# Data Exchange Acceleration Methods in a Decentralized File System

Mikhail. M. Rovnyagin
*National Research Nuclear University MEPhI (Moscow Engineering Physics Institute)*
Moscow, Russia
m.rovnyagin.2015@ieee.org

Dmitry M. Sinelnikov
*National Research Nuclear University MEPhI (Moscow Engineering Physics Institute)*
Moscow, Russia
dsinelnikov96@gmail.com

Sergey S. Varykhanov
*National Research Nuclear University MEPhI (Moscow Engineering Physics Institute)*
Moscow, Russia
masmx64@gmail.com

Anastasia M. Khudoyarova
*National Research Nuclear University MEPhI (Moscow Engineering Physics Institute)*
Moscow, Russia
nastya.chut@gmail.com

Ivan A. Yakovenko
*National Research Nuclear University MEPhI (Moscow Engineering Physics Institute)*
Moscow, Russia
ivan.yakovenko@icloud.com

Tatyana A. Shirokikh
*National Research Nuclear University MEPhI (Moscow Engineering Physics Institute)*
Moscow, Russia
tanya.plys@bk.ru

*Abstract* — **The efficiency of storing and transmitting information in decentralized systems depends on many factors. This article presents methods for accelerating data exchange in a decentralized file system by choosing the optimal file compression algorithm and choosing the optimal server to initialize the connection of a new client. Lossless compression is one of the ways to reduce the load on the channel. In this paper, adaptive compression is considered. This is a special algorithm that, depending on the input data, will adjust the algorithm used to achieve optimal performance in terms of time and compression ratio. To maintain a certain level of quality of service and optimize the use of server resources, it is necessary to adequately predict the workload, assess the current state of the servers, and choose the right balancing strategy. In this article, the ELECTRE IS algorithm is applied to select the optimal server when initializing the connection.**

***Keywords — compression algorithm, data optimization, load balancing strategies, ELECTRE, MCDM***

## I. INTRODUCTION

The efficiency of storing and transmitting data over the network directly depends on the amount of data transmitted. Nowadays, it is still hard to say that all available systems have sufficient channel capacity to transmit data of any volume. There are many different ways to speed up the connection between devices. This article examines data compression and load balancing as methods of network optimization.

The growing popularity of distributed information services has led to a number of problems related to scaling. Three factors obstructing the scaling of such distributed systems are excessive load on servers, network bandwidth loss due to excessive data transmission and excessive latency in data delivery to the client due to transmission over slow connections.

In order to maintain a certain level of service quality for various types of incoming requests and optimize the server resources usage, it is necessary to correctly predict the workload, evaluate the current state of the servers and choose the appropriate balancing strategy. At the moment, there are number of methods and approaches for evaluating the health and performance of server groups, which we will briefly highlight further.

## II. RELATED WORKS

The choice of compression algorithm should be based on the balance between compression time and compression ratio, since different algorithms can compress different types of data in different ways. There are many performance benchmarks available on the network for the size of compressed data and common types, such as "10 gb compression benchmark" [1], however, a synthetic algorithm's benchmark is not enough to implement a dynamic compression algorithm selection.

A similar problem was solved in the article ACE: A Resource-Aware Adaptive Compression Environment [2]. It describes an approach to adaptive compression in an environment with limited resources. It was noticed that there are conditions when compression only slows down the system.

If we consider an acceleration of network connection from the load balancing perspective, one of these approaches is proposed in the article [3], where authors proposed the static approach, which is based on identifying replicas of the servers closest to the client with the necessary requested data, which in turn is placed near the sources of a large number of requests for this data. The disadvantage of this approach is the requirement to store a large amount of geographical information.

Another method of monitoring the state of servers and predicting the load is the multidimensional regression [4] usage, fast Fourier transform. Despite the fairly fast output of results, it is necessary to spend quite a lot of resources to build an accurate model. In some cases, server health assessments are not reliable, because in the case of minor changes in server parameters, the algorithm does not accurately generate the final result.

Another interesting idea to estimate the relationship between load evaluation and resource consumption parameters through a regression decision tree and an average error [4] allows to estimate the correlation between the parameters and more accurately predict server performance.

When evaluating server performance, balancing methods can also be divided into such components as forecasting and analyzing long-term and short-term load [5] and server utilization. The long-term load is modeled as dynamic harmonic regression, in which coefficients are adjusted using sequential Monte Carlo algorithms (SMC), and the short-term load is modeled using autoregression, where the parameters are also evaluated internally by SMC.

