An Experimental Model of the Collect/Report Paradigm

Krassimira Borislavova Ivanova
University of Telecommunications and Post, 1700, Bulgaria, Sofia, 1 Acad. St. Mladenov str.
e-mail: krasy78@mail.bg

Abstract. In the Big Data community, the “Map/Reduce Paradigm” is one of the key enabling approaches for meeting the continuously increasing demands on computing resources imposed by massive data sets. Today it is implemented in many open source projects. The popularity of Map/Reduce is due to its high scalability, fault-tolerance, simplicity and independence from the programming language or the data storage system. At the same time, Map/Reduce faces a number of obstacles when dealing with Big Data. A possible solution of them may be the Collect/Report Paradigm (CRP) and Natural Language Addressing (NLA) approach. It is suitable for storing Big Data in large information bases located on different storage systems – from personal computers up to cloud servers. An experimental Model of the CRP is presented in this paper. An experimental implementation of the CRP to process and store data is outlined. The structures of the input and output data are in the form of RDF triplets. The ease of implementation of this model and the benefits of its use are discussed.

1. Introduction

Big Data is a large amount of data that cannot be processed, analyzed and stored through traditional processes and methods [1], [2]. Three main characteristics define Big Data: Volume (the amount of data from myriad sources), Variety (the types of data: structured, semi-structured, unstructured), and Velocity (the speed at which big data is generated). In addition, three more characteristics are taken in account - Veracity (the degree to which big data can be trusted), Value (the business value of the data collected), and Variability (the ways in which the big data can be used and formatted).

These characteristics cause corresponded problems for storing Big Data; therefore we need a new class of capabilities to augment the way things are done today to provide better line of site and controls over our existing knowledge domains and the ability to act on them. The permanent collection of data complicates their storage and processing. It is no longer possible to store all data in one place; they must be highly segmented to facilitate storage and processing. We can no longer use traditional methods to process data that is not in one place, but distributed too many separate machines. There is a need for new generations of programs that can run on multiple machines simultaneously and efficiently.

Popular approach for representing Big Data is Resource Definition Framework (RDF), or equivalent approaches, such as XML and OWL. Examining currently existing RDF stores we found that they are using relational and object-relational database management systems. Storing RDF data in a relational database requires an appropriate table design. The graph-oriented approach for storing Big Data is one of the preferred. There exists a gradual transition from relational to non-relational models for organizing Big Data [3].

In the Big Data community, the “Map/Reduce Paradigm” is one of the key enabling approaches for meeting the continuously increasing demands on computing resources imposed by massive data sets.
Today it is implemented in many open source projects. The popularity of Map/Reduce is due to its high scalability, fault-tolerance, simplicity and independence from the programming language or the data storage system. At the same time, Map/Reduce faces a number of obstacles when dealing with Big Data. A possible solution of them may be the Collect/Report Paradigm (CRP) and Natural Language Addressing (NLA) approach realized in “Multi-Dimensional Numbered Data Base” (MDNDB™). It is suitable for storing Big Data in large information bases located on different storage systems – from personal computers up to cloud servers [4].

An experimental model of the Collect/Report Paradigm is presented in this paper. An application that uses an exemplary implementation of the CRP to process and store data is outlined. The structures of the input and output data are in the form of RDF triplets. The ease of implementation of this model and the benefits of its use are discussed.

Paper is organized as follow: Chapter 2 shortly represents the Map/Reduce paradigm; Chapter 3 contains a brief introduction to Collect/Report Paradigm; Chapter 4 describes the experimental model of the Collect/Report Paradigm and experiments with it. The paper ends with conclusion and directions for future work.

2. Map/Reduce Paradigm
Map/Reduce is a model created by Google for processing large volumes of data [5], [6]. The model allows their parallel processing regardless of how segmented they are and how many places they are stored. The idea for this model came with the need to process a large set of data. Data that cannot be stored in one place, but must be segmented. Map/Reduce consists of two functions. The first function - "Map", performs filtering and sorting of data, while the second - "Reduce", performs grouping and merging.

