Experimental Analysis of WebHDFS API Throughput

Yordan Kalmukov, Milko Marinov
Department of Computer Systems and Technologies, University of Ruse, Ruse, Bulgaria

***Abstract*—Data analysis is very important for the success of any business today. It helps to optimize business processes, analyze users' behavior, demands etc. There are powerful data analytics tools, such as the ones of the Hadoop ecosystem, but they require multiple high-performance servers to run and high-qualified experts to install, configure and support them. In most cases, small companies and start-ups could not afford such expenses. However, they can use them as web services, on demand, and pay much lower fees per request. To do that, companies should somehow share their data with an existing, already deployed, Hadoop cluster. The most common way of uploading their files to the Hadoop's Distributed File System (HDFS) is through the WebHDFS API (Application Programming Interface) that allows remote access to HDFS. For that reason, the API's throughput is very important for the efficient integration of a company's data to the Hadoop cluster. This paper performs a series of experimental analyses aiming to determine the WebHDFS API's throughput, if it is a bottleneck in integration of a company's data to existing Hadoop infrastructure and to detect all possible factors that influence the speed of data transmission between the clients' software and the Hadoop' file system.**

Keywords*—*WebHDFS API; throughput analysis; data analytical tools; Hadoop Distributed File System (HDFS)

I. Introduction

Data analysis is a key point for success of any business organization. It allows companies to extract knowledge from the data they gather, to optimize their business processes and operations, to predict future failovers and to determine the right moment of maintenance. In general, users and clients are source of enormous amount of data. Companies can use these data to analyze users' behavior [1] and anticipate their demands. So, the analysis of any type of data can provide a significant competitive advantage of the company over rival businesses.

However, data analysis is a complex, time-consuming and computationally intensive task. Data analytics tools are usually either expensive or requires multiple high performance servers to run together with highly qualified IT experts to install and support them. This, of course, is not affordable for small and especially start-up companies. Fortunately, they can still use big data analysis tools, like the ones provided by the Hadoop's ecosystem, by hiring them as services "on demand". The "on demand" business model is a modern trend for hiring web-based (cloud-based) services and paying per request, rather than buying own expensive software and hardware. It provides maximum scalability and flexibility. According to it, computing resources are always available on the Internet and a company can use as many resources as it needs at the moment, while paying as much as it consumes.

Using remote data analytics services however, requires that the company share its data with the service provider [2]. In case big data analytics tools, being provided, are part of the Hadoop ecosystem, they will read the data from the "Hadoop Distributed File System" (HDFS). Although there are several ways to copy data to remote HDFS [3], the most preferable one is through the WebHDFS Application Programming Interface (API). It allows third-party applications to connect to remote HDFS file system and write/read files to/from it. As we are considering big data analysis, the amount of data being transferred is supposed to be large enough, so the WebHDFS API's throughput plays an important role in integration of a company's data to the Hadoop service provider. Other ways of data integration are reviewed in [4],[5],[6],[7],[8] and [9].

The aim of this work is to perform a series of experimental analyses to determine the WebHDFS API's throughput; if it is a bottleneck in integration of a company's data to existing Hadoop infrastructure; and to detect all possible external factors that influences the speed of data transmission between the clients' software and the WebHDFS API of existing Hadoop cluster.

The paper is structured as follows: Section II reviews some previous work done by other researchers. Section III describes the experimental system's architecture and the experimental setup in details. Section IV analyzes and discusses obtained experimental results. Finally, Section V ends the article with a conclusion, outlining and summarizing all key observations authors noticed during the experimental analysis.

II. Related Work

HDFS allows management of large volumes of data using commodity items. This reinforces the need to provide robust data protection to facilitate file sharing in Hadoop, as well as having a trusted mechanism to verify the authenticity of shared files. This is the focus of [10], where the authors' attention is directed to improving the security of HDFS using a blockchain-enabled approach (hereafter referred to as BlockHDFS). User connects to the WebHDFS REST API, through which all data retrieval and modifications are implemented. In BlockHDFS, the blockchain is responsible for storing the file metadata. The costs incurred in storing HDFS file metadata on the blockchain are twofold. First, the WebHDFS API must read a file's metadata from HDFS as a hash value. Second, additional operations are required to store the metadata in the blockchain. However, since the metadata size is typically small, such overhead will neither introduce high latency for HDFS operations nor require a large amount of disk space for blockchain storage. The paper proposes a new approach to introduce blockchain (and more specifically, Hyperledger) to improve the security of the HDFS ecosystem.

