BIG PROBLEMS WITH BIG DATA

KRZYSZTOF GOCZYŁA

Faculty of Electronics, Telecommunications and Informatics,
Gdańsk University of Technology
Gabriela Narutowicza 11/12, 80–233 Gdansk, Poland, krissun@pg.edu.pl

(received: 4 December 2019; revised: 20 December 2019;
accepted: 27 December 2019; published online: 18 January 2020)

Abstract: The article presents an overview of the most important issues related to the phenomenon called big data. The characteristics of big data concerning the data itself and the data sources are presented. Then, the big data life cycle concept is formulated. The next sections focus on two big data technologies: MapReduce for big data processing and NoSQL databases for big data storage.

Keywords: Big Data, MapReduce, NoSQL database, Data science
DOI: https://doi.org/10.17466/tq2020/24.1/e

1. What is big data?

In the last few years, it is difficult to find a term from the IT area that has made a bigger career than "big data". After entering "big data" in the search box, Google returns over 7 billion items. How to deal with it? How to separate meaningful information from such that does not carry any useful content, and sometimes is even false. The billions of items suggested by Google are just a very good example of big data. They are in abundance, they grow very quickly, they have different formats and it is not clear what is useful and what is garbage.

Fortunately, it is not that bad. There are sensible definitions of big data and, more importantly, efficient computer technologies for storing big data and extracting useful information from it. Let us start with a definition — one that, in the opinion of the author of this text, most accurately reflects the crucial characteristics of big data:

Big data are collections of information of **very large volumes**, **high speed of growth**, and **great diversity** that require **special methods and tools** to store, process, and analyze them for decision support, discovering new phenomena and optimizing processes.

In the above definition, some of the characteristic features of big data are highlighted. These features make up the neat symbolism of several "Vs",

introduced primarily by IBM as "3V", and then extended by different parties to next "Vs", not necessarily matching the essence of big data, but certainly very catchy in marketing. Let us start with the first "3Vs":

- **V**olume — a huge volume of data. The meaning of the word "huge" is not strictly defined and depends on the specific domain and application. Normally, it is assumed that this denotes tera- and petabytes per one data repository;
- **V**elocity — a high speed of the data inflow, i.e. a fast increase in its volume in a given repository;
- **V**ariety — a high diversity of the data structure and content. It also refers to different formats and ways of generating the same information by different sources.

The next "3Vs" that we consider important refer to very important aspects related to the sources of data and their usability:

- **V**ariability — refers to a possible instability of the data source meant as irregularity of data provision and its inconsistency;
- **V**eracity — refers to the quality of data in the context of its credibility and accuracy;
- **V**alue — determines the actual usefulness of data for analyses and process optimization.

As shown above, the first three "Vs" relate to the technical aspects of big data, while the next three relate to business aspects.

Big data is generated by sources of very different types. It is impossible to list all of the because new types are constantly appearing. Popular types of big data include data from various business transactions, scientific data (astronomical, biological, medical, etc.), physical measurements, data from sensors and devices connected to the Internet of Things, data generated by users of Internet portals (e.g. social networks), data generated by Internet crawlers and many, many others.

The International Data Corporation estimates that the volume of data currently stored in the world's repositories reaches some 50 ZB (1 ZB = 10^{21} B), i.e. about 7 TB of data per each inhabitant of the Earth, and this amount doubles about every 2–3 years. These are mostly static data; only several percent of global data resources are transferred through computer networks at each moment.

2. Big data life cycle

Big data has its own life cycle that determines the big problems related thereto mentioned in the title of this paper. To put it simply, we can define three main phases of this life cycle as follows:

Phase 1. Acquisition, storage, and transformation of data.

This phase involves identifying data sources, including their veracity and variability, storing the data coming from these sources, and performing initial, usually quite simple processing. In this phase, data cleansing is carried out:

detection and removal of errors, standardization of formats, interpretation of outliers, and other operations aimed at preparing data for further analyses.

