Data Locality in Hadoop

Justyna Kałużka, Małgorzata Napieralska, Oscar Romero, and Petar Jovanovic

***Abstract*—The Apache Hadoop framework is an answer to the market tendencies regarding the need for storing and processing rapidly growing amounts of data, providing a fault-tolerant distributed storage and data processing. Dealing with large volumes of data, Hadoop, and its storage system HDFS (Hadoop Distributed File System), face challenges to keep the high efficiency with computing in a reasonable time. The typical Hadoop implementation transfers computation to the data. However, in the isolated configuration, namenode (playing the role of a master in the cluster) still favours the closer nodes. Basically it means that before the whole task has run, significant delays can be caused by moving single blocks of data closer to the starting datanode. Currently, a Hadoop user does not have influence how the data is distributed across the cluster. This paper presents an innovative functionality to the Hadoop Distributed File System (HDFS) that enables moving data blocks on request within the cluster. Data can be shifted either by a user running the proper HDFS shell command or programmatically by other modules, like an appropriate scheduler.**

***Index Terms*—Distributed File System, Big Data, Apache Hadoop, HDFS**

I. Introduction

DATA means terms and information - transferred and processed for further calculations and analysis. Development of modern technologies and digitalization of everyday life cause the growing amount of data generated by people but especially by computers. One can talk about the quantity which was impossible to store and process even few years ago, mainly because of the hardware limitations [1]. Therefore, there is a need for creation and continuous development of technologies which enable managing and analysing these large data quantities. To describe this phenomenon the term *Big Data* is currently used.

A. Big Data

Term Big Data typically refers to the large datasets of size exceeding the storage ability of the typical relational databases as for example in areas like as health care, business and sciences. However, the definition changes amongst sectors and with development of newer - more capable - tools and technologies. The "5 V's" concept typically enumerates the following features [2]:

1) **volume** refers to the data quantity and its massive growth in recent years;
2) **velocity** is related with the speed at which data is generated but also the increasing velocity in which data streams are arriving for processing;
3) **variety** describes the lack of a common unified structure for modelling the data;
4) **veracity** refers to the relatively poor quality of stored data due to its inconsistency and incompleteness;
5) **value** i.e. usefulness of collected information and ability to turn it into desired worth [3], [4].

The natural consequence of market tendencies is that the top technological companies and research institutions are continuously working on solutions how to meet requirements of storing the big volumes of data and processing them in reasonable time to maximize the benefit.

B. Apache Hadoop

Milestone in this Big Data research was set by an open-source software framework called Apache Hadoop, which enabled the distribution of storage and data processing across the computers cluster - set of the connected machines, also called nodes. Files are divided into smaller parts (chunks, blocks) and can be replicated on more than one machine for providing more reliable and fault-tolerant data processing. One of the Hadoop's principles states that "Moving Computation is Cheaper than Moving Data", meaning that it is more efficient to execute tasks close to the node with needed data rather than moving the data itself, especially in case of huge blocks [5].

Hadoop is a free and fully open-source framework under the license of Apache Software Foundation, i.e., available to be modified and enhanced by anyone. The Apache projects are identified as developed in collaborative process by volunteers (often referred as the open-source community) and with open and pragmatic software license. Open source community is very varied but usually and by default helpful, and open to the new contributions. Information about Hadoop and its components is always free and available online - not only documentation but also numerous tutorials, scientific papers and other studies.

Although Hadoop is an effective way to store and process huge amounts of data in a reasonable time, it obviously has some bottlenecks. One of them refers to so called *data locality*. After a new task from a user is recognized by the system, it is processed and system recognizes which of data blocks are needed for that task. By default, the local data access is favored over this from the remote nodes.

Although one of the Hadoops basic concepts is to move computation to data, framework handles data shipping with its built-in scheduler. Blocks of data are transferred just before they are needed. An alternative way, which would significantly increase the system performance, is to transfer data in-advance.

J. Kałużka is with Skyscanner Ltd, Barcelona, Spain (e-mail: justyna.kaluzka@skyscanner.net).

M. Napieralska is with the Department of Microelectronics and Computer Science (DMCS), Lodz University of Technology, Łódź, Poland (e-mail: mnapier@dmcs.pl).

