Compas'2016 : Parallélisme/ Architecture / Système
Lorient, France, du 5 au 8 juillet 2016

Performance Evaluation of NFS over a Wide-Area Network

Abdulqawi Saif[1,2,3,4], Lucas Nussbaum[1,2,3]

[1]Université de Lorraine, LORIA, France
[2]Inria, Villers-lès-Nancy, France
[3]CNRS, LORIA - UMR 7503, France
[4]Xilopix, Epinal, France
{firstname.lastname}@loria.fr

Abstract
The Cloud and Big Data movements triggered a move to more centralized, remote storage resources. However, the use of such remote resources raises a number of challenges at the protocol level. In this paper, we first evaluate the influence of several factors, including network latency, on the performance of the NFS protocol. Then, we explore how statistical methods such as a fractional factorial design of experiments could have helped to drastically reduce the number of required experiments while still providing a similar amount of information about the impact of the factors.

Keywords : performance evaluation ; design of experiments ; network latency ; NFS

---

1. Introduction

The Big Data movement has put a lot of focus on the importance of data, and on the importance of proper data management approaches. It encompasses changes in the way we store data, by moving from traditional expensive storage arrays to more cost-effective, scalable and flexible Software-Defined Storage solutions such as Ceph [20], GlusterFS [1] or iRODS [14]. At the same time, there is a push towards moving our computing and storage facilities from small on-premises datacenters to larger datacenters or Cloud offerings in order to reduce costs and increase reliability and elasticity.
This move towards remote and centralized storage resources raises a number of challenges when aiming at maintaining high-performance access. The protocols that were designed to access storage resources over a local network are not necessarily able to perform efficiently when network latency increases due to the need to transfer data over hundreds or thousands of kilometers. Modern storage systems often provide an object-based interface, where storage objects are read or written using GET/PUT methods that are able to perform well over high-latency network. However, most applications still have not been ported to such interfaces, and still rely on traditional POSIX-like directories and files access. Several designs are possible to provide that interface, such as relying on a kernel driver on the client side (which can be a limitation when supporting a wide range of systems), or using a FUSE filesystem (which has performance impacts). As a result, most modern Software-Defined Storage solutions still

provide a traditional NFS or CIFS interface (e.g. GlusterFS [1]), that provides an unintrusive way to access storage from clients.
In this paper, we illustrate the challenges raised by the use of a traditional NFS interface in a high-latency WAN setup. After providing some context and related work in Section 2, we present a set of experiments that demonstrate the influence of several factors, including network latency, on NFS experiments in Section 3. Finally, in Section 4, we explore how statistical methods could be used to reduce the number of experiments in such many-factor evaluations. We conclude the paper and highlight some opportunities for future work in Section 5.

2. Context and Related Work

The Network File System (NFS) was introduced in 1984 to allow sharing resources in a network for heterogeneous client machines [15]. Its first public version (NFSv2) had several limitations (no asynchronous write operations). NFSv3 (1988) overcomes this and added support for the TCP protocol. NFSv4 (2000) came with additional improvements such as compound RPCs to divide any operation into smaller ones, additional security features, client delegations to authenticate the clients to do some operations without contacting the server [4, 16], altogether with other features in order to improve the overall performance.
Many studies have focused on the evaluation of the performance of NFS. Chen et al. [6] made an intensive comparison of NFSv3 and NFSv4.1. They also tested the newly implemented features of NFSv4.1 as well as tuning NFSv3 & NFSv4.1 to increase the throughput performance. They show that the performance of NFSv3 and NFSv4.1 depends on the network latency, i.e., NFSv4.1 was faster on a high latency network and slower than NFSv3 on a network with a lower latency. The same authors additionally focused on the performance of NFSv4.1 [5] by studying its behavior against several factors in Linux as the kernel version and the memory management. Radkov et al. [13] compared several versions of NFS (v2, v3 and v4) with iSCSI as representatives of file and block storage systems, and concluded that NFS is comparable with iSCSI in terms of performance. Furthermore, Martin et al. [11] tested NFS sensitivity against latency. They showed that NFSv3 can tolerate high latency more than NFSv2. Others have focused on performance evaluation of client-side tuning. Ou et al. [12] tested NFSv4 besides other storage file systems such as iSCSI and Swift in the context of cloud storage systems. Their important finding, which is related to our context, shows that the client-side configuration plays a significant role over the performance of IP-based storage systems like NFS. Lever et al. [10] uncovered several performance and scalability issues in the Linux NFS client that affected write latency and throughput.
The main difference between the experiments presented in this paper and the ones presented in previous works is the focus on a high latency environment, using a realistic distributed testbed instead of using an emulated network. Also, we use statistical methods (fractional factorial and full factorial designs). Those are rarely used in Computer Science, but we could find a few examples: Videau et al. [19] used a fractional factorial model to design their multi-parameter experiment in order to determine the impact of each parameter on the results. Khoshgoftaar and Rebours [9] used a full factorial model to evaluate several learning algorithms.

