2022 IEEE 16th International Conference on Application of Information and Communication Technologies (AICT) | 978-1-6654-5162-8/22/$31.00 ©2022 IEEE | DOI: 10.1109/AICT55583.2022.10013548

A Novel Blockchain-Integrated Distributed Data Storage Model with Built-in Load Balancing

Obadah Hammoud
Department of Engineering Cybernetics
National University of Science and Technology
Moscow, Russia
obadah.hammoud@gmail.com
ORCID: 0000-0003-2936-832X

Ivan A. Tarkhanov
State Academic University for Humanities,
Federal Research Center "Computer Science and Control"
of Russian Academy of Sciences
itarkhanov@gaugn.ru
ORCID: 0000-0002-8544-8546

***Abstract*— Most companies relay on centralized servers, which are considered as a fast and easy to deploy solution for file sharing, but they have many downsides, like security and trust issues, which can be solved using DLT (blockchain) solutions. However, DLT-based solutions have many challenges, such as nodes orchestration, adjusting reliability and double-spending, as well as load balancing. This research proposes a DLT (Distributed Ledger Technology) based model for files storage for companies using p2p networks, which takes into account both data reliability and storage space efficiency. More efficient data storage is achieved through the use of the Erasure coding technique. For load balancing, a set of algorithms that can be implemented in the form of a smart contract were proposed. The described model can become the basis of the IT infrastructure for trusted exchange of documents, technical and scientific information, games distributions, mobile applications, updates of scientific, regulatory and technical databases in b2b and b2c format**

Keywords—P2P; load balancing; blockchain; data distribution; data backup; virtual disks; virtual clusters; erasure coding

I. INTRODUCTION

Nowadays most companies or consortia rely on centralized servers for files sharing, as they are easy to deploy and cost-effective solutions. However, centralized solutions are an easy target for several attacks, e.g., DoS attacks [1]. From the other hand, centralized solutions may have natural limits for scaling. In such cases, companies usually use distributed solutions, which have advantages as well as disadvantages, which vary by the implementation. DApps are considered as one of the distributed storage technologies which have tracked much attention in the last decade. While DApps offer high reliability and availability, it creates much data redundancy, as data is duplicated on all nodes, which is not optimal for storage space. In this research, we use DApps to create a shared policy among nodes which leads to nodes orchestration. However, when designing a distributed system, there are challenges to be considered. [2]. Table I summarizes these challenges.

The use of an exclusive blockchain (DLT) allows to solve some of these problems, such as fault tolerance and security. Also, it can be used for a wide range of tasks, such as creating a trusted repository of mobile applications [10] or exchanging design or regulatory documentation in organizations that are afraid of substituting important legal documents or other kinds of data [11]. The problems of heterogeneity and scalability are proposed to be solved using Erasure Coding technique [12], and the migration and load balancing control logic can be implemented in the form of a chaincode (or a smart contract). This study presents a model (architecture and set of algorithms) for distributed files sharing between untrusted companies. The main tasks of its application are the exchange of legally significant documents, a distributed storage for mobile applications, and the publication of scientific and technical information.

TABLE I. DISTRIBUTED SYSTEMS MAIN CHALLENGES

Heterogeneity	It is the ability of nodes with various specifications to work together. In some systems, each storage node has a backup node/nodes [3], which requires each backup node to have at least the same storage capacity of the original storage node.
Transparency	The system should appear as a single node for the end user. In Azure [4], it is required to organize nodes into pools and create load balancing rules, so these nodes are accessible using a single IP.
Fault tolerance	It means that the system should be able to work despite the failure of a node or several nodes. Many systems, such as Luster [5] and Ceph [6] depend on a centralized metadata server, which is considered as a single failure point.
Scalability	Adding more resources should be handled smoothly by the distributed system. These resources can be upgrades to existing nodes, such as increasing the storage capacity, or the addition of new nodes. Many systems experience downtime when adding new components.
Concurrency	Multiple users should be able to access the system at the same time.
Openness	Openness is determined by the sharable resources available to the end user. In distributed storage systems, these resources are mainly storage devices.
Migration and load balancing	Workload should be split between existing nodes, so a node shouldn't be handling all the work while other nodes are idle. In systems such as Microsoft Azure or Amazon AWS [7], One commonly used load balancing rule is to switch between nodes using round-robin when storing files. Such a rule might not fully achieve the load balancing concept (possibly unfair load distribution).
Security	Distributed systems have a big plus over centralized solutions regarding security. However, in 2016, Linux mint servers got hacked [8], despite Linux mint uses over 120 mirror servers [9] distributed across the world.