O. Romero and P. Jovanovic are with the Departament d'Enginyeria de Serveis i Sistemes d'Informació (ESSI), Universitat Politècnica de Catalunya, Barcelona, Spain (e-mails: {oromero, petar}@essi.upc.edu).

ISSN 2080-8755

Moving data blocks between nodes would enable studies on more advanced scheduling algorithms. In-advance data shipping could allow a user to manually relocate data, rather than rely on default arrangement, and take advantage later from its locality. Currently, Hadoop automatically organizes blocks across cluster without the user interference. Therefore designing and developing functionality of forced or instructed replication, is very challenging and innovative [6].

II. Apache Hadoop Architecture

The first official release of Apache Hadoop framework was presented in December 2011 and it consisted of so called Two Pillars of Hadoop 1.x - file storage HDFS and MapReduce on top of it. It considerably popularized MapReduce concept and presented the potential of distributed data processing.

From the architectural point of view, Hadoop uses a master/slave architecture, where one device has control over others. Both in MapReduce and HDFS components one can distinguish masters and slaves (Fig. 1) [7], [8].

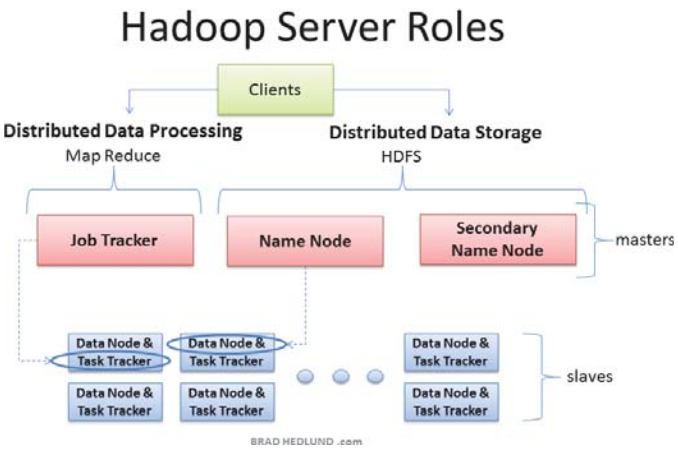

Fig. 1. Hadoop master/slave architecture [7]

With switching to the next major release, Hadoops primary components were re-written to add new functionalities. Fig. 2 presents architecture changes between Hadoop 1 and 2. The main difference was dividing MapReduce functionalities and decoupling separate module YARN, which took responsibility for managing resources. Also HDFS architecture was slightly improved. These changes eliminated such limitations of Hadoop 1 like problems with horizontal scalability and cluster restrictions (only 4000 nodes) [5].

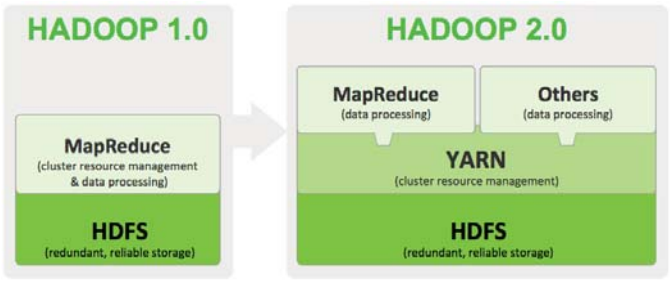

Fig. 2. Hadoop 1 vs Hadoop 2 architecture [9]

Following paragraphs describe the main components of Apache Hadoop architecture.

A. MapReduce

MapReduce is the programming paradigm enabling the massive scalability across the cluster nodes. Developers can write applications to process big amounts of raw data on large clusters with a parallel, distributed algorithm that improves speed and reliability of the system. Although the Apache Hadoop project is just one of the MapReduce implementations, MapReduce plays the key role in the framework. The whole concept assumes the same data flow and stages since its introduction in 2004 [10].

MapReduce algorithm consists of two stages. Jobs submitted to the cluster are called also respectively map and reduce tasks [11].