3. Experimental Evaluation of NFS

In this section, we describe a series of experiments performed on the Grid'5000 testbed [3]. We used three sites of the testbed, as shown in Figure 1. Thanks to Grid'5000 capabilities, we deployed our own software environment (Debian *jessie-x64-nfs* image, Linux 3.16.0) and set

up an NFS server on one site. Similarly, we set up an NFS client on the same site as the server, and two other clients on two other sites.

To allow using NFSv4 beside NFSv3, we configured NFS on the clients by enabling the name mapping daemon *IDMAPD* which is required by NFSv4. In addition, we make sure that the server accepts 1M as I/O size value be verifying it on *nfs_xdr* header file. All nodes (server and clients) are connected over a 10 Gbps network (both the Ethernet network on each site and the Grid'5000 inter-site network). This setup is representative of one where an NFS client accesses a server over the Internet, with a high-bandwidth link, and varying network latency (local or remote NFS server).

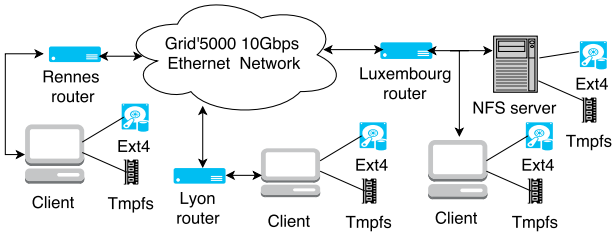

Figure 1: Experiment topology over Grid'5000 testbed

3.1. Experimental setup and factors

NFS performance can be affected by several factors (parameters) and configurations. In this paper, we focus on six factors that can have an impact on NFS performance. Those factors are manipulated on NFS clients. Three factors are NFS *tuning* parameters (changeable through the *mount* command). All factors are listed below, starting by the tuning ones :

NFS Version: [*NFSv3, NFSv4*]. NFSv3 is the stable and widely used version [18] while NFSv4 is the latest version released with promising specifications [17].

Synchronicity of I/O operations: [*sync*, *async*]. They are used on the client side (see `nfs(5)`) to control the behavior of the client when issuing writes. In asynchronous mode, the client may delay sending write requests to the server, and return from the `write` request immediately. In synchronous mode, the client will wait until the server replies to return from the `write` request. This option should not be confused with the server-side `sync/async` option (see `exports(5)`) which controls whether the server must wait until data is written to stable storage (e.g. HDD) before issuing a reply to the client.

NFS I/O size: [*64 KB, 1 MB*]. Those values are used because *64KB* represents the default value of I/O size of several NFS implementations while *1 MB* also represents the maximum I/O value that is used by most NFS servers.

Storage type: [*Hard disk – ext4*, *In-memory – tmpfs*]. We use normal Grid'5000 compute nodes with a single hard disk drive, and not nodes which would provide good I/O performance by spreading data over several disks, as such nodes are not available on all needed sites. Thus, the local hard disk drives of our nodes are likely to be a major performance bottleneck. In order to focus on the protocol aspects of performance rather than just on the performance of local storage devices, we do not limit ourselves to experiments using data stored on *ext4* filesystems backed by hard disk drives, but we also perform experiments using in-memory data, by setting up a *tmpfs* filesystem on nodes.

File size: [*100 MB, 5 GB*]. They are used to represent the small and big files.

Latencies: [*0.027 ms*, *6.87 ms*, *13.9 ms*]. They represent the measured one-way latencies between the server and Luxembourg, Lyon and Rennes clients, respectively.