II. DISTRIBUTED DATA STORAGE MODEL

The architecture of the proposed model consists of the following parts:

1) Storage space: It is formed by the nodes which store the actual data contained in virtual disks. In addition to storing files, these nodes host the metadata, so there is not a single server dedicated for metadata hosting, which prevents bottlenecks. There are no certain limitations on these nodes, so they might have different characteristics (ex: processors architectures, storage capacities, etc.). All these nodes run a specific software in order to interact with each other and with the load balancer
2) Load balancer: It exists in the form of a blockchain smart contract. The load balancer has many responsibilities:
    a. Managing VDs (Virtual disks), including locating, creating and distributing VDs.
    b. Managing files, including creating, updating, deleting, distributing and verifying files.
    c. Managing users and authentications.
3) User interface: It allows the user to interact with the load balancer and nodes, in order to read / add / delete / modify files

Fig. 2 illustrates the architecture of the described model.

The use of Erasure Coding in DApps has been discussed earlier [11]. In this model, file storage consists of virtual clusters (VCs). Each cluster consists of three VDs located on different nodes, so as a requirement, the network must consist of three physical nodes at least. When a user uploads a file, it gets split into 2 halves, and the according erasure coding gets calculated. Thus, each half is saved on a VD, and the erasure coding on the third VD. It is important to note that VDs are statically assigned, so one VD is always used to store the first halves of files, another for the second, and the last VD for the Erasure coding. Fig. 1 shows how the concept is achieved, where *-P1 and *-P2 are parts of the file, and *-EC is the erase coding.

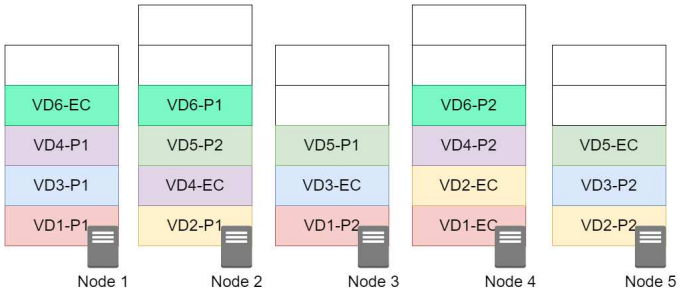

Fig. 1. Files distribution usingVDs.

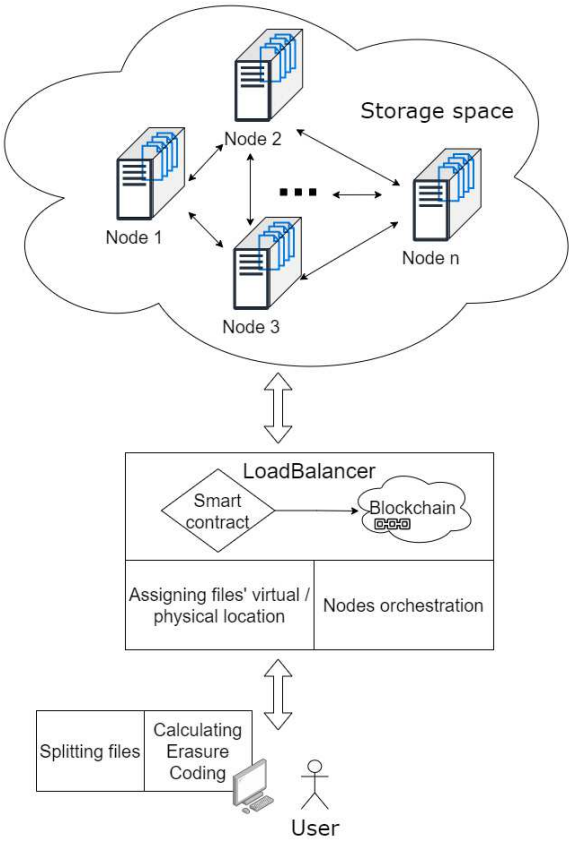

Fig. 2. Proposed system architecture.