The "Map" function is performed by each process on the data assigned to it for processing. A traditional example of performing the function is counting words in text. The Map function has the task of creating key-value pairs. Each key is a unique word, and each value is the number of the given key in the text. After processing, the output from each machine is sent to the corresponding "Reduce" function. The "Reduce" function receives the result from the Map function in the form of a key-value, which are sorted and merged. Collecting the data for all words, the function sums the values and creates new key-value pairs (Figure 1).

![Figure 1. Map/Reduce Paradigm](source https://slideplayer.com/slide/14040052/).

One of the main differences between traditional processing and Map/Reduce is the storage of the data to be processed. With traditional processing, we need all the data in one place. In contrast, in Map/Reduce, they are split and stored in different repositories. The advantage of the Map/Reduce Paradigm is that it runs on each machine independently and in parallel. We can have a cluster of machines, each of which store a small amount of data and execute its own Map/Reduce process, independently of all the others. Its parallel implementation allows the strong fragmentation of data for faster processing and easier storage. Its implementation requires the implementation of the Master/Slave Paradigm [7]. It helps data to be segmented and processed by multiple threads on a single core, multiple cores on a single processor, multiple processors on a single machine, or even multiple machines on a
network. The use of this model allows constant expansion of the system, which in turn drastically reduces the processing time and allows the processing of a very large amount of data.

In the Master/Slave model, we have one master (Master) who assigns tasks to several subordinates (Slaves) (Figure 2).

![Master-Slave model](image)

**Figure 2.** Master-Slave model.

### 3. Collect/Report Paradigm

The Collect/Report Paradigm (CRP) (Figure 3) is a new approach to data storage and access that builds on the Natural Language Addressing (NLA) model [8]. NLA does not rely on indexing and relations.

NLA consists in assuming the internal computer codes of letters as co-ordinates in multi-dimensional information space. Different words and phrases have different lengths and require using of addressing with variable length of the co-ordinate arrays, i.e. to have variable dimensions in one and the same time. The possibility to use coordinates is good for graph models where it is possible to replace search with addressing.

Like Map/Reduce, the Collect/Report is designed to handle large amounts of data. CRP accepts input data in the form of RDF triplets (subject, relation, object). This allows easy localization of the memory location where they need to be saved. The subject and the relation are responsible for the address in the memory and only the object is recorded, which reduces the amount of recorded data. The model allows recording and collection to take place on all machines simultaneously.

The Collect/Report works with two main functions: “Collect”, which has the role of collecting and saving the necessary data, and “Report”, which must provide them when needed. In CRP, individual machines are associated with certain layers of data.

The “Collect” function "listens" to the flow of data that passes and "collects" only those that are associated with the current machine. Once collected, the data is saved in the layer to which the machine is associated. This allows a huge amount of data to be recorded in parallel. The strong segmentation of the cores/machines that "listen", the constant flow of data, favors the easy expansion of the architecture.

The "Report" function also "listens" for queries as only machines that have corresponded data report. The kernels are activated only when they receive a request for the corresponding data. After a request is received, the kernels check the required address and return the data stored there. Each kernel is preconfigured to serve a segment of shared memory. If there is no data written in the searched address, the kernel does not return anything.

The machines listen to the data stream, take and record only those data that are related to the part of the memory they manage. When the data is requested, the machine, which has in its memory data about the searched keywords or phrases, reports. The output data from the CRP is in RDF triples format.

Main advantages of CRP are: (1) All nodes independently in parallel do collecting information. It is possible one node to send information to another; (2) Reporting information is provided only by the nodes which really contain information related to the request; the rest nodes do not react, they remain silent; (3) Input data as well as results are in RDF-triple or RDF-quadruple format (Figure 3).
3.1. Comparison of Map/Reduce and Collect/Report

Map/Reduce Paradigm and Collect/Report Paradigm are two different approaches to working with very large data sets.

Comparing the two paradigms we can conclude that:

- Map/Reduce results in key and value pairs. Collect/Report is designed to work with data presented in RDF triples format (subject, link, object) and the results are in the same format.
- In the Collect/Report model, data is provided only by machines that actually contain data related to the query. The others do not react. In Map/Reduce, all machines send a response.
- An important advantage of Collect/Report is the reduction of traffic and readiness to extract data within microseconds.
- Another important difference is that with Map/Reduce the calculations and data must be completely independent. The Collect/Report model assumes that all data is interconnected and can be processed together, taking into account all interconnections.
- Map/Reduce accepts complete data, which it converts. Collect/Report accepts data, which it records permanently and reports if necessary.
- Collect/Report reduces the amount of saved data, Map/Reduce does not.