The data cleansing stage is often underestimated in terms of the effort needed. Data from one source can contain conflicting information or information that is hard to identify uniquely. Our local GUT example is the spelling of places from where the students of GUT come. In the MyGUT system, the name "Starogard Gdański" is stored in 34 different ways (excluding case sensitivity).

Phase 2. Data mining using methods of statistics and artificial intelligence.

In this phase, we strive to discover the new knowledge hidden in the data. This phase strictly depends on the purpose for which the acquired knowledge is to be used and how it is going to be interpreted and exploited in Phase 3. The methods used in this phase are interdisciplinary and fall within the field of data science [1]. A data scientist is an engineer who is not only familiar with data analysis methods and techniques but also can apply these methods in a way that adds value to specific business activity.

Phase 3. Leveraging the results of Phase 2 into a business strategy.

The knowledge acquired so far is used for analysis at the strategic level. To this end, appropriate business intelligence tools are desirable that enable modeling of the current and future business states, enriched with sophisticated and creative data visualization techniques.

The three phases of the big data life cycle vary in nature depending in which area they are implemented. For example, in the business setting of a company, in Phase 1, transaction data is acquired from various places and branches of business activity. In Phase 2, such data is used to calculate the key performance indicators (KPIs) relevant for the specific business as well as to explore trends, correlations, and other relationships among the data. In Phase 3, the results of Phase 2 are presented to the management staff with the possibility of flexible manipulating them to enable the headquarters to make strategic decisions regarding the company's future operations.

In the scientific research setting, let us take astronomical data as an example, Phase 1 consists of collecting images of selected parts of the Universe taken at different frequencies of electromagnetic waves and then cleansing it to eliminate the measurement noise. In Phase 2, astronomical objects and observed astronomical phenomena are identified and labeled with descriptive metadata. In Phase 3, after thorough investigation of the results of Phase 2, next exploration steps are taken, aimed at detailing knowledge about the objects identified in Phase 2.

The remainder of this paper is devoted to selected popular technologies for storing and transformation of big data, i.e. for the implementation of Phase 1 of the big data lifecycle. Readers interested in the next phases are referred to the extensive literature in the field of data science and business analysis.

3. MapReduce for big data processing

Generally, the key to big data management is massive parallelism and distribution of processing and data. This is due to the fact that centralized systems do not reveal sufficient horizontal scalability necessary for effective processing of large volumes of data. Moreover, big data management requires the introduction of quite different data processing and storage paradigms than in "classic" data processing systems. One of the more interesting new paradigms of parallel and distributed computing is MapReduce, introduced by Google in the early 2000s and then intensively developed by various organizations.

The general idea behind MapReduce is simple: calculations performed in parallel on multiple nodes take *key /value* pairs as an input and produce other *key /value* pairs, depending on the problem. This is performed in two phases. In the Map phase, a pair (k_1,v_1) is transformed into a pair (k_2,v_2). In the Reduce phase, calculations are performed on data grouped according to identical values of keys k_2. Generally, in this phase, each pair $(k_2, \text{List}(v_2))$, is transformed into a pair $(k_2, \text{List}(v_3))$.

As an illustrative example, let us consider the problem of calculating the number of occurrences of each word in a large document collection. The Map phase can look like the code fragment below (formulated in a self-explanatory pseudocode):

```
Map (String key, String value):
  for each word w in value:
    Emit (w, 1);
```

where *key* is the name of the document, and *value* is the content of the document. The result of a single execution of the Map procedure is a set of pairs (*word* , 1). Of course, one parallel node can execute Map multiple times, for many documents.

In the Reduce phase the following procedure is executed:

```
Reduce (String key, List values):
  int result = 0;
  for each v in values:
    result += v;
  Emit (key, result);
```

where *key* is a given word, and *values* is the list of counters for the *key* produced by Maps (in this example this is just a list of 1s).

Note that to transfer the results from Maps to Reduces, an intermediate layer is needed that groups all the results of the Map phase according to the same values of the key (in this case for the same word) and passes them to the Reduce phase. Contemporary platforms implementing the MapReduce paradigm, such as Apache Hadoop, free programmers from the necessity to develop this layer, as well as from all the coordination of parallel and distributed processing performed on many nodes, in a large distributed environment prone to node and network failures.