Further, in the considered model, the load balancer is implemented using four algorithms:

1) Adding new nodes algorithm (A1)
2) Node resuming algorithm (A2)
3) VC selection for uploading files algorithm (A3)
4) Node recovery algorithm (A4)

A. *Adding new nodes algorithm (A1)*

When adding a new node, the total free space and the maximum virtual devices storage space are calculated. Also, it is required to consider the number of hot VDs (virtual devices which are used the most). Hot VDs should be distributed equally. To determine the list of hot VDs, it is possible to use Least Recently Used (LRU) algorithm, where the number of slots for the LRU algorithm is defined as the total number of clusters divided by the number of nodes. The coldest VDs contain the Erasure Coding.

Migration Algorithm when adding a new node:

1) Sort nodes by the number of hot nodes, in descending order
2) Calculate the ideal distribution of hot VDs:

$$\#(H_n) = \frac{\#(H_N)}{\#(N)}, \forall n \in N \quad (1)$$

Where:

- N: The set of nodes in the system
- H_N: Set of hot VDs in the system
- H_n: Set of hot VDs in the node n

The number of hot VCs in the system is equal to the number of hot VDs divided by 2, because only 2 VDs from a VC can be considered hot as the third VD is reserved for erasure coding.

3) Calculate nodes' ideal free space percentage:

$$S = \frac{\sum_{i=1}^{\#(N)} S_i}{\sum_{i=1}^{\#(N)} S`_i} * 100\ \% \qquad (2)$$

Where:

a. S: Optimal storage percentage for each node

b. S_i: Used space in node i

c. S`i: Total space in node i

4) For each node (n) in the list, starting from the first element, a set of hot devices is selected, using the following rules:

Assuming:

a. D_n: The list of hot devices to be selected from the node n

b. $D`_n$: The list of hot VDs in the node n

c. D: The list of hot VDs already selected before the node n

d. d_2 and d_3 are the two other parts of the VD d

Select the maximum possible D_n devices, which has the following rules:

$\forall\ n \in N, d \in D_n$:

a. $\#(D_n) + \#(D) <= \#(H_n)$ (3)

b. $\#(D`_n) — \#(D_n) >= \#(H_n)$ (4)

c. $d_2 \notin D, d_3 \notin D$ (5)

5) The loop stops once the number of the selected hot VDs is equal to the count of H_n

6) The process repeats for cold devices, so the new node gets the specified number of hot and cold devices.

7) The remaining required number of VDs is achieved with VDs which are not hot neither cold

It is important to note that all erasure coding VDs are considered cold, and they can be selected directly as cold devices, which in turn helps balancing the load.

B. Node recovery algorithm (A4)

Specific heartbeat messages are sent from all nodes to the load balancer regularly. These messages detect that the node is still working. When a node doesn't send a message for a specific number of sequential intervals, it gets transferred to "offline devices" list.

When a node is unreachable, its VDs must be rebuilt into the rest of the existing nodes. When a node goes offline, rebuilding the lost VD is required in order to save data. This process has some requirements:

1) A VD cannot coexist in the same node with another VD that belongs to the same cluster

2) The distribution of data should be fair enough, as data recovery process should consider all available nodes

This problem can be considered as a game theory problem, where it is required to find the strategy which gives the highest outcome for the system (best VDs distribution). Distributing VDs is a stochastic game, as it is a repeated game, while future actions depend on the current action. In this case, making an action does not lead to a whole new game, but to a game where the set of possible actions is less by 1. The player of this game is the system, and it has a finite set of nodes, and a finite set of VDs. The purpose of this game is to distribute the VDs across nodes in a fair way, considering the 2 conditions mentioned above. This game can be defined by the tuple (Q, N, A, P, R), where:

Q is a finite set of the game states, where in each state there is a distribution of VDs

N is a finite set of players, which is equal to one (the system)

$A= A_1 \times \cdots \times A_n$, where A_i is a finite set of available actions (ex: rebuild VD_5, VD_7 and VD_{15} in node 1, and VD_1, VD_6 in node2, etc.).