Paper [11] discusses one of the most significant challenges of next-generation big data federation platforms, namely the access control in Hadoop systems. The paper critically analyzes and explores security limitations in Hadoop systems and presents a tool called "Big Data federation access broker" to address eight major Hadoop security limitations. To validate the performance of the broker, authors have conducted a set of experimental studies on a real Hadoop cluster. They made a comparison between read and write operations, performed through WebHDFS, in two cases - without using any security measures (pure WebHDFS access) and when using the authors' proposed broker model for accessing the big data. Performance analysis of operations executed over WebHDFS with files of various sizes was done as well.

Authors of [12] discuss the design of a data transfer service, called Stargate, to address the challenges of large data transfers over a WAN. Stargate implements a content-addressable protocol and multi-layered caching to cope with these challenges. It uses a novel approach that localizes computation, cache, and transfers to achieve efficient data access in cluster computing. Stargate is evaluated experimentally by comparing its performance with two widely used Hadoop data access methods - DistCP and WebHDFS. DistCP is a built-in Hadoop data delivery tool. DistCP preorders data, while WebHDFS provides data access on-demand. The elapsed times of three Hadoop benchmarks that have different I/O workloads were compared to evaluate efficiency. Experiments show that Stargate over WAN has comparable performance to HDFS running on a LAN. It also has lower overhead than WebHDFS, which is widely used for remote access to data from Hadoop clusters.

Apache Spark uses a cluster of compute-optimized servers on which the execution modules run, and a cluster of servers, optimized for performing storage operations and hosting the HDFS data. However, the network transfer from the data warehouse to the computing cluster becomes a serious obstacle for big data processing. Near-data processing (NDP) is a concept that aims to ease the network load in such cases by offloading some of the computing tasks to the storage cluster. Rachuri et al. present an architecture and basic principles of implementation of an NDP system for Spark [13]. HDFS can be configured to add redundancy by copying the same blocks of files across multiple data nodes to improve fault tolerance. It also provides an API - WebHDFS. In the proposed implementation, the authors take advantage of the replication factor to increase the number of data nodes that can perform operations related to offloading the computational tasks and intercept the WebHDFS communication between the client and the data node to perform NDP operations. Simulation results and experiments conducted on the developed prototype show that SparkNDP can help reduce the execution time of Spark queries compared to both - the default approach of not directing any tasks to the repository, and the direct NDP approach to offloading all tasks to the repository.

High Performance Computing (HPC) and Big Data are two trends that are starting to converge. In this process, aspects of hardware architectures, system support, and programming paradigms are revisited from both perspectives. The authors of [14] present their experience on this path of convergence. They propose a framework through which some of the programming problems, arising from such integration, are solved. An integrated environment has been developed that integrates: (1) COMPS, a programming environment for developing and running parallel applications for distributed infrastructures; (2) Lemonade, a data mining and analysis tool; and (3) HDFS, the most widely used distributed file system for big data. In order to implement the integration between COMPS and HDFS, aspects of the available techniques for communication between external applications, in particular those written in Java and Python, and HDFS are considered. HDFS provides interfaces through a direct Java API, a command-line interface (CLI), a REST API (WebHDFS), and a C API (libhdfs). The proposed solution provides processing of large data transfers, with access to low-level functions.

WebHDFS allows users to connect to HDFS from outside the Hadoop cluster, which is especially useful when an external application needs to load data into or out of HDFS or work with the data stored in HDFS. WebHDFS also supports (for all HDFS users) operations such as reading files, writing to files, creating directories, changing access permissions, renaming, etc. The WebHDFS API is used for two functions in [15]: 1) after server-side processing is complete, this data is stored in HDFS via the WebHDFS API; and 2) when the created final data for visualization in raw text format is requested by clients, the data is passed to them via the WebHDFS API.

A system architecture combining the "IP multimedia subsystem (IMS)" platform and the Hadoop system used in the distributed storage of the IMS service resources is proposed in [16]. The result is a manageable Hadoop-based data center for telecommunication service providers. Interoperability between different systems is achieved through RESTful web services. The WebHDFS API is used to allow services to interact with HDFS, while the Oozie Web Services API is used for the compute service. The conducted tests prove the availability, scalability, and reliability of the proposed system. Experimental results show that system performance is improved, especially in terms of disk space utilization and system throughput.

Although HDFS works well with medium-sized and large files, its performance seriously degrades in case of multiple very small files. To overcome this shortcoming, the authors of [17] propose a system to improve the performance of HDFS using a distributed full-text search system. By indexing each file's metadata, such as name, size, date, and description, files can be quickly accessed through efficient metadata searches. Additionally, by consolidating many small files into one large file to be stored with better space and I/O efficiency, the negative performance impacts caused by directly storing each small file separately are avoided.