- *Map stage*
  Input data is read from data storage and divided into blocks. Each block is processed by a map task, line by line. Then the map function submitted by a user generates the output data (in form of key-value pairs). Intermediate data collected in buffer is sorted, written to local disk as many file spills and merged into a single map output file.
- *Reduce stage*
  After transferring data to the proper node, outputs of different mappers are grouped by the previously defined key. User reduce function produces the final data which is later compressed and written as output to HDFS.

MapReduce is widely popular because of its scalability, allowing processing huge amount of data stored in one cluster and being relatively easy to use - developers can write applications in any of popular programming languages (like Java, Python, or Ruby) to run them them as the MapReduce jobs.

B. YARN

Apache Hadoop YARN (Yet Another Resource Negotiator) was introduced with Hadoop 2 and took some of the MapReduce functionalities. This cluster management technology can be described as a large-scale, distributed operating system used in Apache Hadoop framework that separates resources and scheduling management from the data processing part.

In Hadoop 1 JobTracker in MapReduce was responsible for resource management, tracking resources consumption and job life-cycle management. Therefore the fundamental idea of YARN is based on splitting these functionalities into global ResourceManager (responsible for resources) and per-application ApplicationMaster (job scheduling/monitoring).

The main ResourceMananger and per-node slave NodeManager are aimed to manage resources for executing applications in a distributed manner. The per-application ApplicationMaster negotiates resources with the ResourceManager and collaborates with NodeManager in order to execute and monitor the tasks.

The ResourceManager is responsible for handling all available cluster resources among applications using two components - Scheduler and ApplicationsManager. The first one

allocates resources based on the applications needs but without monitoring, tracking status or restarting the failed tasks; the latter - accepts job-submissions directing them to the specific per-application ApplicationMasters. The per-machine NodeManagers are responsible for monitoring their resources usage, tracking and reporting this information to the ResourceManager [5], [12].

C. HDFS

Hadoop Distributed File System is the primary distributed storage used in Hadoop - scalable and reliable, designed especially for large clusters of commodity servers, aiming to be fault-tolerant and running on low-cost hardware [5].

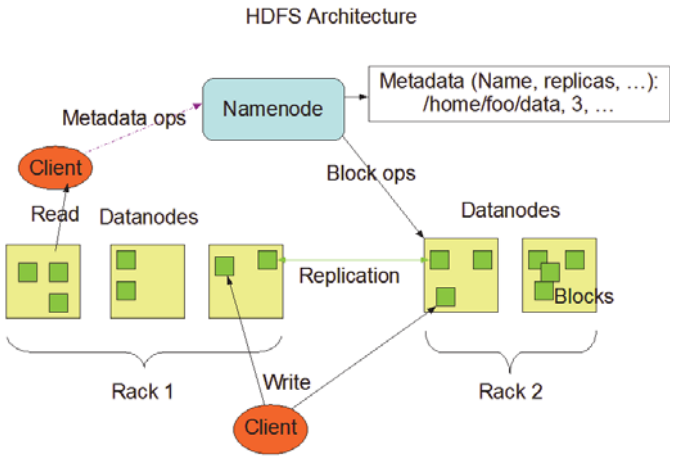

Fig. 3. HDFS architecture [5]

Cluster with the deployed HDFS consists of one main NameNode and DataNodes in the master-slave architecture - Fig. 3. Machine containing NameNode behaves as a master server. A single NameNode is responsible for managing the file system namespace and regulating users access to files. Also, it performs the typical file system operations on stored files (like copying, moving).

Every node in a cluster has (typically one) DataNode responsible for its data storage, where files divided into one or more blocks are kept. They perform reading and writing requests as well as the operations on blocks ordered by NameNode.

Apart from the typical distributed file system features, HDFS is designed and developed to fulfil high efficiency goals by enabling parallel read and write operations. Hardware failure is not treated as exception but rather as a norm. Blocks are by default stored in more than one node in order to be easily recovered in case of the partial breakdown. Also, possible faults are monitored and quickly detected, so that nodes can be recovered [13].