$P: Q \times A \times Q \rightarrow [0,1]$ represents the transition probability function. $P(q, a, \hat{q})$ is the probability of transitioning from state q to state $\hat{q}$ after taking action a

$R = r_1, \ldots, r_n$, where r_i: $Q \times A \rightarrow R$ is the resulting payoff. Payoff in this problem can be defined by several variables: Selecting a node for a VD, where this node has another VD in the same cluster gives a negative payoff. Sending a VD to a node which preserve having a "fair" distribution results in a positive outcome, and so on.

While there is a single-agent stochastic game, this problem is a Markov Decision Process. After defining all the possible states, the path which guarantees having the highest payoff is chosen. With a big number of VDs and/or nodes, the problem cannot be solved in polynomial time. This problem is considered an np hard problem.

Instead of that, we propose the following algorithm:

1) Define the set of VDs to be rebuilt (K_{VD})

2) Define the number of the maximum slots number (number of VDs that can be added) for each node.

3) Define the set of possible VDs that a node can hold.

4) Select one of the VDs from K_{VD} if not empty, otherwise go to step 9

5) Select the node which has the highest capacity to requests number ratio

a. If two (or more) nodes have the same capacity to requests number ratio, select the "colder" node if the VD is hot, otherwise, select the "hotter" node

b. If two (or more) nodes have the same number of hot VDs, select a random one

6) Subtract one slot from the according maximum slots number for this node
7) Subtract one request from all the nodes which had a request from the selected VD
8) Go to 4
9) Stop

Obviously, the complexity of this algorithm is O(n).

In general, we can assume that the nodes are supposed to have enough more space than what is needed to recover a lost node (the memory remaining after losing and rebuilding a node should be enough to create new several VDs in each node). To test this algorithm, a problem is proposed, where there are exactly 2 available virtual slots in the whole system after rebuilding all the lost VDs. This can be challenging, as the system shouldn't have conflicts resulting from positioning 2 virtual devices from the same cluster in the same node, and the resulting free space (number of virtual slots) is relatively small. A4 source code with a working example can be found in this link [13].

C. *Node resuming algorithm (A2)*

When a node is connected again, the resume process starts:

1) All the hashes of the existing VDs are calculated
2) It sends a hello message to the load balancer, along with the hashes of VDs it has
3) The load balancer compares the hashes with the recent version to check for VDs which have changed since the node got offline, and sends a message to the nodes which have these VDs
4) These nodes transfer the required VDs directly
5) Once the copy process has finished, these nodes remove the virtual devices to free space, and the load balancer updates its records
6) During the process, if the user tries to edit / add / delete a file in a VD, a temporary buffer for the device being transferred is created, and all new modifications are written in it (a configuration file illustrates the process + the new files if exist). Once, the transfer of this device is done, the new modifications get added

D. *VC selection for uploading files algorithm (A3)*

VC selection for uploading files:

1) Load balancer selects the VC with the most free space
2) If it doesn't have enough space to store the new file, the load balancer selects the top 3 nodes which have the most free space, and creates a new virtual cluster

Uploading a file:

1) The user sends a request to the load balancer, containing the total file size, total hash, blocks hashes, file name and the desired location
2) Load balancer verifies the user and the request, and checks if file exists: if yes, go to step 3. Otherwise, go to 5
3) Send the user "FILE_EXISTS" message, and stop
4) Load balancer selects which VDs to use, based on Virtual Cluster Selection for Uploading Files Algorithm
5) Load balancer stores a file storage request with the information sent from the user + the selected nodes and VDs
6) Load balancer sends where to store the file as three parts. Message format:

```
<UPLOAD_LOCATION>
        <REQUEST>${request_id}</REQUEST>
        <NODE>
                <ID>${node_id}</ID>
                <VD>${vd_id}</VD>
                <BLOCK>${part}</BLOCK>
        </NODE>
        <NODE>....</NODE>
        <NODE>...</NODE>
</UPLOAD_LOCATION>
```