We have a total of 96 possible combinations of factors (2^5 x 3^1 – five two-level factors & one three-level factor), such as *[tmpfs, NFSv3, sync, 64 KB, 5 GB, Lyon client]*. Furthermore, each combination will be tested with sequential *reading* and *writing* access patterns. We use the *dd* command with */dev/zero* as source and *1MB* as a block size to perform the writing operations. Similarly, the *cp* command is used to perform the reading operations.
The experiments are executed sequentially by changing the access pattern of each experiment, i.e., every experiment performs a write operation before reading the written file again, then the next experiment is similarly executed and so on. Moreover, to determine the exact read/write time of each experiment, all of them are executed with (1) preventing the cache effect on the server as well as on the clients, and (2) re-mounting the corresponding NFS directory after performing the pair (read and write) operations of each experiment. The statistical validity of the results shown next is increased by replicating each experiment five times to get, in total: 960 executions (96 experiments x 2 access patterns x 5 replications).

3.2. Results

This section shows the obtained results. They are presented by separating reading and writing results as shown below.

3.2.1. Reading results

Figure 2 shows that using *tmpfs* provides much better results than *ext4* in all cases, due to the lower speed of HDD. Considering only *tmpfs*, one can see that the best performance occurred when the files are read from the local client. That can enable us to confirm that the high latencies environment (represented by Lyon and Rennes) strongly affect NFS reading performance. For instance, when a client is changed from Luxembourg to Lyon, the throughput is reduced on all NFS versions by at least 8.6x, 7x on figures 2-(A) and 2-(B), respectively.

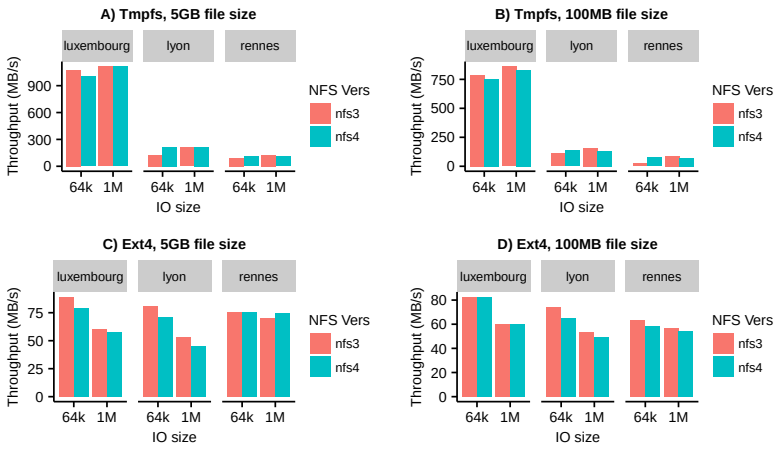

Figure 2: Reading results

Changing NFS versions has only a limited impact on the obtained results. Thus, we can assert that those versions have approximately the same level of sensitivity against the applied latency. However, NFSv4 causes a slight decrease in performance when compared with NFSv3. We also found that the performance is slightly improved on *tmpfs* by increasing I/O size because larger I/O requests reduce the number of I/O requests, and thus the penalty induced by network latency.

3.2.2. Writing results

The writing results are presented in figures 3 and 4. As with reads, the sync writing results also exhibit reduced performance when using an HDD, compared to when using *tmpfs*.
The HDD results are nevertheless affected by the different latency like the results of *tmpfs* which are strongly influenced. The *tmpfs* results show that the cost of changing the client from the local site to another one with *6.8 ms* as latency can decrease the throughput by 17.5x. We

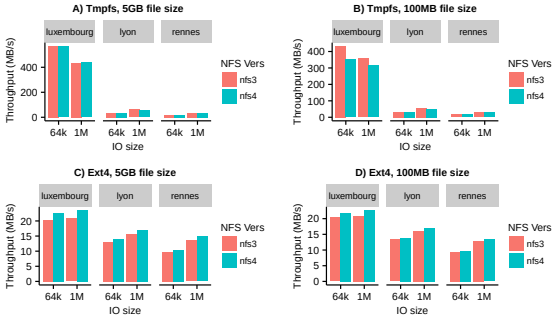

Figure 3: Synchronous write results

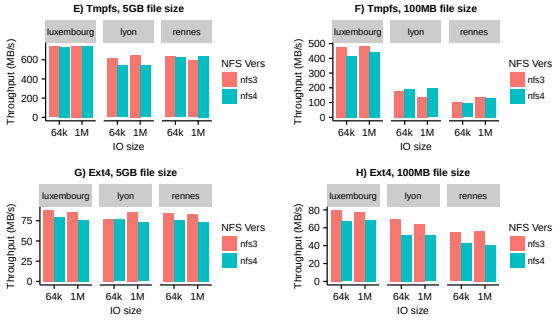

Figure 4: Asynchronous write results

also found that NFSv3 shows a better performance over NFSv4 when the local client writes 100 MB file on a server memory. We hypothesize that this is an inconvenient of NFSv4's compound RPCs when performing write operation on a low latency connection and a high-speed storage material. In contrast, NFSv4 shows a better performance when writing on HDD.