A more complex approach to choosing balancing strategies is the usage of genetic load balancing algorithms, which allows to adjust the set of tasks that a particular server can handle [6, 7].

## III. Compression algorithms

In this article, lossless compression algorithms have been considered. These algorithms are widespread: used for everyday file storage, in the medical domain, for file distribution, and in many other places for optimal data storage with the possibility of identical recovery [8].

Existing compression algorithms can be divided into three groups of algorithms by compression ratio:

- slow (0 - 10 MB/s) are mostly descendants of LZMA (LZMA, LZMA2, XZ, 7-zip), bzip2 and brotli by Google [9].
- average (10 - 500 MB/s) are deflate (gzip) and zstd by Facebook.
- fast (1 GB/s) are lzo, lz4 (Facebook) and snappy (Google).

The strongest algorithms, they are also slow ones, are mainly used for long-term data storage.

Average algorithms provide the optimal data transfer time.

And finally, fast algorithms help to optimize application performance, for example, lz4 in Elasticsearch.

## IV. Data definition

In usual work, users use and exchange various types of data. Among them, we can preliminarily identify the following types of data:

- Executable (many zeros, binary form).
- Photo/video (binary view, complex structure, often compression is already applied – JPG).
- Text (suitable for encoding applications).
- Mixed.

File format "tar" will be used to generate data sets. There is no compression when forming a tar file. The tar format was originally designed for recording on magnetic tape – its primary task is to organize files and their parameters for more compact recording on a tape.

After receiving the tar, we will apply various compression algorithms to it and measure their characteristics.

In this paper, we considered the algorithms brotli, bzip2, gzip, lzma, std and snappy. All tests were performed on Apple M1 with 8 GB of RAM with Python 3.10.

The graph in Figure 1 shows the compression ratio of a various files for different algorithms. As can be seen from the graph in Figure 1, files containing already compressed formats are poorly amenable to further compression. Columns containing pdf, mp4 and img files retain the same size. We got the best indicator for JSON containing text files.

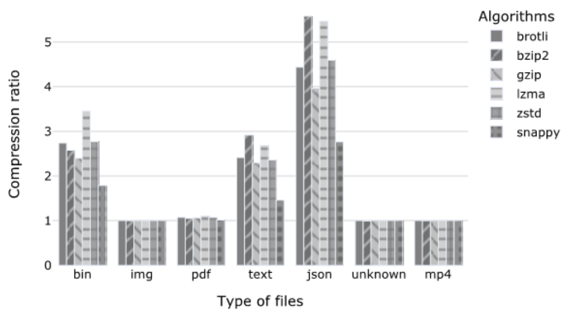

Fig. 1. The dependence of compression ratio on the file type for different compression algorithms

The results correspond to the compression theory. All compression algorithms use so-called character encoding. Their predictable sequence in text files allows you to encode characters more strongly and thereby compress the file even more.

It can also be noticed that different file types have different results depending on the compression algorithm used. For example: json, text files, binary files. Also, in Figure 2 the dependence of compression time on file type for each algorithm is shown. There is a significant difference between text and json files. And for the video, the LZMA algorithm show the best result.

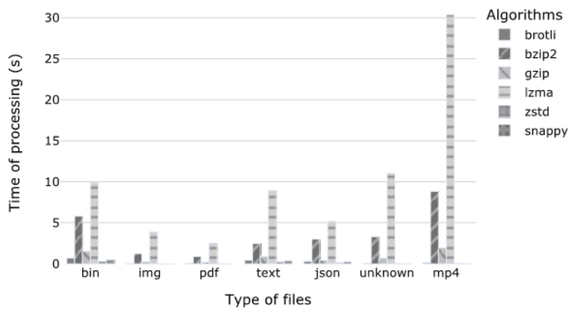

Fig. 2. The dependence of compression time on file type for different compression algorithms

In this case, we have used the same compression parameter for each algorithm. However, each of these algorithms can be configured to use its own compression level – a special setting that determines the size of the compressed block.

We considered compression with different settings for algorithms and determine the optimal compression level for the tar file containing 53 MB of json files, as an example.

We will also highlight the following data categories:

- BIN – binary.
- TXT – text.
- JSN – json files (mime-type application/json).
- UKN – unknown.