7) The user sends the files parts to the desired locations, along with the request_id and the according file parts hashes to each node
8) When a node receives the request, it checks the load balancer if the request id exists, and checks which hash it is expecting
9) If the request doesn't exist, send error message to the client and stop, else, accept the file
10) The load balancer removes the request from its records
11) After receiving the file, calculate the hash.
12) If the hash doesn't match, send error message to the user, remove the part, and stop
13) Send a 200 message to the user and load balancer
14) If the load balancer receives 200 from all nodes, the request gets removed, and it adds the new file entries, and recalculates the according disks free space
15) STOP

III. RESULTS

The proposed architecture's prototype uses Hyperledger Fabric [14] network, and there are no restrictions on the used network (ex: Microsoft Azure, private network, etc.). The advantages of using Hyperledger Fabric are the following:

- It is possible to create DApps without the need of cryptocurrency
- Consensus mechanism is much faster and less resource intensive than other common consensus algorithms like PoW used by other popular blockchain networks such as Ethereum [15]. Consensus in Hyperledger Fabric involves accepting a transaction based on the number of signatures obtained among all possible signatures, adding accepted transactions into a special block, and then validating this block
- Hyperledger Fabric is a limited access blockchain network (transactions do not have to be publicly published), and its architecture supports having several organizations

To test the performance of the described Hyperledger Chaincode compared with other popular blockchain networks, the according chaincode / smart contract containing all the described algorithms was implemented in three different forms: Hyperledger, Hyperledger Besu (Ethereum Enterprise) and Ethereum. The first two of them were deployed on cloud

virtual machines, while the last one was deployed on Ropsten public network. After that, 1000 files with random content were generated, and files were distributed across three virtual machines. Interaction with the chaincode was accomplished with a user client implemented using JavaScript. Table II contains the results of the required time to upload and download these files in the three specified networks. According the table, Hyperleger gave the best results. Time required to upload and download files is not critical for the end user and is comparable to P2P projects.

TABLE II. TEST RESULTS IN VARIOUS NETWORKS

Network		Time in seconds			
		Min	***Max***	***Avg***	***Median***
Hyperledger	Upload	0.535	21.762	1.242	0.585
	Download	0.27	16.615	0.927	0.432
Hyperledger Besu	Upload	2.21	12.214	3.203	3.294
	Download	1.937	26.012	2.986	2.902
Ropsten	Upload	3.512	388.917	27.27899	11.935
	Download	1.142	13.673	1.597874	1.258

As the tests were performed physically on one server, the time of network interaction is not indicated in the tests.

The following considerations were taken into account when implementing and testing the prototype:

1) The system should at least have three nodes, while it is advised to have more than three
2) Nodes cannot be fully used, as losing a node requires rebuilding the lost VDs in the remaining nodes, in order to achieve high availability
3) The maximum file size is determined by the VD capacity * 2, as the file gets split

In full backup systems which have one backup server, the failure of a node activates the backup server. If the backup server after that fails, data is lost. In the proposed system, if a node fails, its data (VDs) gets rebuilt automatically on the rest of nodes. After that, if another node fails, the data are rebuilt again on the rest of nodes, and the process continues until there is only one node in the system, or the free space in the system is over. This means that the proposed system can be more reliable that full backup systems. However, the reliability of the system is not fixed, because it depends on the existing nodes in the system, the number of failed ones, and total free space in the system. A VD is lost permanently in one of two cases:

1) Two nodes fail at the same time, that have common clusters (and only these VCs are affected, while the rest of virtual disks are rebuilt)
2) There is no more enough space to rebuild lost VDs

However, the reliability of the proposed system can be more than the reliability of full backup servers in 2 conditions:

$$R_{Proposed\_system} > R_{full\_backup} \leftrightarrow \begin{cases} \#(N) > 3 \\ \forall n \in N: \quad V`_{N\{n\}} > V_n + \frac{V_n}{\#(N)-1} \end{cases}, \quad (6)$$

where:

- V_n: Number of existing VDs in node n_i
- $V`_n$: Number of free VD slots in node n_i

IV. DISCUSSION

Consideration should be given to other distributed solutions that perform the task of sharing files.