HDFS is a widely used open-source scalable and reliable file management system designed as a general-purpose distributed file storage solution. WebHDFS is a service for accessing data stored and maintained in HDFS. It runs on all nodes in the Hadoop cluster and provides a REST interface for data access. Unlike other file systems or data transfer tools, WebHDFS detects the layout of data blocks stored in HDFS. Using this block information, clients can directly access the

HDFS node (data node) on which the data is stored. This not only reduces data access latency, but also provides load balancing of data access requests. This motivated the authors of [18],[19],[20] to investigate the performance of HDFS in remote data access.

III. Experimental Setup

To study the throughput of the WebHDFS API, a testing client must be developed to access the interface in both read and write modes. The client, for the current experiment, has been implemented in the php programming language. It uses the open source library PHP-Hadoop-HDFS [21], implemented and maintained by Aleksandr Kuzmenko. It is a wrapping library that does not do any specific data processing, but just composes the necessary HTTP requests to access the WebHDFS API. The access itself is done through the cURL (client URL library) library [22], distributed together with the php interpreter. The WebHDFS API could be accessed without PHP-Hadoop-HDFS library, but it facilitates the access, since the library frees the programmer from having to know the WebHDFS API itself. Instead, the programmer should only know the methods that the library implements and their input arguments. The architecture of the experimental system is presented on Fig. 1.

When performing the experiments, several key parameters should be monitored: total time; upload speed; download speed; bytes uploaded; bytes downloaded; and response HTTP code. Fortunately, all of these, together with many more parameters, are measured by the cURL library itself.

The throughput of the WebHDFS API is not the only limitation factor. The network speed is important as well, even more important. Even if the API itself allows the transfer of hundreds of megabytes per second, if the user's Internet speed is slow, then the API’s throughput does not matter at all. Therefore, experiments should be performed from different type of computer networks:

1) Internet- This is the most important experiment, as this is the most realistic scenario for accessing the Hadoop cluster. Most likely, the greatest limitation factor will be the Internet connection speed.

2) Corporate LAN of the service provider: This experiment is important in case of a large company, having multiple offices, maintaining its own Hadoop infrastructure and MAN network between the different locations. Experiments could be done at multiple network speeds - 1 gbit/s and 100 mbit/s seem to be the most realistic speeds for a company’s MAN.

Although writing to HDFS is more important than reading, both operations will be tested. Writing is more important since company’s data should be saved to HDFS, before being analyzed by the Hadoop’s data analytical tools. So, the data flow direction in general will be from the company to the Hadoop cluster. However, reading is also useful.

Experiments have been conducted with small, medium-sized and large files which are generated with the Windows’ fsutil application. They contain only zeros (i.e. no meaningful information). Since we are making performance experiments, the content of the files does not matter at all, but their exact size does. It is important that their size can be precisely controlled.

The Hadoop cluster consists of 1 name node (2 x Intel Xeon Silver 4110, 32 total threads, 64 GB RAM) and 9 data nodes (Intel Xeon E-2124, 16 GB RAM). Servers are connected through 24 Port Gigabit switch HPE OfficeConnect 1820.

The experimental application, developed in php, runs on a laptop computer (Intel i7-7500U, 12 GB RAM) for all experiments in all types of networks. Using the same laptop for all experiments is intentionally done in order to ignore the influence of the client’s hardware.

The access from the client to the WebHDFS API is done through the:

1) Internet – the access is done from a laptop computer, connected to a home router. According to the subscription, the guaranteed Internet speed is 80 mbit/s.

2) University of Ruse’s campus network – the access is done from the same laptop, connected to any point of the university’s campus-wide 100 mbit/s network.

3) 1 gbit/s cluster’s switch – the laptop is connected directly to the 1-gigabit switch of the Hadoop cluster.