On the other hand, the asynchronous writing results show an increased performance over the synchronous results on all experiments, but its most significant impact is shown on the remote clients. For instance, the highest throughput of Lyon client is raised to 18.9x using the *async* option on *tmpfs* and a 5 GB file. By having `write` requests return immediately, this mode is able to hide the latency between the server and the client. Of course, this could have an impact on data reliability.

4. Statistical Analysis of Results

In most Computer Science papers, experimental results are analyzed in a rather raw way, with only a limited use of statistics (computing averages, medians, etc.). However, statistics can be a useful tool, to help validate results by excluding the effects of chance or other unknown factors, and ensuring that all results are generally coherent. In experiments involving many factors such as those described in this paper, statistics can also contribute to a deeper understanding of the results, by providing a way to identify the role of each factor and their interactions.

In this section, we show how statistical methods [8, 7, 2] (factorial designs) could be used to plan experiments, and identify the relative importance of factors, and their interactions.

First, in section 4.1, we use a *full factorial design* (evaluating all combinations of factors). This design is applied over two-level factors in order to measure the impact of each factor when its value changes from a *low level* to a *high level*, providing a way to make a first pass over all factors and decide which ones to exclude from further experiments (because their impact would be limited). For this reason, the factors from section 3.1 are represented as: (1) **A** : NFS version with (-1) NFSv3 , (+1) NFSv4, (2) **B** : Storage type with : (-1) *tmpfs* , (+1) *ext4*, (3) **C** : Sync option with : (-1) sync , (+1) async, (4) **D** : I/O Size with : (-1) 64 KB , (+1) 1 MB, (5) **E** : File size with : (-1) 100 MB , (+1) 5 GB, and (6) **F** : Client Latency with : (-1) Luxembourg , (+1) Rennes (only Luxembourg and rennes are used as this method is limited to two-level factors).

However, a full factorial design requires a very large number of experiments (320 experiments in our case, as will be explained later). So, in section 4.2, we use a *fractional factorial design* to analyze the same factors using a reduced number of experiments (45 in our case), at the expense of not obtaining information about interactions between all factors.

Ultimately, both models determine if the effects are significant using Student's t-distribution to calculate a 95% confidence interval of each effect.

4.1. Full factorial design

In this section, the full factorial design is used over: $2^k * r$ experiments where k is the number of factors ($k = 6$) and r is the number of replications ($r = 5$), that is, 320 experiments.

The key idea of $2^k * r$ design is to isolate the errors through the different executions. This leads to calculating the estimated behavior of the response variable y, which is the estimated time to read/write a file in a given experiment, using this model:

$y = q_0 + q_i x_i + q_j x_j + q_k x_k + q_{ij} x_i x_j + q_{ik} x_i x_k + q_{jk} x_j x_k + \ldots + q_{ijk} x_i x_j x_k + \ldots + e$

where the response variable y is the estimated time to read/write a file in a given experiment, q_0 is the mean response of the model, x_i represents the correspondent level of factor i (-1 or +1), q_i $with$ $i \in \{A, B, \ldots, F\}$ is the impact of the factor i when changing its value from -1 to +1, q_{ij} $with$ $i, j \in \{A, B, \ldots, F\} \wedge i \neq j$ represents the impact of interaction between the factors i and j, and e represents the model errors.

The result of applying this model on writing access pattern is shown in Figure 5. It shows the effects of all factors under study, and of their interactions. The x-axis of this figure can be read as described in this example: *A, AB, CDE* represent those three cases : *Factor A (NFS versions), interaction between factors A and B (NFS versions, storage type), and interactions between factors A,B and C (NFS versions, storage type and Sync option)*, respectively. Moreover, those effects can be read as follows : *the larger the absolute value, the bigger the influence on the results*.