III. Implementation of New Functionalities

During studies on the Apache Hadoop framework, there appeared different approaches how to solve the problem of in-advance data shipping. The ideal way, according to the reusability principle, was to take an advantage of the existing code and reuse it by adding the new modules. It aimed to identify parts of source code where the necessary changes should be made to provide the desired solution but also to make it reusable later for the further improvements [5].

The desire plan was that the new module could be invoked by the thrid-party components. In addition, the Hadoop user should have a possibility to manually call the desired in-advance data shipping functionality.

After trying different approaches to connect this with modules moving blocks, the best one was to add a new custom functionality. Its role would be to directly access blocks of data and be able to stream data to the specific data nodes. Although the data streaming is present in many parts of the Hadoop framework, the useful classes were Processor and Dispatcher (Fig. 4) which can access the single blocks of data.

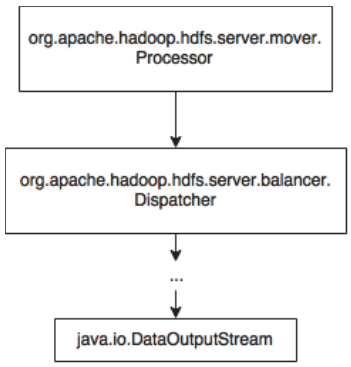

Fig. 4. Mover class diagram

In this stage, the new module was designed to be executed by a user from the command line. The architecture presented on Fig. 5 contains all classes used in the implementation. White boxes indicate which classes were already existing in Hadoop source code and altered by adding new methods while yellow boxes represent newly added classes.

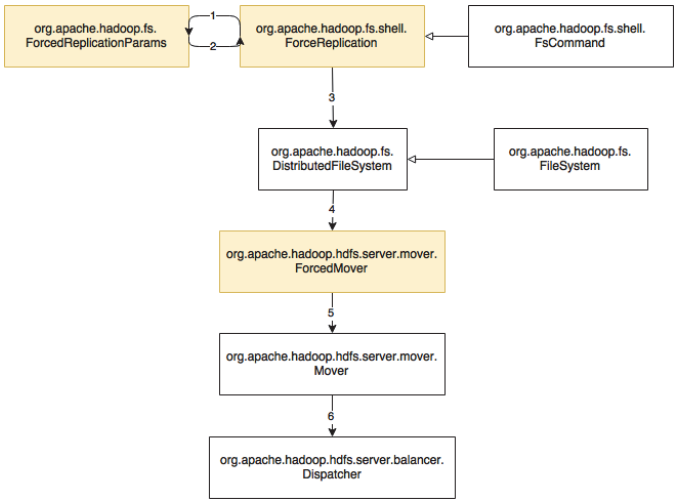

Fig. 5. ForceReplication architecture diagram

The main logic behind the new functionality of instructed data shipping is included in the class `ForcedMover`. Class `ForceReplication` is the one directly interacting with a user while changes in `Processor` (inner class of `Mover`) and `Dispatcher` make it possible to execute the actual data shipping. Name force replication is explicitly used in this study as in fact moving data can be defined as instructing (forcing) replicas to be placed in a given datanode.

The whole process starts from a user who commands to move blocks of data. Information, such as which block and which datanodes are involved, is stored as an instance of `ForcedReplicationParams`. On the diagram in Fig. 5 it is indicated as steps 1 and 2. Step 3 introduces the demand of forced replication to the `DistributedFileSystem` - implementation of `FileSytstem` which directly calls `ForcedMover` (step 4). Later the `Processor` and `Dispatcher` perform the actual data shipping.

The basic way a user can manage the cluster is through commands executed on the namenode. These shell-like commands directly interact with HDFS and allow a user to manipulate files, analogously to the UNIX commands.

Main class supporting all commands operating on files is `FSCommand`. Thus, also the class `ForceReplication` invoking the force replication extends it. `FSCommand` provides common functions for all commands like operations on extra parameters and checking whether the file exists.

In order to call the force replication, a user needs to use the following command:

```
bin/hadoop –forceReplication <blockID>
<srcDatanode> <destDatanode> <path>
```

User has to determine the specific parameters:

- `blockID` - every block is defined by its parameters, including ID
- `srcDatanode` - datanode where currently the block is located
- `dstDatanoce` - datanode where block should be moved

All this information can be obtained from HDFS, e.g. using the `fsck` command. In case a user tries to execute the command forceReplication without proper parameters he gets the appropriate warning.