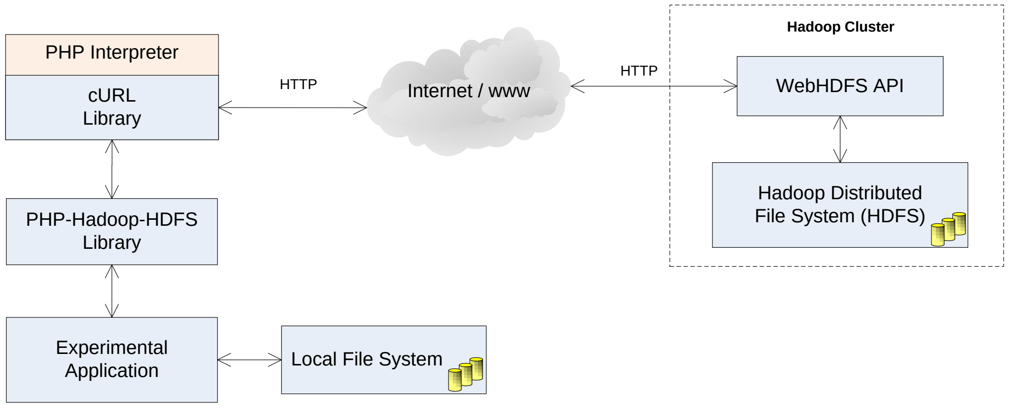


Fig. 1. Architecture of the system for experimental study of the WebHDFS API’s throughput.

IV. Experimental Results

A. *Writing Data to HDFS*

As previously mentioned, when integrating a company's data to an existing Hadoop cluster, writing/saving files to WebHDFS is the most important operation. So, it is tested with priority.

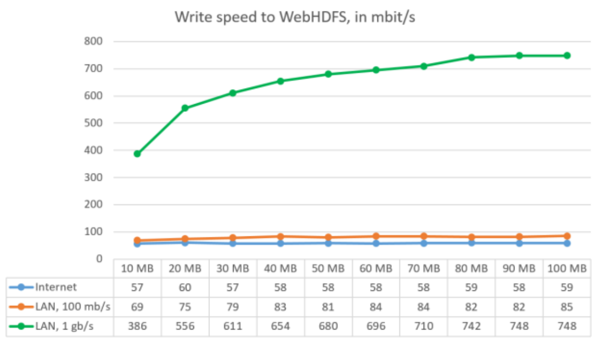
Experiments started with small to medium-sized files from 10 to 100 MB, with a step of 10MB. Results are presented on Fig. 2. They show almost constant write speed when WebHDFS is accessed from the Internet and the university's campus-wide 100 mb/s network. The transfer speed almost reaches the network's capacity – with the 80 mbit/s Internet connection, the achieved speed is around 60 mbit/s, while within the 100 mbit/s campus-wide LAN, we achieve sustainable transfer of 85 mbit/s.

When the laptop is connected directly to the cluster's gigabit switch, the writing speed is times higher and is increasing with the increase of the file size.

Since the API supports high-speed data transfer, we decided to go further and experiment with medium-sized files, from 100 to 300 MB with step of 50 MB, and large files from 500 to 1500 MB with step of 500 MB. Results are shown on Fig. 3 and 4 respectively. For larger files, upload speed becomes constant (about 800 mb/s) for the gigabit network as well. That proves the WebHDFS API supports very high writing speeds and could not be considered as bottleneck in the integration architecture. Most probably, the API just saves the incoming data to the HDFS file system without applying any complex processing on them.

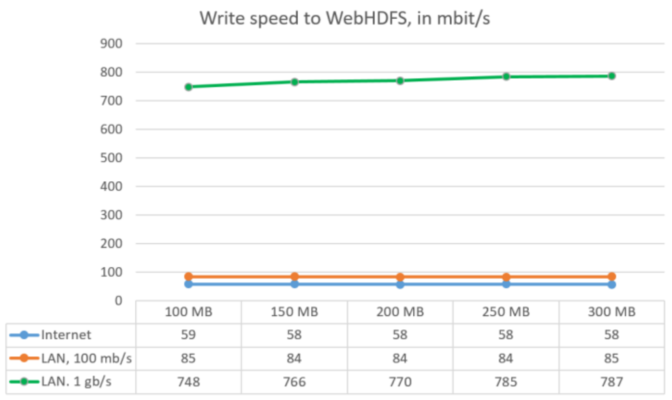
As known from everyday usage of computer network and different types of file transfer, copying single large files is much more efficient than copying multiple smaller files. There are objective reasons for that, including metadata overheads. So, it is worth testing how much slower uploading multiple smaller files will be in respect to a single large file, having the same total size.

Three experiments have been performed with single large files of 100 MB, 200 MB and 300 MB, and 10 x 10 MB, 20 x 10 MB and 30 x 10 MB. The size of the single large file exactly matches the total sum of bytes of the respective many 10 MB files. Results are presented on Fig. 5. Expectedly, uploading a single large file is faster than uploading many smaller files, having the same total size as the large one.