In the last category, we included all the formats that were poorly compressed or not compressed at all. There is no point to run compression for such files.

We have calculated the possible values of the operating time / compression ratio depending on the specified parameter for each algorithm.

So, for example, for brotli we've got the results shown in Figure 3.

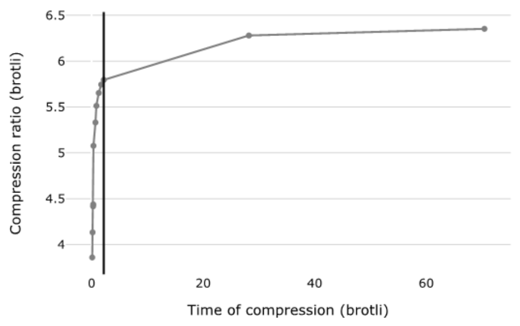

Fig. 3. The dependence of compression ratio on compression time for Brotli

The compression ratio of the file is on the ordinate axis, the time spent in seconds is on the abscissa axis.

The inflection is clearly visible, it is this point that will have the properties that we want – the greatest compression in the shortest time.

The elbow method was used to find the inflection point [10]. The point of this method is to plot perpendiculars to the line connecting the end points and measure the perpendiculars to the X-axis. The maximum value will indicate an inflection of the curve, which is called the elbow. Repeating this technique for the rest of the algorithms, we got the optimal parameter value for each algorithm. An example for zstd is shown in Figure 4.

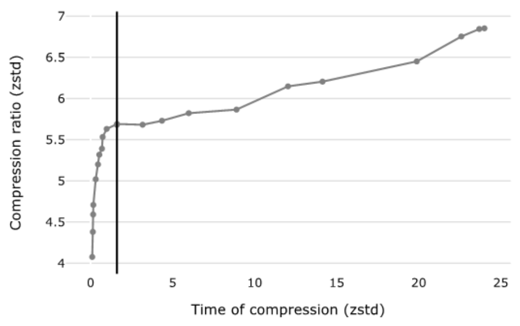

Fig. 4. The dependence of compression ratio on compression time for Zstd

## A. Data classification

In order to determine what kind of data we received at the entrance, it is necessary to define a certain metric, meta-information. For regular files, we can use mime-type:

- application/octet-stream
- image/png
- application/gzip
- application/pdf
- text/x-diff

It is also suggested to use file extensions from the "tar -tvf" program output for classification.

The study shows an interesting dependence. The data type strongly affects the compression ratio and the running time of the algorithm. So, for example, the Linux kernel in its original form has size about 900MB. Using zstd algorithm the size can be reduced by 10 times. However, this result is not free – the execution time of the algorithm reaches 363 seconds. Such compression time can be critical for data storage system. Another competitor of this algorithm is bzip2. The compression ratio is slightly lower – about 8.1, however, the time is much less – only 64 seconds. At the same time, there are some types of data (for example, images or encrypted content) that cannot be compressed. In this case, it is necessary to pay attention to the algorithm's compression time - none of the algorithms used could compress the video, but gzip spent 19 seconds, and lzma as much as 166 seconds.

Thus, the problem of adaptive compression can be solved by applying the classification of compressed data and selecting the optimal algorithm depending on the situation. From this we can conclude that some types of data do not need to be compressed at all.

## B. Choosing the suitable algorithm

After completing the previous steps, we received two matrices with the information about compression ratio of the particular algorithm for specific data types, and compression time.

To select the optimal algorithm Formula (1) was proposed.

$$F(t,r,w) = log\frac{1}{t} + \frac{r*w}{5} \qquad (1)$$

Where,

F – algorithm evaluation function,

t – time spent on compression,

r – the compression ratio of the file (the ratio of its original size to the compressed one),

w – the weighting factor used to adjust the priority.

For example, the following tables were obtained. Table 1 contains the ratio of an uncompressed file to a compressed one using a specific algorithm. Table 2 contains execution times of algorithms, complementary the first one. As can be seen, for text data, the bzip2 and brotli algorithms give a very similar result, but the execution time of the algorithms differs almost twice. The weakest and fastest algorithm, as expected, turned out to be snappy.