Openstack uses a greedy algorithm to locate what is the best node to put a new instance [16]. This process is time and resources consuming, as it goes through all possible solutions..

Hui Zhao et al. [17] proposed a solution, which considers the disk I/O + CPU usage for virtual machines placement policy. Instead of using a greedy algorithm which estimates all possible ways to decide where to put this VM, they suggested using Ford Fullkerson algorithm to achieve this purpose. This algorithm has the complexity of O(E*N), where N is the number of nodes, and E is the number of required iterations. This study does not fit our model, because:

1) I/O usage is not steady for a given node. We can assume that a newly created VM (Or VD) can have the highest I/O, but once the storage is full, it will consume much less I/O
2) The discussed problem doesn't require CPU processing, (ex: no applications are being executed within the virtual machine / drive)

Another study has a similar idea [18], where authors set a threshold for CPU usage. Once the threshold is exceeded, a VM is moved to another host based on the Disk I/O.

For all methods that depend on moving VMs based on the disk I/O usage, it is important to consider that a VM can have a high disk I/O for a short time, and then another VM can have it, so moving VMs cannot be helpful. On the contrary, moving these VMs frequently can degrade the total performance.

Moon-Hyun Kim el al. [19] considered several parameters, like nodes response time, CPU usage, number of VMs within nodes, etc. This algorithm starts with a list of nodes which can be added, and then performs several tests and reduces the list with every test to end up with the desired node. This solution might select the best option, but it is not guaranteed, as it is not possible to predict incoming traffic destination. Also, the algorithm requires many iterations to choose nodes.

Raid-5 [20] is a popular solution for many companies [21] to manage accessing files in centralized servers containing several disks. Raid-5 uses Erasure coding, by splitting the file among all the disks except for one, where the Eraser Coding is stored. There is no specific disk for storing Erasure Coding, so it gets stored each time on one of them. Despite that our proposed solution is designed to work with decentralized nodes, the storage scheme can be implemented in such cases, where a centralized server uses several disks. However, as the storage model provided by Raid-5 is not dynamic.

Analysis shows that all the discussed solutions require more time to distribute data among nodes, and do not consider specific scenarios, like expanding existing disks. Azure simply destroys the existing disk, and allocates a new virtual disk (there are some few cases where the disk is not destroyed, but it has to be in one of specific regions, and with specific conditions on how much size it is, and how much size to add, otherwise, the disk gets destroyed and recreated [22]).

V. CONCLUSION

This research proposed a new data storage model, which allows reducing data redundancy while maintaining high reliability. This model also allows distributing data and balancing data usage among nodes in a polynomial time. Furthermore the proposed model aims to offer high availability and reliability while maintaining low data duplication. Required protocols and algorithms were also proposed. This model handles load balancing using a blockchain smart contract and distributes data on fair bases regarding data size and access frequency, without using a greedy algorithm. At the same time proposed model stores data in virtual clusters and disks, which makes it easier to manage. Hyperledger Fabric was selected for this task, and the chaincode was written in Go.