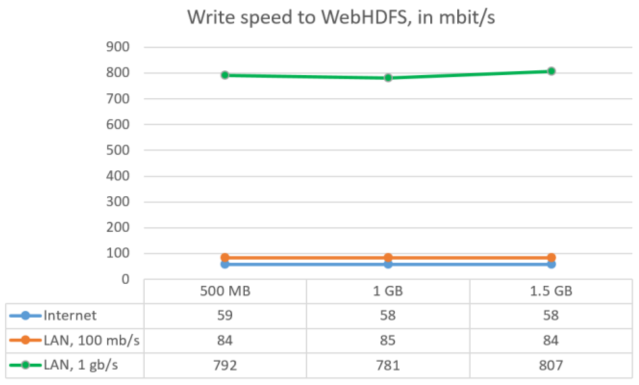
	10 MB	20 MB	30 MB	40 MB	50 MB	60 MB	70 MB	80 MB	90 MB	100 MB
Internet	57	60	57	58	58	58	58	59	58	59
LAN, 100 mb/s	69	75	79	83	81	84	84	82	82	85
LAN, 1 gb/s	386	556	611	654	680	696	710	742	748	748

Fig. 2. Write speed to HDFS via the WebHDFS API with relatively small file sizes - from 10 MB to 100 MB.



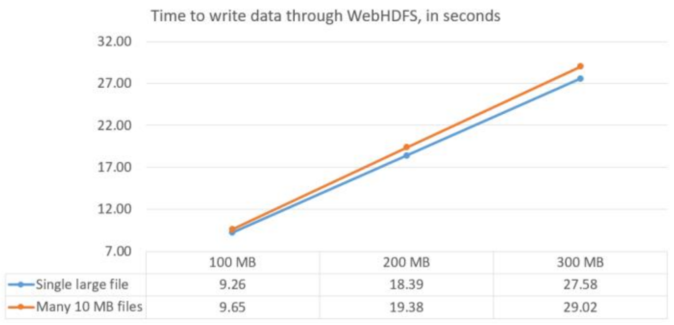
	100 MB	150 MB	200 MB	250 MB	300 MB
Internet	59	58	58	58	58
LAN, 100 mb/s	85	84	84	84	85
LAN. 1 gb/s	748	766	770	785	787

Fig. 3. Write speed to HDFS via the WebHDFS API. File sizes from 100 MB to 300 MB. Write speed is constant, independent on the file size.



	500 MB	1 GB	1.5 GB
Internet	59	58	58
LAN, 100 mb/s	84	85	84
LAN, 1 gb/s	792	781	807

Fig. 4. Write speed to HDFS via the WebHDFS API. File sizes from 500 MB to 1500 MB. Write speed is still constant, although files have got very large.

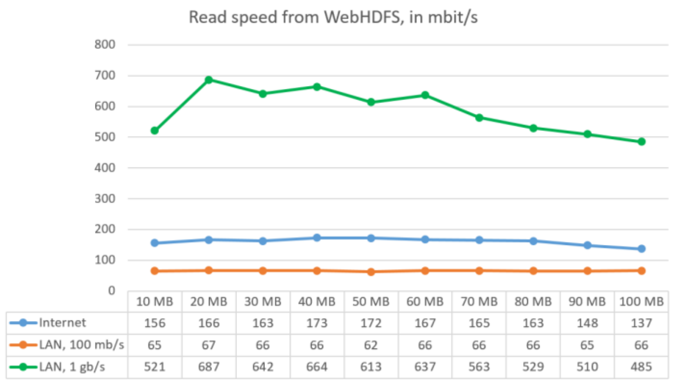


	100 MB	200 MB	300 MB
Single large file	9.26	18.39	27.58
Many 10 MB files	9.65	19.38	29.02

Fig. 5. Time to write data (single large file or multiple small files) to HDFS via the WebHDFS API.

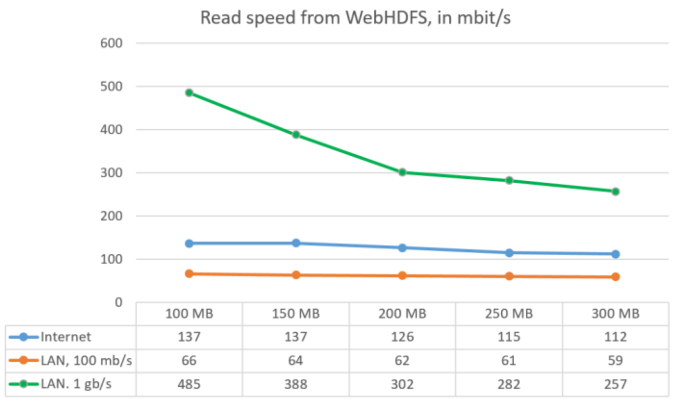
B. *Reading Data from HDFS*

Although reading data from HDFS is less important operation when integrating a company's data to existing Hadoop cluster, a series of experiments will be done by using the files, already uploaded to the HDFS through the WebHDFS API. The same client is used, as in the previous experiments for writing data, and runs on the same laptop computer as well. Results are shown on Fig. 6 (for files 10 to 100 MB), Fig. 7 (for files 100 to 300 MB) and Fig. 8 (for files 500 to 1500 MB).