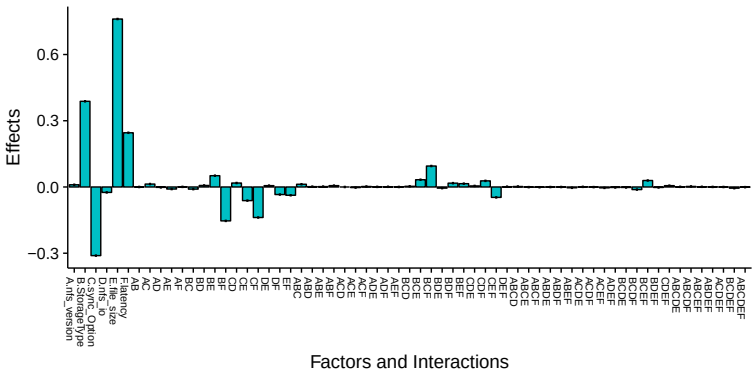

Figure 5: Effects of factors and interactions

As a result, we found statistically that the four main factors that make the most variation on the results are: (1) the file size, (2) the storage systems, (3) the sync options, and (4) the client latency. For example, the first case can be interpreted as : when the transferred file is changed from -1 (100 MB) to +1 (5 GB), it makes the most variation on the result (transfer time) and so on. We also found that, there are some significant interactions such as *BF, CF*, and *BCF*, and some factors that don't have much impact on the results, such as *A (NFS version)*.

4.2. Fractional factorial design

In many cases, full factorial designs generate too many experiments to be usable in practice. One solution to reduce the number of experiments without reducing the number of factors is to use a fractional factorial design.

The general idea is to create a set of experiments that is reduced compared to the one obtained with a full factorial design, but that still, by construction, enables the study of the effect of each factor, even if information about the interactions between some factors is lost.

A fractional factorial design (described in more detail in [8], chapter 19) uses only $2^{k-p} * r$ experiments where p is a positive number less than k chosen to reduce the number of experiments. In what follows, we choose $p = 3$ (and $k = 6; r = 5$ as previously), and thus generate 45 experiments.

To generate the plan, we prepare a sign table for $k - p$ factors and their interactions, i.e., fill $k - p$ columns *(A,B,C)* as truth table variables and calculate the interaction (multiplication) of

A	B	C	D_{AB}	E_{AC}	F_{BC}	ABC
-1	-1	-1	1	1	1	-1
-1	-1	1	1	-1	-1	1
-1	1	-1	-1	1	-1	1
-1	1	1	-1	-1	1	-1
1	-1	-1	-1	-1	1	1
1	-1	1	-1	1	-1	-1
1	1	-1	1	-1	-1	-1
1	1	1	1	1	1	1

Table 1: Fractional design with $k = 6$ & $p = 3$

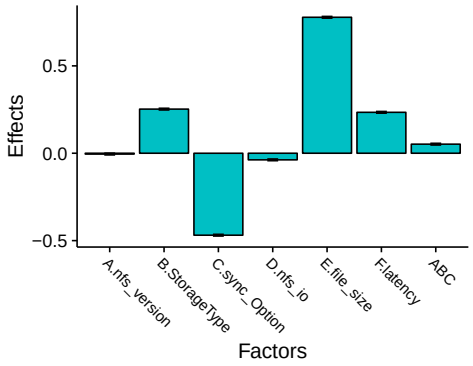

Figure 6: Fractional factorial results

those columns. We then put the rest of factors *D,E,F* over an equal number of interactions as shown in table 1 (for example, we chose to use the AB column for the factor D). Note that as a result, the effects of factors *D,E,F* will be confounded with the interactions *AB, AC, BC*, which means that their respective effects cannot be separated.

In the resulting matrix, each row represents an experiment. For instance, the first row deals with the result of the experiment that has the factors **A,B,C** at *NFSv3, tmpfs, sync* levels and the factors **D,E,F** at *1MB, 5 GB, Rennes* levels, respectively.

Fractional factorial design benefits from some matrix properties to do the calculation. It is shown in table 1 that (1) the sum of every column equals to 0, (2) the sum of product of any two columns also equals to 0, and (3) the sum of absolute values of any column equals to 2^{k-p}.

Furthermore, the allocation of factors on interactions is not unique. For example, instead of having *D=AB, E=AC, and F=BC*, we can use *D=AC, F = BC, and E = BC* and so on. It is possible to compare two different allocations by calculating its *resolution* (see [8], section 19.4).

The result of applying this model on writing experiments is shown in Figure 6. This result shows that the first four important factors that have a significant effect on the writing time are the same factors that were obtained using the full factorial model in the previous section. Thus, this model gives the same conclusion but with reducing the number of experiments from 320 to 45.