`ForceReplication` uses instances of class `ForcedReplicationParams` to store information about blocks which should be transported throughout the whole implementation. It contains also the basic validation whether given path does not lead to the folder and verifying checksum with built-in `FileSystem` method.

The implementation of the abstract general `FileSystem` for DFS system is called `DistributedFileSystem`, which is the way end-user interacts with Hadoop. At this point, all information about blocks and nodes can be accessed in a straightforward way. Connected to the instance of a namenode, `DistributedFileSystem` has access to all storage reports. Therefore, this step was chosen to execute the detailed validation. Algorithm goes one by one through existing blocks and datanodes and looks for ones indicated by user. If not found, it stops the execution with the appropriate message. It also checks whether source and destination datanodes are not the same.

Class `ForcedMover` is the one where the planning of the block movement is placed. It uses specific classes `LocatedBlock` and `DBlock` to manage source data block and design the new desired one. Similarly it uses `MLocation` to point the proper datanodes instances and correct type of storage (out of ARCHIVE, SSD, DISK and RAM_DISK) for `Processor`. Finally it can call the instance of `Processor` and function `scheduleMoveForcedReplica`. In the end, a user gets information about the current state in form of logs.

The last classes directly involved are the `Processor`, subclass of the `Mover` class, and `Dispatcher`. Function `scheduleMoveForcedReplica` inside the `Processor` specifies the instance of `StorageGroup` (connection between datanodes and types of storage). Then the final move can be determined and added to the queue. Finally, function calls `Dispatcher` to execute the desired move. Inside the `Dispatcher` class, a user is also informed about the result of operation - moving a desired block between datanodes.

As indicated before, the desired result was a module which can be a solid base for further enhancements which includes various in-advance schedulers which automatically reallocate data to benefit from the data locality approach in Hadoop. It requires further studies on how to improve the efficiency by reducing the latency and to improve the performance of running applications. The main topic in this paper was to make it possible to manually shift data across the cluster.

The outcome of these studies is the functionality allowing Hadoop users to instruct movement of blocks between specified nodes in the cluster.

IV. Conclusions

The paper presents results of the Apache Hadoop framework analysis for introducing the new functionalities in the Hadoop Distributed File System. The proposed solution is elaborated in more details together with its main advantages true to the open source projects guidelines. The developed solution allows user to freely redistribute data across the cluster and make use of the data locality principles. Furthermore, the developed module can simplify the further studies on more advanced schedulers. Proposed functionality can be a good basis for the future Hadoop enhancements. It would be enough to connect it programmatically with any further modifications, working with instructed redistribution of data. Example of such improvements could be an advanced scheduler which recognizes in advance which data blocks will be needed while task is already running. In order to prevent HDFS from transferring the following data blocks to the machine where the computation occurs, scheduler would be responsible for these movements. Studies showed that it could significantly improve the built-in data locality solution and therefore, the resource utilization and the whole system performance [14].

References

[1] J. Manyika, M. Chui, B. Brown, J. Bughin, R. Dobbs, C. Roxburgh, and A. H. Byers, “Big data: The next frontier for innovation, competition, and productivity,” McKinsey Global Institute, Tech. Rep., June 2011.