TABLE I. COMPRESSION RATIO COMPARATION

| | brotli | bzip2 | gzip | lzma | zstd | snappy |
|---|---|---|---|---|---|---|
| BIN | 3.25 | 2.58 | 2.37 | 3.42 | 3.04 | 1.79 |
| TXT | 2.72 | 2.92 | 2.25 | 2.61 | 2.75 | 1.46 |
| JSN | 5.80 | 5.58 | 3.86 | 5.24 | 5.69 | 2.77 |

TABLE II. COMPRESSION TIME COMPARATION

| | brotli | bzip2 | gzip | lzma | zstd | snappy |
|---|---|---|---|---|---|---|
| BIN | 10.36 | 5.82 | 1.27 | 6.59 | 2.57 | 0.51 |
| TXT | 5.07 | 2.49 | 0.41 | 4.84 | 3.10 | 0.37 |
| JSN | 2.20 | 3.03 | 0.36 | 2.64 | 1.62 | 0.26 |

However, in real conditions, a file rarely consists of only one data type. For example, a common case is the distribution of an application package or library. This file will consist, for example, of 50% binary files and 50% text data. In this case, we can multiply the resulting tables by the vector (0.5, 0.5, 0) and apply the formula (1) to sum up the result. For simplicity of calculations, we took the weighting coefficient equal to 5. The final value is shown in table 3.

TABLE III. COMPRESSION TIME COMPARATION

| brotli | bzip2 | gzip | lzma | zstd | snappy |
|---|---|---|---|---|---|
| 0.94 | 1.33 | 2.48 | 1.27 | 1.85 | 2.44 |

Thus, we can conclude that for our conditions, the gzip algorithm will be the most profitable, which will provide the strongest compression in the shortest time.

## V. EXPERIMENT

We implemented the developed algorithm of the compression method selection in Python programming language, to demonstrate it. Metrics and results collected were presented in tables 1 and 2.

The main program is a script that receives information about the compressed file or directory, the name of the output file and the weighting factor from formula 1. This coefficient helps to adjust the preference for compression time or compression ratio of the file. With a value of 5, preference will be given to the optimal algorithm for two parameters.

An example of the launch is presented on the listing.

```
> python3 adaptive.py -i test_data/text-archive.tar -level 1
File contains: 0.0% JSON, 55.3% BIN, 44.7% TXT and 0.0%
UNKNOWN                                                 data
Prefered               algorithm:              Algo.snappy
Original        size         =         642.50           mb
Output       file:       test_data/text-archive.tar.snappy
Compressed   size   =   366.74   mb,   algorithm   =   snappy
Ratio                        =                         1.75
Elapsed time = 5.69
```

The input is a tar file containing 55.3% binary and 44.7% text data. The system suggested using the snappy algorithm with a weighting factor of 1.

Repeating the launches with different values of the weighting factor, we obtained the following results:

TABLE IV. EXPERIMENTAL RESULTS

| Level | Algorithm | Compression | Time |
|---|---|---|---|
| **1** | snappy | 1.75 | 5.54 |
| **5** | gzip | 2.41 | 7.82 |
| **11** | zstd | 3.40 | 34.17 |
| **24** | lzma | 3.10 | 52.77 |

As can be seen from the obtained results, the algorithm is able to propose an optimal algorithm in case when the file types are correctly recognized. The inaccuracy of type recognition led to an increased execution time for LZMA, while the compression ratio worsened comparing to zstd.

## VI. METHODS FOR CHOOSING THE BALANCING STRATEGY AND EVALUATING SERVERS PERFORMANCE

### *A. Balancing strategies classification*

In a distributed system, it is necessary not only to optimize the work with the data itself as much as possible, but also to adequately select the most productive server according to the criteria that can quickly complete the task. For example, receiving and processing a heavy file.

Server selection criteria and balancing strategies, in turn, are divided into classes [6]:

- *Static.* Static algorithms and strategies are based on the estimation of server resources and its configuration (the number of hops, connection bandwidth, hardware specifications – CPU, available memory). This approach takes the resource component for evaluation, but indirectly affects availability, because the server response time also depends on the server capacity and its workload.
- *Statistical.* Statistical approaches calculate performance based on past data by estimating latency and throughput. The main disadvantage of this approach is that it works unstable and unreliable if the traffic is unstable. Also, more additional calculations are required to predict performance on more heterogeneous load.
- *Dynamic.* Dynamic approaches continuously perform periodic measurements to estimate the current state of the network and the state of the server. Measurements can provide a review of the resources' availability, reflect the state of the network on the server. An example of such measurements is ping execution and measurement of the response delay. However, there is a disadvantage. For example, if the network state changes very quickly during the transfer of a file between the client and the server. In this case, for a more objective estimation of the network state, it will be necessary to increase the number of measurements, which will additionally load the server and distort the evaluation result.
- *Distributed.* The feature of distributed approach is that all nodes are distributed into clusters, within each cluster there is a central node that distributes requests. Thus, all clusters are responsible for load balancing.
- Semi-distributed. With a semi-distributed approach, all nodes are shared to distribute requests.
- Non-distributed. In the non-distributed approach, a centralized node receives all requests and distributes them among the others.

Various types of balancing strategies allow you to estimate the resource potential or availability of the server [11, 12].

In this paper static and dynamic metrics were used, as they are the most informative and stable for servers' performance and workload estimation. Also, the estimation of both server resources and their availability has been made.

In general case, metrics are classified [13] into:

- Client's. Network and server performance metrics are usually collected using the active probes method. These can be: the geographical location of the node, the number of hops, Round Trip Time (RTT, shows the time spent sending the signal and the time it takes to confirm that the signal was received), delay, bandwidth.
- Server load: response time, delays.
- RR DNS: contains information about the chain of domain names. The metric defines all the attributes of a domain name, such as NAME, TYPE, CLASS, TTL, RD length, and DATA).
- Router Metrics: IPv6 allows you to assign an Anycast address to one or more interfaces, then routers select the nearest interface. This, in turn, is determined by the number of transitions [14].

Static metrics and server load metrics have been used in this work.

## VII. Choosing the Balancing Strategy

Client metrics are designed to evaluate heterogeneous servers in a distributed environment, the response time of which depends on the characteristics of both the server and the network: network routes, bandwidth, and the load of transmission channels.

The time value T [11] specified in formula (2) will be used to calculate the delay and processing time of a client request:

$$T = T_{DNS} + T_{connect} + T_{remaining} * \left(\frac{\hat{N} - N_{latency}}{N - N_{latency}}\right) \qquad (2)$$

where $T_{DNS}$ – DNS lookup time (in this case, the total time taken from traceroute),

$T_{connect}$ – time to establish a TCP/IP connection,

$T_{latency}$ – delays from the moment of sending the request to the moment of getting the first response,

$T_{remaining}$– time to receive the remaining packages,

$\hat{N}$ - the size of the object in bytes that is transmitted over the network,

$N_{latency}$ – the object size in bytes received on the first response,

$N$ – total number of bytes transmitted over the network,

The formula (3) is used to calculate the throughput:

$$BW = \frac{bytes}{T_{latency} + T_{remaining}} \qquad (3)$$

where $bytes$ – bytes transmitted,

$T_{latency}$ – delays from the moment of sending the request to the moment of getting the first response,

$T_{remaining}$ – time to receive the remaining packages.

Dynamic hybrid metric has been used as an active measurement, which is to select n servers with the fastest response, to request a fixed-size test object and to calculate throughput [13].

For an objective estimation of the servers' availability, it is necessary to analyze together bandwidth, latency, total memory, free memory, system load, the number of processors, since there are no guarantees of fast data transfer with a high value of only one of the parameters. Each of these criteria has its own priority when making the final decision on choosing a server. For example, if the free memory of one server significantly exceeds the memory of another, but the bandwidth of this server is significantly lower, then the duration of transfer of a heavy file will obviously slow down the server, due to this, the advantage in the amount of available memory is leveled.

Therefore, it is necessary to perform a multiple-criteria decision-making (MCDM) or multiple-criteria decision analysis (MCDA). The most optimal algorithm for performing this task is ELECTRE IS, which allows to set weight characteristics for each of the criteria and threshold values to make a ranked list of the most optimal servers [15, 16].

## VIII. Experiment

To conduct the experiment, a distributed system of servers connected accordingly the "Star" topology was organized (see Figure 5). The client servers are connected to the main load balancing server (load balancing middleware), which analyzes the state of the servers and selects the most productive client server using the multi-criteria analysis and decision-making method ELECTRE IS [15, 16].