As a rule, Map and Reduce operations should be simple. Many algorithms based on this paradigm have been developed, including those for the following problems: searching document collections for a given text pattern, measuring the popularity of websites based on their access logs, building a graph of website links, building inverted files (text indices), as well as for more advanced problems, such as sorting documents or implementation of relational algebra operators. Figure 1 depicts the schema of the MapReduce framework.

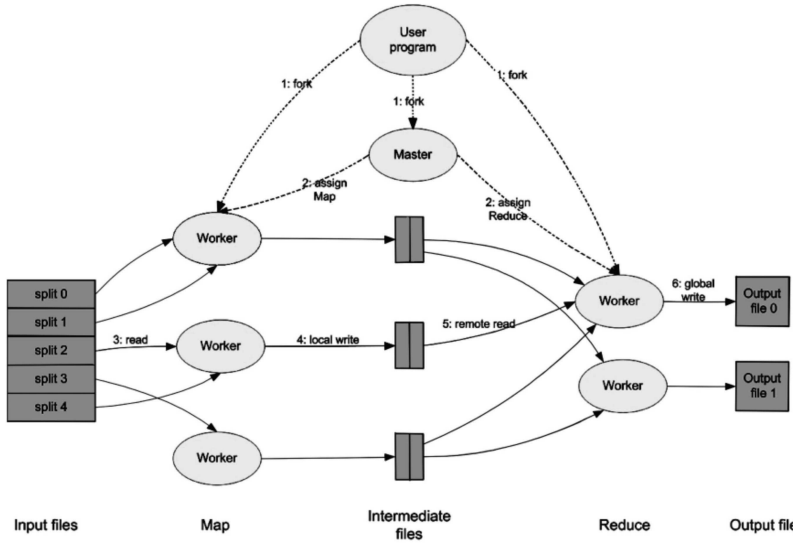

Figure 1. A schema of the MapReduce framework. The ovals represent nodes (Workers), the rectangles represent files in a distributed file system. The Master node coordinates the work of the Map and Reduce nodes. Arcs show the order of operations (source: [2])

4. NoSQL databases for big data storage

The requirement for massive parallelization and distribution of processing strongly affects the world of databases. The classic **ACID** paradigm (**A**tomicity, **C**onsistency, **I**solation, **D**urability) to which the relational databases conform appears too rigid for management of large data, particularly in environments composed of hundreds or even thousands of processing nodes with highly partitioned and replicated data. The very concept of a database transaction must be modified due to the possibility of node failures or communication errors in large distributed environments. Moreover, typical big data solutions are not transactional systems that require absolute accuracy, even at the expense of availability. Therefore, the rules of the ACID paradigm are replaced by the **BASE** paradigm. This paradigm does not assume that the distributed database will always be in a consistent state. The priority is the availability of the system (**B**asically **A**vailable), even at the cost of global consistency (**S**oft state).

Nevertheless, it is important that the database system as a whole constantly strives to reach the data consistency state (**E**ventually consistent) meant as global correctness and integrity of data.

The BASE paradigm is supported by some theory called the **CAP** Theorem (**C**onsistency, **A**vailability **P**artition tolerance). Without going into much detail, the CAP Theorem claims that in a distributed database system it is not possible to maintain both system availability and data consistency in a situation of network fragmentation. If some nodes have failed or become isolated as a result of a communication failure (i.e. we are faced with network partitioning), then the system is made inaccessible till the time when global consistency has been restored or becomes potentially inconsistent if other, healthy, nodes remain available. If an application required accessibility and consistency at any time of system operation, we must give up distribution and use a centralized database system.

An important part of this non-classical database world are NoSQL databases. The essential features of NoSQL databases that distinguish them from relational databases are:

- no strict database schema (*schema on read* instead of *schema on write*: types of data stored in a database are determined by the application upon retrievals rather than by the system upon insertions);
- complex, nested data structures (no normalization);
- limited indexing capabilities;
- limited or no conformance to ACID rules, BASE rules instead;
- high horizontal scalability;
- efficient management of big data using data distribution and parallel processing;
- no (or very limited) declarative query language; instead there are APIs to popular languages: Java, JavaScript, Ruby, Python, Gremlin, Erlang, and others.