[2] D. Laney, "3D data management: Controlling data volume, velocity, and variety," META Group, Tech. Rep., February 2001. [Online]. Available: http://blogs.gartner.com/doug-laney/files/2012/01/ad949-3D-Data-Management-Controlling-Data-Volume-Velocity-and-Variety.pdf
[3] K. Normandeau, "Beyond volume, variety and velocity is the issue of big data veracity," September 2013. [Online]. Available: http://insidebigdata.com/2013/09/12/beyond-volume-variety-velocity-issue-big-data-veracity/
[4] B. Marr, "Why only one of the 5 vs of big data really matters," March 2015. [Online]. Available: http://www.ibmbigdatahub.com/blog/why-only-one-5-vs-big-data-really-matters
[5] Apache Software Foundation, "Apache hadoop." [Online]. Available: http://hadoop.apache.org/docs/current/
[6] P. Jovanovic, O. Romero, T. Calders, and A. Abelló, "H-word: Supporting job scheduling in hadoop with workload-driven data redistribution," 2016.
[7] B. Hedlund, "Understanding hadoop clusters and the network," September 2011. [Online]. Available: http://bradhedlund.com/2011/09/10/understanding-hadoop-clusters-and-the-network/
[8] "Apache hadoop (mapreduce) internals - diagrams," http://ercoppa.github.io/HadoopInternals/.
[9] Hortonworks, Inc, "Hortonworks." [Online]. Available: https://hortonworks.com
[10] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," *Commun. ACM*, vol. 51, no. 1, pp. 107–113, 2008. [Online]. Available: http://doi.acm.org/10.1145/1327452.1327492
[11] H. Herodotou, "Hadoop performance models," *CoRR*, vol. abs/1106.0940, 2011. [Online]. Available: http://arxiv.org/abs/1106.0940
[12] V. K. Vavilapalli, A. C. Murthy, C. Douglas, S. Agarwal, M. Konar, R. Evans, T. Graves, J. Lowe, H. Shah, S. Seth, B. Saha, C. Curino, O. O'Malley, S. Radia, B. Reed, and E. Baldeschwieler, "Apache hadoop YARN: yet another resource negotiator," *ACM Symposium on Cloud Computing, SOCC '13, Santa Clara, CA, USA, October 1-3, 2013*, pp. 5:1–5:16, 2013. [Online]. Available: http://doi.acm.org/10.1145/2523616.2523633
[13] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The hadoop distributed file system," 2010, pp. 1–10. [Online]. Available: http://dx.doi.org/10.1109/MSST.2010.5496972
[14] J. Kałużka, "Data locality in hadoop," *M.Sc. Thesis (Lodz University of Technology, Universitat Politècnica de Catalunya)*, October 2016.

Justyna Kałużka received the BSc degree in Telecommunications and Computer Science in 2014 at International Faculty of Engineering, Lodz University of Technology, Poland and BSc degree in Informatics in 2016 (also TUL). As for the master studies, she was working on her final project "Data Locality in Hadoop" during studies exchange at Universitat Politècnica de Catalunya and was entitled with the MSc degree in Computer Science and Information Technology at TUL in 2016. Since then she has been working in industry (currently Skyscanner Ltd) on the microservices-based cloud architecture.

Małgorzata Napieralska received the M.Sc. in 1982 from Technical University of Łódź (TUL), and in 1991 Ph.D. in microelectronics from Institut Nationale des Sciences Appliquées (INSA) de Toulouse (France).
Currently she is with the Department of Microelectronics and Computer Science, TUL. She is author or co-author of over 140 scientific publications and 2 books. She participated in the preparation and realization of 15 European research and educational projects. She was a head of 2 interdisciplinary grants of the Polish Committee of Scientific Research and participant of 11 others. She is interested in VLSI design, modelling of semiconductor devices, as well as interdisciplinary applications of Integrated Design, nanotechnology and biometrics.

Oscar Romero received the BSc degree in Informatics Engineering at Barcelona School of Informatics (Universitat Politècnica de Catalunya, Spain) in 2004. In 2010 he obtained the PhD degree in Computing from the same university - his thesis was entitled "Automating the Multidimensional Design of Data Warehouses". Currently he is working at the Department of Service and Information System Engineering (UPC) while his main interest are business intelligence, Big Data and the semantic web.

Petar Jovanovic graduated in Software Engineering School of Electrical Engineering at University of Belgrade, Serbia in 2010 and obtained the MSc degree in Computing at Barcelona School of Informatics (Universitat Politècnica de Catalunya, Spain). During his PhD at UPC he was working on the problems of requirement-driven design and optimization of data-intensive flows. In 2016 he obtained the PhD title under the cotutelle between Universitat Politècnica de Catalunya and Université Libre de Bruxelles. His main research interests fall into the business intelligence field.