5. Conclusion & Future Work

In this paper, we performed a series of experiments on NFS over Grid'5000, with a wide range of parameters before applying the full and fractional factorial models over those experiments. The main originality of our work is firstly making a real and complex case study on how we can use the statistical analysis such as the fractional factorial design to reduce the number of experiments. We show that, instead of doing a full factorial experiment, we could have done a fractional factorial design which would have required only 12.5% of full factorial experiments to obtain the same conclusions as shown earlier. Secondly, we show that the network latency has a significant impact on the performance of NFS, but that some tuning allow us to mitigate its effects: with default parameters, performance on a remote client with 6.8 *ms* as latency is 11.6%, 5.7% of the performance of a local client (client located in the same data center as the server) on reading and writing operations, respectively. But with tuning, the performance can increase up to 20%, 83% of the performance of a local client, which makes the use of NFS over high-latency networks an acceptable option.

We aim at extending that work in the future by building an enlarged benchmark to try tuning

more parameters and decide which is the best tuning configuration that can be used to transfer user data. We are also looking at extending the parameters to include other file systems like Ceph or GlusterFS altogether with using several distributed servers and more powerful storage hardware. All of that should be accompanied by a deeper statistical analysis.

Bibliographie

1. Glusterfs. `https://www.gluster.org/`.
2. Antony (J.). – *Design of experiments for engineers and scientists*. – Elsevier, 2014.
3. Balouek (D.) et al. – Adding virtualization capabilities to the Grid'5000 testbed. *In : Cloud Computing and Services Science*, pp. 3–20. – Springer International Publishing, 2013.
4. Charbon (A.), Harrington (B.), Fields (B.), Myklebust (T.), Jayaraman (S.) et Needle (J.). – NFSv4 test project. – In *Proceedings to the Linux Symposium*, 2006.
5. Chen (M.) et al. – Linux NFSv4.1 performance under a microscope. – In *LISA*, pp. 137–138, 2014.
6. Chen (M.) et al. – Newer is sometimes better: An evaluation of NFSv4.1. – In *SIGMETRICS*, pp. 165–176, 2015.
7. Hinkelmann (K.) et Kempthorne (O.). – Design and analysis of experiments. 2012.
8. Jain (R.). – *The art of computer systems performance analysis*. – John Wiley & Sons, 2008.
9. Khoshgoftaar (T. M.) et Rebours (P.). – Improving software quality prediction by noise filtering techniques. *Journal of Computer Science and Technology*, vol. 22, n3, 2007.
10. Lever (C.) et Honeyman (P.). – Linux NFS client write performance. – In *FREENIX Track, USENIX Annual Technical Conference*, p. 29, 2002.
11. Martin (R. P.) et Culler (D. E.). – NFS sensitivity to high performance networks. *SIGMETRICS Performance Evaluation Review*, vol. 27, n1, 1999, pp. 71–82.
12. Ou (Z.), Hwang (Z.-H.), Ylä-Jääski (A.), Chen (F.) et Wang (R.). – Is cloud storage ready? a comprehensive study of IP-based storage systems. *UCC15*, 2015.
13. Radkov (P.), Yin (L.), Goyal (P.), Sarkar (P.) et Shenoy (P. J.). – A performance comparison of NFS and iSCSI for IP-networked storage. – In *FAST*, pp. 101–114, 2004.
14. Rajasekar (A.) et al. – iRODS primer: integrated rule-oriented data system. *Synthesis Lectures on Information Concepts, Retrieval, and Services*, vol. 2, n1, 2010, pp. 1–143.
15. Sandberg (R.), Goldberg (D.), Kleiman (S.), Walsh (D.) et Lyon (B.). – Design and implementation of the sun network filesystem. – In *Summer USENIX conference*, 1985.
16. Shepler (S.), Eisler (M.) et Noveck (D.). – Network file system (NFS) version 4 minor version 1 protocol. *The Internet Engineering Task Force (IETF)*, 2010.
17. Shepler (S.), Eisler (M.), Robinson (D.), Callaghan (B.), Thurlow (R.), Noveck (D.) et Beame (C.). – Network file system (NFS) version 4 protocol. *Network*, 2003.
18. Tarasov (V.), Hildebrand (D.), Kuenning (G.) et Zadok (E.). – Virtual machine workloads: The case for new NAS benchmarks. – In *FAST*, pp. 307–320, 2013.
19. Videau (B.), Touati (C.) et Richard (O.). – Toward an experiment engine for lightweight grids. – In *MetroGrid*, p. 22, 2007.
20. Weil (S. A.), Brandt (S. A.), Miller (E. L.), Long (D. D. E.) et Maltzahn (C.). – Ceph: A scalable, high-performance distributed file system. – In *OSDI*, pp. 307–320, 2006.