Recently, there has been a great variety of NoSQL systems, mainly open source systems. Below we follow the classification proposed in [3]. However, remember that specific systems can be mixtures of different classes or even unique specimens of their own.

In the **key-value systems**, data is stored in the form of pairs (*key, value*). The value may be of any type, e.g. any sequence of bytes interpreted programmatically. The key may be hierarchical, as in Figure 2. Some representatives of this class are Riak, Redis, Voldemort, and Oracle NoSQL (commercial).

In the **column systems** data is stored by columns, not—as is usually the case in relational databases—by rows. For efficiency reasons, data items from a single column from different rows are stored close to each other, on the same disk page or adjacent disk pages. Each row can have a different set of columns. Columns are grouped into column families to enable storing collections. The total number of columns in one table may be very large. Due to the nature of big data, for a given row, usually, most of the columns have no specified values. Thanks to

Keys	Values
...	...
Pomerania	*data about Pomerania*
Pomerania/Gdańsk	*data about Gdańsk in Pomerania*
Pomerania/Gdynia	*data about Gdynia in Pomerania*
...	...
Mazovia	*data about Mazovia*
Mazovia/Warsaw	*data about Warsaw in Mazovia*
Mazovia/Warsaw/Mokotów	*data about Mokotów in Warsaw*
...	...

Figure 2. An example of data in a key-value database with information on Poland

the way the data is stored, there is no necessity to store NULL or UNKNOWN values. Popular column systems are HBase, Cassandra, and Hypertable.

The **document systems** are mainly devoted to storing text documents. Any document has an identifier and content – a complex, flexible structure allowing nesting other documents. In Figure 4 a JSON document that describes a town is presented. Popular representatives of this class are MongoDB and CouchDB.

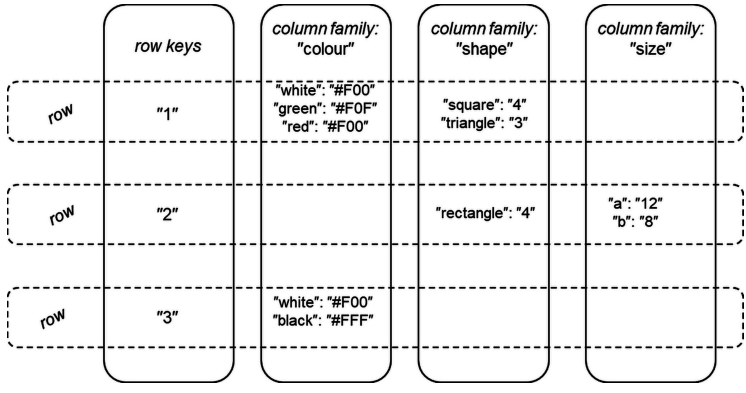

Figure 3. An example of data in a column database. Attribute data items are depicted together with their internal representation. Note that most fields are empty

A **graph database** is a set of nodes and relations between nodes represented as arcs. Nodes and arcs can have their own properties in the form of *key/value* pairs. Popular graph databases are Neo4J and Apache Spark GraphX.

The classification presented here is illustrative rather than comprehensive and complete. There is a large variety of NoSQL databases, and new ones with very different degrees of maturity are constantly appearing. Hybrid solutions composed of a NoSQL database and a relational system are also used. In such a configuration, a NoSQL database is the front-end of the system used to load raw data and pre-process it (filtering and cleansing). Data that has been processed

```
>printjson (db.towns.findOne( {"_id" : ObjectId("a0454d06bf7704eaaff32129")}))

                        {
                "_id" :  ObjectId("a0454d06bf7704eaaff32129"),
                              "country" : {
                              "$ref" : "countries",
                "$_id" : ObjectId("a0454df740f340296ceaaf21")
                              },
                              "name" : "Portland",
                              "state" : "OR",
                "last_census" : "Thu Sep 20 2007 00:00:00 GMT",
                              "population" : "582000",
                              "mayor" : {
                              "name" : "Sam Adams",
                              "party" : "D"
                              },
                              "famous_for" :  [
                                      "beer",
                                      "food"
                                      ]
                        }
```

