-strauch.de/nosqldbs

[13] Vanessa Rombaut. (2016). Top 5 Problems with Big Data (and how to solve them). Retrieved May 13, 2018, from https://www.business2community.com/big-data/top-5-problems-big-data-solve-01597918

[14] Wani, M. A., & Jabin, S. (2018). Big Data: Issues, Challenges, and Techniques in Business Intelligence, (October 2016), 613–628. https://doi.org/10.1007/978-981-10-6620-7_59

[15] Zhang, R., Snodgrass, R. T., & Debray, S. (n.d.). Application of Micro-Specialization to Query Evaluation Operators. Retrieved from https://www2.cs.arizona.edu/~rts/pubs/zhang_smdb2012.pdf