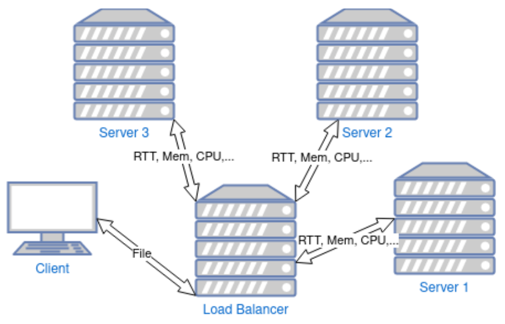

Fig. 5. The experiment network diagram

The ELECTRE algorithm allows to set weight characteristics for each of the criteria, or in terms of the method of alternatives:

-  *veto boundary for the criterion that establishes a critical difference between alternatives and imposes the statement that one alternative is "no worse than the other";*
-  *the indifference boundary for the criterion, illustrating the non-critical difference between the estimates of alternatives;*
-  *the preference boundary, which allows to make a strict conclusion of the preference of one alternative to another.*

The algorithm receives a dictionary, where the keys in the form of IP addresses, and the values is an array of characteristics: the number of cores, memory in bytes, free memory in bytes, the average server load in 1 minute, 5 and 15 percent, RTT in seconds, the number of hops, the bandwidth in bytes/second of client servers.

The weighting factors are set empirically for each of the parameters in accordance with their importance when choosing a server. The weighting factors are shown in Table 4. For example, free memory, bandwidth and server load will have more weight than the number of cores, since the server may have the largest number of processors, but it will not have enough memory to process client request. Or it will have the largest amount of RAM, while the tasks performed on the server are currently very heavily load the system as a whole, so performance will drop, which will lead to an increase in the processing time of the client request, a decrease in bandwidth, and so on.

TABLE V. ELECTRE-IS COEFFICIENTS

| Boundaries / Criterias | weights | vetoes | preference | indifference |
|---|---|---|---|---|
| **Amount of proc** | 0,01 | 2 | 2 | 0,6 |
| **RAM** | 0,05 | 2097152 | 4194304 | |
| **Memory** | 0,35 | 1048576 | 10485760 | |

|  |  |  |  |  |
|---|---|---|---|---|
| **CPU (1 minute)** | 0,1 | 10 | 20 |  |
| **CPU (5 minute)** | 0,05 | 8 | 10 |  |
| **CPU (15 minute)** | 0,02 | 8 | 8 |  |
| **RTT** | 0,2 | 3 | 10 |  |
| **Hops** | 0,06 5 | 2 | 3 |  |
| **Bandwidth** | 0,15 5 | 0,7 | 1,0 |  |

An agent is installed on client servers that sends static server metrics that characterize resources.

The load balancer is an asynchronous web server. The function of the web server is to collect information about each client by receiving messages, storing this information in the database and performing continuous updates of information about the status of the servers using a worker running in the background, which updates every 2 minutes.

During the experiment, several servers were connected and the best one was selected. If you run the client on the same server as the main one, then the local address and the local server are selected, since it has the largest amount of memory (304 GB and the least delay in data transmission).

```
Input    data:    {'192.168.56.222':    [4,    15.542884826660156,
83.07524108886719, 1.0, 0.75, 0.0, 21.08, 3, 0.7390146471371505],
'192.168.15.119': [8,  8.0,  18.13378143310547, 19.219970703125,
16.961669921875,    17.962646484375,    0.04085183143615723,    3,
0.7390146471371505],    '127.0.1.1':    [8,    15.335712432861328,
304.6867904663086, 9.844970703125, 8.642578125, 6.121826171875,
0.025,     1,     0.7264397905759162],     '192.168.99.4':     [8,
5.487361907958984,    0.32358551025390625,    51.325593900954146,
74.63984301576716,      35.318100716144656,      53.694,       2,
0.7390146471371505]}

The best server is 127.0.1.1
```

As soon as the server is disconnected, the server with address 192.168.56.222 is selected, since it has the largest amount of memory and is the least loaded.

```
Input  data:  {'192.168.15.119':  [8,  8.0,  18.133399963378906,
18.609619140625,        17.352294921875,         17.877197265625,
0.03517770767211914, 3, 0.7390146471371505], '192.168.56.222': [4,
15.542884826660156, 83.07524108886719, 1.5, 0.75, 0.0, 20.453, 3,
0.7387687188019967],  '192.168.99.4':  [8,  5.487361907958984,
0.32358551025390625,    49.91594063370359,    12.12770301393315,
45.3839858037852,      73.348,      2,      0.7390146471371505]}
The best server is 192.168.56.222
```

## IX. CONCLUSION

This article presents methods for accelerating data exchange in a distributed file system by choosing an optimal file compression algorithm and choosing the optimal server. ELECTRE-IS is used to select the optimal server. Adaptive compression was also implemented, which allowed to reduce the overhead of compression and at the same time ensure optimal use of traffic.