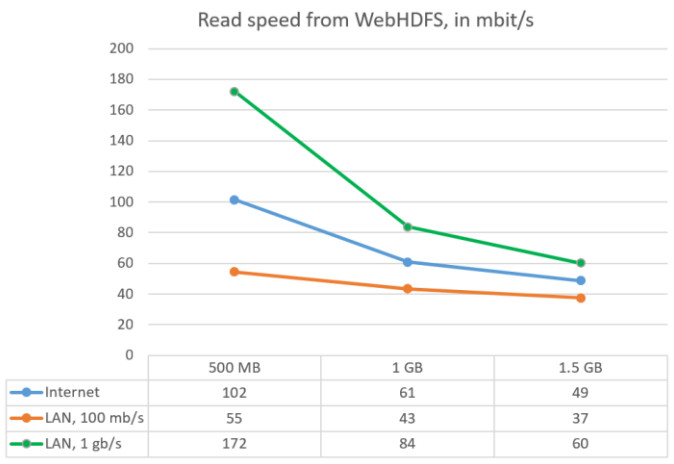
	10 MB	20 MB	30 MB	40 MB	50 MB	60 MB	70 MB	80 MB	90 MB	100 MB
Internet	156	166	163	173	172	167	165	163	148	137
LAN, 100 mb/s	65	67	66	66	62	66	66	66	65	66
LAN, 1 gb/s	521	687	642	664	613	637	563	529	510	485

Fig. 6. Read speed from HDFS via the WebHDFS API with relatively small file sizes - from 10 MB to 100 MB.



	100 MB	150 MB	200 MB	250 MB	300 MB
Internet	137	137	126	115	112
LAN, 100 mb/s	66	64	62	61	59
LAN. 1 gb/s	485	388	302	282	257

Fig. 7. Read speed from HDFS via the WebHDFS API. File sizes from 100 MB to 300 MB. On high-speed networks, read/download speed is decreasing with increasing the file size.



	500 MB	1 GB	1.5 GB
Internet	102	61	49
LAN, 100 mb/s	55	43	37
LAN, 1 gb/s	172	84	60

Fig. 8. Read speed from HDFS via the WebHDFS API. File sizes from 500 MB to 1500 MB. Read/download speed is decreasing with increasing the file size on all networks.

In contrast to the constant writing speed however, the reading (download) speed rapidly falls with increasing the file size – significantly noticeable for the high-speed 1 gbit/s connection. The initial suspicion/assumption was this decrease of the reading speed is related to the higher number of packets that the large files consist of. There is a sense in that – since larger files contains many more packets, more time may be required to reconstruct the file from the higher number of packets. To test if the assumption is correct, the experiments have been repeated with measuring not just the total time, but transferring time and file saving time separately.

Results show that the transferring time, without the time needed to save the file within the local file system is commensurate with the total time. And time needed to reconstruct and save the file is actually very small and does not depend on the file size. So, the assumption is wrong. Since the communication between the client and the WebHDFS API is handled by the cURL library, distributed with the PHP interpreter, then the cause of the reading speed decrease could be either the cURL library itself or the API. To determine where the problem is, the WebHDFS API should be accessed in another way. It could be accessed directly through a browser, but it is not very convenient. Other tools like Postman or Rester are not very suitable as well, since they do not measure times. The cURL library however is not developed specially for PHP, but it is an open source project ported to almost any programming language. It is provided as a built-in application in Unix/Linux/MacOS and could be downloaded as external stand-alone application for Windows.

We take the HTTP queries, generated by the PHP-Hadoop-HDFS library and run them from the cURL applications for MacOS and Windows. Results show that files are downloading (read) from HDFS with very high speed almost reaching the maximum throughput capacity of the relevant type of network, regardless of the file size. So, the causer of the read speed decrease in our experiments is determined to be the cURL library, distributed with the PHP interpreter. Further experiments will be done with different versions of PHP and its accompanying cURL library.