The two models for working with Big Data are fundamentally different. They rely on different input formats, have different processing functions and different result formats. However, they could complement each other, be used together to achieve an even better result.

It is possible to integrate the Map function from the Map/Report Paradigm to the Collect/Report Paradigm. The second expects RDF triplets as input, which would require the Map function to be modified to return the required format. Ultimately, this would lead to a better model that works with a larger data set.

4. Experimental model of the Collect/Report Paradigm

The program realization of the experimental model of the CRP was done by Konstatin Savov under the supervision of the author.

The application server uses the ASP.NET Core platform version 3.1. This modern platform allows the creation of a Web-based application almost instantly.

The communication of the user with the server is carried out thanks to the platform developed by Facebook - ReactJS. Microsoft's TypeScript library is used with ReactJS. It has the role of typing pure JavaScript code, making it much more readable and easy to understand.

The application is uploaded and configured to run on a server in Microsoft's cloud services - Azure. SQL Server is used for data storage.

4.1. Architectural diagram

The application communicates with the end user with data in the form of RDF triplets. The application interface facilitates the user in entering and visualizing data.

The Frontend part (Figure 4) communicates with the server on behalf of the client, sending him data to store or to find certain definitions.
The Backend part (Figure 4), in turn, performs the necessary calculations based on NLA and Collect/Report, and sends the coded information to the appropriate cores.

![Application architecture](image)

**Figure 4.** Application architecture.

4.2. Data input

The user enters the data in the form of RDF (Resource Description Framework) triples - subject, relation, object. They enter the application, where they are transmitted to all cores. The transmission takes place through the centralized program, through which all requests pass. This makes it easy to add cores without changing the architecture.

The role of the centralized program, which controls the communication with the cores, is to optimize the work as it makes all the calculations that would be needed for each of the cores.

After submitting the data, each kernel performs the appropriate checks on its configurations and decides whether to process them. Kernels use the subject and the relation to generate an address to memory where the data from the object should be written. The data is converted to an address using coding in the ASCII standard.

When the kernel is configured to work with the given address, it uses the already converted data to find the exact cell in memory where to write it. In our implementation, the memory cell is simulated by a column in the database.

The data is always saved, regardless of whether there is already a definition of the specified address.

Upon completion of processing, the kernel returns information that it has done the work. This makes it possible to monitor individual cores.

4.2.1. Example. Let's use the application to write the sentence - "Sky is blue". The information entered is divided as follows: "Sky" is the subject, "is" is the relation, and "blue" is the subject we have to store.

As we enter the sentence, we see that the data has been successfully processed by the 3rd core. This is because, in the current implementation, the program adds the length of the subject and the relation, and sees which core is responsible for the corresponding length. The core 3 is configured to be responsible for all data with a total subject length and relation up to 5 to 6 characters, inclusive (Figure 5). Every physical core processes one logical node.
In order to facilitate the tracking of the recorded data, there is an implemented page that retrieves from the entire database for all records in it. Because most of the data is encoded in the form of an address, it is not readable. This helps in case of leakage. In the photo below we can see an example of freshly received information.

The data associated with the “subject” and “relation” is encoded while the object is recorded directly. If we imagine that the encoded subject and relation together form an address in memory, then only the object would be written in this cell (Figure 6).

| Address: (83107121, 105115), Value (blue) |

4.3. Implementation of the cores
Each of the seven cores inherits the following interface. This ensures that the cores will have an import, export and a name to associate with them.

This makes adding extra cores very easy. After adding as many additional cores as we want, the program automatically detects them and uses them on the next call.

Each time the program is called it finds all the cores that implement the above interface and passes them to the part of the code responsible for calling the cores and then collecting and structuring the results.

Using this kernel model gives us direct access to write and search logic. All that needs to be configured when adding new cores is the address for which the kernel is responsible and its name (for tracking purposes).