Figure 4. A printout from a document database. The document identified by *id* is stored in *db* database in *towns* collection. It contains a reference to another document in *countries* collection

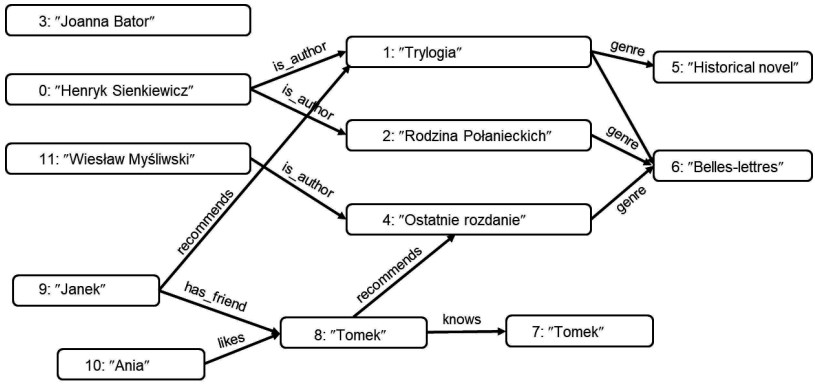

Figure 5. A graph database representing a social network. The arcs are named according to the meaning of relations between the objects represented by uniquely numbered nodes

in such way is then transferred to a relational database — the back-end of the system for structured storage more appropriate for exploration processes.

5. Conclusions

As mentioned before, the field of big data is extremely broad, multi-layered, and interdisciplinary in both technological and scientific terms. Interested readers are referred to literature studies, as well as to individual experiments with tools mentioned in this paper and others, not explicitly mentioned here. It is clear that professional environments for big data processing require powerful computing clusters, however, familiarization with them is possible using generally available cloud computing services and even mid-range personal computers.

For the reasons mentioned above, only some representative references on big data — from more than 7 billion — are quoted in this paper, those that seem especially valuable for exploring the world of big data. [1] is an excellent introduction to data science. The presentation of specific methods of data analysis

is illustrated with practical examples from the business world. In [2] a MapReduce platform with support for the intermediate layer between Map and Reduce was presented for the first time. [3] is a comprehensive practical introduction to NoSQL databases, with examples of using them in popular computing clouds.

The next three items in the References, although not referenced explicitly in the text, give a deep insight into two issues that seem crucial for the further development of the big data technology and the big data idea. [4] provides an exhaustive overview of scalability techniques for big data processing, both batch and streaming ones, as well as big data storage. The authors of [5] discuss the impact of big data on modern societies — an aspect that cannot be overlooked in the context of the opportunities that technology development brings as well as the risks associated with it. Finally, [6] presents theoretical aspects of the processing and extracting information from big data in a clear yet precise way and is recommended for theoretically oriented readers.

References

[1] Provost F and Fawcett T 2013 *Data science for business*, O'Reilly Media Inc.
[2] Dean J Ghemawat S 2013 *(Google, Inc.): MapReduce: Simplified Data Processing on Large Clusters*, 6th Symposium on Operating Systems Design and Implementation
[3] Redmond E and Wilson J R 2018 *Seven databases in seven weeks: a guide to modern databases and the NoSQL movement - Second Edition*, The Pragmatic Programmers
[4] Marz N 2015 *Big Data: Principle and Best Practices of Scalable Real-Time Data Systems*, Manning Publications Inc.
[5] Mayer-Schőnberger V and Cukier K 2014 *Big Data: A Revolution That Will Transform how We Live, Work and Think*, Harcourt Publishing
[6] Leskovec J Rajaraman A and Ullman J D 2014 *Mining of Massive Datasets*, Cambridge University Press