Another interesting and unexpected result occurred when reading files from WebHDFS over the Internet. The utilized home Internet subscription plan is guaranteeing speed of 80 mbit/s. However, reading was done at speeds up to 175 mbit/s. The connection between the home computer and the Internet Service Provider (ISP) is actually higher than the guaranteed 80mbit/s. Apparently, the ISP also has a high-speed connection to the university’s network, where the Hadoop cluster is located. So, the access to the servers from the home computer is done in a kind of MAN network with speeds significantly higher than guaranteed Internet connection. Interestingly, this is not the case when writing data to WebHDFS. When writing (uploading) files, they are transferred at a speed no higher than the guaranteed Internet connection. This, however, is a specific case study related to the specific ISP and should not be considered as an essential part of the results of the experiments.

V. Conclusions

Since WebHDFS is the most preferable way for remote access of the distributed file system HDFS, it is important to know its capabilities, performance and throughput.

After performing dozens of experiments and additional analyses, the results could be summarized in the following conclusions:

1) WebHDFS API allows data exchange with the Hadoop Distributed File System (HDFS) at very high speeds, and in general it is not the limitation factor, but the speed of the network itself. In a 1 gbit/s network, we achieve data transfer rates of around 800 mbit/s. On slower networks, the network capacity is almost entirely compressed.

2) The speed of writing (uploading) files through the WebHDFS API does not depend on the size of the files, but remains constant and it is only limited by the network capacity.

3) In contrast to the constant write speed, the reading (download) speed through the WebHDFS API decreases rapidly as the file size increases.

4) However, the reason for the decreasing read speed is not in the WebHDFS API itself, but in the implementation of the cURL library, distributed together with the PHP interpreter. When accessing the API in alternative ways, the read speed remains constant and comparable to the write speed.

5) When reading files from the WebHDFS API by using PHP and cURL, it is mandatory that the PHP interpreter is configured to use a larger amount of RAM memory than the size of the files being read. This is expected since the data transfer happens in multiple small network packets, but in order to reconstruct the file from them, they must be stored and arranged in a common buffer (located within the RAM memory).

6) When writing files through the WebHDFS API, the amount of RAM memory that the PHP interpreter can work with is not of such importance because the data is read from the local file system in chunks, which are typically much smaller than the default memory limit of 128 MB. In this case, the buffer size is important on the receiving side – i.e. on the server where the WebHDFS API is running. This size is managed by Cloudera Manager and is large enough.

7) In relation to the above-mentioned, it has been observed that files larger than 2GB cannot be reliably written to HDFS via the WebHDFS API. For all of our attempts to upload a file larger than 2 GB, the server did not return any HTTP response, although in some cases the files were actually stored in HDFS. The fact that files of 1.99 GB were always reliably saved, but 2.0 GB were not, suggests that the reason might be a WebHDFS setting, maybe the server-side buffer in question or something else. Further analysis could help to determine the exact cause.

8) A large number of small files are transferred more slowly than a single large file of the same total size. This observation is absolutely expected given the fact that with many small files, many separate HTTP requests are made, each of which has time to resolve the domain to an IP address, time to connect, time to make and receive the request, etc.

The WebHDFS API allows data exchange with HDFS at very high speeds, so it could not be considered as a bottleneck in the integration of a company's data to an existing Hadoop cluster. However, by default, it does not perform any user access control by itself, so additional means should be designed and implemented, or integrated, to control user access and guarantee data isolation (no third-party company should be able to access data of another company).

ACKNOWLEDGMENTS

This work is supported by the Science Fund of the "Angel Kanchev" University of Ruse and the European Regional Development Fund under grant BG05M2OP001-1.002-0002-C02 "Digitalization of Economy in Big Data Environment".

REFERENCES

[1] Ashayer, S. Yasrobi, S. Thomas and N. Tabrizi, "Performance Analysis of Hadoop Cluster for User Behavior Analysis", 2018 IEEE 20th International Conference on High Performance Computing and Communications; IEEE 16th International Conference on Smart City; IEEE 4th International Conference on Data Science and Systems (HPCC/SmartCity/DSS), Exeter, UK, 2018, pp. 805-809, doi 10.1109/HPCC/SmartCity/DSS.2018.00135.

[2] Y. Kalmukov, M. Marinov. "Hadoop as a Service: Integration of a Company's Heterogeneous Data to a Remote Hadoop Infrastructure". International Journal of Advanced Computer Science and Applications, 13(4), pp. 49-55, 2022, DOI:10.14569/IJACSA.2022.0130406.

[3] Y. Kalmukov, M. Marinov, T. Mladenova, I. Valova. "Analysis and Experimental Study of HDFS Performance". TEM Journal, 10(2), pp. 806-814, ISSN 2217-8309, DOI: 10.18421/TEM102-38, May 2021.