## X. FUTURE WORKS

For the adaptive compression system, it is planned to add more accurate file types' recognition by their contents in the future. One of the ways to implement it is to create a hash function that establishes a correspondence between random file instances and recognized types. This will help resolve the mismatch between the mime-type values and the file contents. Determining the type directly affects the ability to compress the file and the optimal choice of algorithm.

In the future, it is planned to introduce an additional module for node's health estimation in real time for the load balancer to increase the speed of the algorithm's response to changes in the network state. It is also planned to conduct a number of experiments and studies with other weighting factors to improve the ELECTRE-IS algorithm.

## XI. REFERENCES

[1] Mahoney, M. "10 gb compression benchmark".

[2] Sucu, Sezgin, and Chandra Krintz. "Ace: A resource-aware adaptive compression environment." Proceedings ITCC 2003. International Conference on Information Technology: Coding and Computing. IEEE, 2003.

[3] Gwertzman J. S., Seltzer M. The case for geographical push-caching //Proceedings 5th Workshop on Hot Topics in Operating Systems (HotOS-V). – IEEE, 1995. – C. 51-55.

[4] Yan Y. et al. An experimental case study on the relationship between workload and resource consumption in a commercial web server //Journal of Computational Science. – 2018. – T. 25. – C. 183-192.

[5] de Freitas J. F. G. et al. Sequential Monte Carlo methods to train neural network models //Neural computation. – 2000. – T. 12. – №. 4. – C. 955-993.

[6] Golchi M. M., Saraeian S., Heydari M. A hybrid of firefly and improved particle swarm optimization algorithms for load balancing in cloud environments: Performance evaluation //Computer Networks. – 2019. – T. 162. – C. 106860.

[7] Mahmood A., Khan S. A., Bahlool R. A. Hard real-time task scheduling in cloud computing using an adaptive genetic algorithm //Computers. – 2017. – T. 6. – №. 2. – C. 15.

[8] Blelloch, E., 2002. Introduction to Data Compression, Computer Science Department, Carnegie Mellon University.

[9] Alakuijala, Jyrki, et al. "Comparison of brotli, deflate, zopfli, lzma, lzham and bzip2 compression algorithms." Google Inc(2015): 1-6.

[10] Satopaa, Ville, et al. "Finding a" kneedle" in a haystack: Detecting knee points in system behavior." 2011 31st international conference on distributed computing systems workshops. IEEE, 2011.

[11] Dong Y., Zhang Z. L., Hou Y. T. Server-based dynamic server selection algorithms //International Conference on Information Networking. – Springer, Berlin, Heidelberg, 2002. – C. 575-584.

[12] Balasubramanian J. et al. Evaluating the performance of middleware load balancing strategies //Proceedings. Eighth IEEE International Enterprise Distributed Object Computing Conference, 2004. EDOC 2004. – IEEE, 2004. – C. 135-146.

[13] Dykes S. G., Robbins K. A., Jeffery C. L. An empirical evaluation of client-side server selection algorithms //Proceedings IEEE INFOCOM 2000. Conference on computer communications. nineteenth annual joint conference of the IEEE computer and communications societies (Cat. No. 00CH37064). – IEEE, 2000. – T. 3. – C. 1361-1370.

[14] Andrews M. et al. Clustering and server selection using passive monitoring //Proceedings. Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies. – IEEE, 2002. – T. 3. – C. 1717-1725.

[15] Demidovski A. V. Comparative analysis of multicriteria decision making methods: ELECTRE, TOPSIS and ML-LDM // International Conference on Soft Computing and Measurement. – Federal State Autonomous Educational Institution of Higher Education St. Petersburg State Electrotechnical University LETI named after V.I. Ulyanov (Lenin), 2020. – T. 1. – C. 234-237.

[16] Figueira J. R. et al. ELECTRE methods: Main features and recent developments //Handbook of multicriteria analysis. – 2010. – C. 51-89.