4.4. Implementation of the import
The import is performed as follows:
1. The program receives a new request from the user and the data is in the form of RDF triplets.
2. Assemble all cores that implement the above interface.
3. The subject and relation are transformed into an address.
4. The data is submitted to all cores. At the time of the call, the kernel that is called and the corresponding process that executes it are recorded. This makes it easier to collect the results and keep track of when all cores have finished processing.
5. The program waits for the completion of the execution of all cores.
6. The kernel, which is configured to be responsible for the given part of the memory, performs the calculations and returns a result. The others do not return anything.
7. After completing all cores, the data is taken from the process and a result is formed together with the name of the core that performed the recording.
8. The result is returned to the Frontend part of the program to be presented to the user.

4.5. Data search
When the user wants to find certain information, he must enter Subject and Relation. After sending them to the centralized server, it sends the entered data to all cores to start processing. Again, each kernel performs checks to verify that it is responsible for the given of address couple to which the current one belongs.

Kernels that are configured for other addresses ignore subsequent checks and return nothing - "silent".

The kernel, configured to process the address, checks for a record in the given memory cell. If there is a record at the required address, it is returned from the kernel to the central program, which presents the result to the user.

4.5.1. Example. Let's look for the information entered above. We enter the subject and the relation - "Sky", "is" and the same core corresponds to "blue" (Figure 7).

![Figure 7. Example of data search.](image)

4.6. Implementation of search
Request is very similar to imports. It has the same implementation steps:
1. The program receives a request.
2. All kernels are called.
3. The information entered by the user is submitted to them.
4. The kernel, which is configured to be responsible for the given part of the memory, performs the calculations and returns the result that the user is looking for. The others do not return anything.

Just as with import, when calling kernels, information about their state is stored in the form of a key-value (kernel-process). This facilitates the collection of data after the completion of all calculations.

5. Conclusion
The current implementation of the Collect/Report Paradigm has shown its applicability and the ease with which it could be implemented in any program that processes a large amount of data.

The Collect/Report Paradigm in combination with NLA (Natural Language Addressing) finds strong application when working with large volumes of data. Storing and processing are major difficulties for typical databases, which must index all data before they can be accessed. By excluding indexes, we achieve a constant complexity O(1) of recording and searching. The only limitation would be the speed
of the physical device itself. Removing indexes also reduces the memory that must be allocated to preserve them.

The CRP opens up new possibilities for easy data processing and storing, and easy upgrades when needed. Compared to classic databases and processing methods that are difficult to upgrade, CRP allows easy and efficient expansion in the future. This would also contribute to financial relief in data expansion.

6. Directions for future work
The next step in the development of the ideas presented in this paper is the development of a project for industrial software implementation and conducting initial program experiments in order to select good algorithms for individual functions.

At the next stage, we can proceed to the implementation of the Collect/Report Paradigm in full with the ability to work in cloud data warehouses.

References
[1] Obama Administration unveils 2012 Big Data initiative: announces $200 million in new R&D investments Executive office of the president http://www.whitehouse.gov/sites/default/files/microsites/ostp/big_data_press_release_final_2.pdf
[2] Fact Sheet 2012 Big Data across the federal government Executive office of the president http://www.whitehouse.gov/sites/default/files/microsites/ostp/big_data_fact_sheet_final.pdf
[3] Ivanova K, Vanhoof K, Markov K and Velychko V 2013 Introduction to storing graphs by NL-addressing IJ ITA 20(3) 263-284
[4] Markov K, Ivanova K, Vanhoof K, Velychko V and Castellanos J 2015 Natural Language Addressing (Hasselt, Kyiv, Madrid, Sofia: ITHEA®) p 315
[5] Dean J, Ghemawat S 2008 MapReduce: simplified data processing on large clusters Comm. ACM 51(1) 107-113
[6] Grolinger K, Hayes M, Higashino W, L’Heureux A, Allison D S and Capretz M A M 2014 Challenges for MapReduce in Big Data Proc. of the IEEE SERVICES (Alaska, USA)
[7] Shrestha R 2017 High availability and performance of database in the cloud - traditional master-slave replication versus modern cluster-based solutions Proc. of the 7th IC on Cloud Computing and Services Science
[8] Markov K and Ivanova K 2015 General structure of Collect/Report paradigm for storing and accessing Big Data IJ ITA 22(3) 266-276