[4] M. Sarnovsky, P. Bednar and M. Smatana, "Data integration in scalable data analytics platform for process industries", 2017 IEEE 21st International Conference on Intelligent Engineering Systems (INES), pp. 187-192, 2017.

[5] J. Pokorný, "Integration of Relational and NoSQL Databases", Vietnam Journal of Computer Science, vol. 6, no. 4, pp. 389-405, 2019.

[6] S, Ramzan, I.S. Bajwa, B. Ramzan, and W. Anwar, "Intelligent Data Engineering for Migration to NoSQL Based Secure Environments", IEEE Access, vol. 7, pp. 69042-69057, 2019.

[7] Cholissodin, D. Seruni, J. Zulqornain, A. Hanafi, A. Ghofur, M. Alexander and M. Hasan, "Development of Big Data App for Classification based on Map Reduce of Naive Bayes with or without Web and Mobile Interface by RESTful API Using Hadoop and Spark", Journal of Information Technology and Computer Science, vol. 5(3), pp. 302–312, 2020.

[8] Anilkumar and B. Shireesha, "A Study on Optimized Big Data Performance and its Industrial Development", Journal of Advanced Research in Technology and Management Sciences, vol. 1(1), pp. 1-11, 2019.

[9] Erraissi, A. Belangour, A.Tragha, "A Comparative Study of Hadoop-based Big Data Architectures", International Journal of Web Applications, vol. 9(4), pp. 129-137, 2017.

[10] Mothukuri, S. Cheerla, R. Parizi, Q. Zhang and K. Choo, "BlockHDFS: Blockchain-integrated Hadoop distributed file system for secure provenance traceability", Blockchain: Research and Applications, vol. 2(4), pp. 1-7, 2021.

[11] F. Awaysheh, M. Alazab, M. Gupta, T. Pena and J. Cabaleiro, "Next-generation big data federation access control: A reference model", Future Generation Computer Systems, vol. 108, pp. 726-741, 2020.

[12] Choi and J. Hartman, "Stargate: remote data access between Hadoop clusters", in Proc. of the 36th Annual ACM Symposium on Applied Computing (SAC '21), ACM, NY, USA, pp. 32–39, 2021.

[13] S. Rachuri, A. Gantasala, P. Emanuel, A. Gandhi, R. Foley, P. Puhov, T. Gkountouva and H. Lei, "Optimizing Near-Data Processing for Spark", 2022 IEEE 42nd International Conference on Distributed Computing Systems (ICDCS), pp. 636-646, 2022.

[14] L. Ponce, W. Santos, W. Meira, D. Guedes, D. Lezzi and R. Badia, "Upgrading a high performance computing environment for massive data processing", Journal of Internet Services and Applications, Vol. 10:19, 2019.

[15] Khan, Y. Li, A. Anwar, Y. Cheng, T. Hoang, N. Baracaldo and A. Butt, "A Distributed and Elastic Aggregation Service for Scalable Federated

Learning Systems", ArXiv, abs/2204.07767, 2022. https://arxiv.org/abs/2204.07767.

[16] Y. Seraoui, M. Bellafkih and B. Raouyane, "A high-performance and scalable distributed storage and computing system for IMS services", 2016 2nd International Conference on Cloud Computing Technologies and Applications (CloudTech), Marrakech, Morocco, pp. 335-342, 2016.

[17] W. Xu, X. Zhao, B. Lao, G. Nong, "Enhancing HDFS with a full-text search system for massive small files", The Journal of Supercomputing, vol. 77, pp. 7149-7170, 2021.

[18] U. Özdil, and S. Ayvaz, "An experimental and comparative benchmark study examining resource utilization in managed Hadoop context", Cluster Computing, 2022. https://doi.org/10.1007/s10586-022-03728-7.

[19] Raj, R. D'Souza, "A Review on Hadoop Eco System for Big Data", International Journal of Scientific Research in Computer Science, Engineering and Information Technology, vol, 5(1), pp. 343-348, 2019.

[20] T. Ma, F. Tian and B. Dong, "Ordinal Optimization-Based Performance Model Estimation Method for HDFS", in IEEE Access, vol. 8, pp. 889-899, 2020.

[21] PHP-Hadoop-HDFS Library, Pure PHP unified wrapper for WebHDFS and CLI, https://github.com/adprofy/Php-Hadoop-Hdfs (Accessed March 2023).

[22] The cURL Project, https://curl.se/ (Accessed March 2023).