The model has the following advantages:

1) Having a VD dedicated for erasure coding can help balancing the load on nodes
2) Managing files is easier when grouped within VDs
3) This model does not use virtual machines, which means more speed and less resources usage
4) This model supports backups, without having dedicated backup servers
5) There are no restrictions on the capacities of nodes, so nodes shouldn't have equal storage spaces to have matching backup nodes. The node which has more storage space holds more VDs
6) In full backup systems, having two storage nodes means having two additional backup nodes, so the total required storage is reduced by 25%
7) The more nodes exist in the system, the more reliability is achieved, and that is because of two factors:
   - There will be more possible virtual empty slots were lost VDs can be rebuilt
   - The system will survive losing several nodes

However, the model has the following limitations:

1) The loss of two nodes simultaneously might lead to permanent loss of data
2) Data rebuilding process requires extra time, while files replicas are directly accessible in full backup systems

The proposed architecture will make it possible to implement a trusted exchange of documents, technical and scientific information, distributions of games, mobile applications, updates of scientific, regulatory and technical databases in b2b and b2c format.

The future work involves the development of security issues, metadata and versioning storage and optimal blockchain architecture.

REFERENCES

[1] M. Sachdeva, G. Singh, K. Saluja, and K. Singh, "A Comprehensive Survey of Distributed Defense Techniques against DDoS Attacks," Apr. 2022.

[2] K. Nadiminti, M. Assuncao, and R. Buyya, "Distributed Systems and Recent Innovations: Challenges and Benefits," InfoNet Mag., vol. 16, Jan. 2006.

[3] X. Li, Y. Qi, P. Chen, and X. Zhang, "Optimizing Backup Resources in the Cloud," in 2016 IEEE 9th International Conference on Cloud Computing (CLOUD), Jun. 2016, pp. 790–797. doi: 10.1109/CLOUD.2016.0109.

[4] "Cloud Computing Services | Microsoft Azure." https://azure.microsoft.com/en-us/ (accessed Apr. 21, 2022).

[5] R. Salunkhe, A. D. Kadam, N. Jayakumar, and S. Joshi, "Luster a scalable architecture file system: A research implementation on active storage array framework with Luster file system," in 2016 International Conference on Electrical, Electronics, and Optimization Techniques (ICEEOT), Chennai, India, Mar. 2016, pp. 1073–1081. doi: 10.1109/ICEEOT.2016.7754852.

[6] S. Weil, S. Brandt, E. Miller, D. Long, and C. Maltzahn, Ceph: A Scalable, High-Performance Distributed File System. 2006, p. 320.

[7] "Cloud Computing Services - Amazon Web Services (AWS)," Amazon Web Services, Inc. https://aws.amazon.com/ (accessed Apr. 22, 2022).

[8] "Beware of hacked ISOs if you downloaded Linux Mint on February 20th! – The Linux Mint Blog." https://blog.linuxmint.com/?p=2994 (accessed Apr. 21, 2022).

[9] "Mirrors - Linux Mint." https://linuxmint.com/mirrors.php (accessed Apr. 21, 2022).

[10] J. Park, S. Lee, G. Kim, and J. Ryou, "Decentralized Blockchain-Based Android App Store with P2P File System," in Advances in Computer Science and Ubiquitous Computing, vol. 536, J. J. Park, D.-S. Park, Y.-S. Jeong, and Y. Pan, Eds. Singapore: Springer Singapore, 2020, pp. 525–532. doi: 10.1007/978-981-13-9341-9_90.

[11] O. Hammoud, I. Tarkhanov, and A. Kosmarski, "An Architecture for Distributed Electronic Documents Storage in Decentralized Blockchain B2B Applications," Computers, vol. 10, no. 11, p. 142, Nov. 2021, doi: 10.3390/computers10110142.

[12] S. B. Balaji, M. N. Krishnan, M. Vajha, V. Ramkumar, B. Sasidharan, and P. V. Kumar, "Erasure coding for distributed storage: an overview," Sci. China Inf. Sci., vol. 61, no. 10, p. 100301, Oct. 2018, doi: 10.1007/s11432-018-9482-6.

[13] "Hammoud, O., DistributedFileSystem" https://github.com/Obadah-H/DistributedFileSystem/blob/735131ac2cf1b89e54fbfb25425294b40a7015d2/jupyter/nodes_recovery_algorithm.ipynb (accessed Aug. 15, 2022).

[14] E. Androulaki et al., "Hyperledger fabric: a distributed operating system for permissioned blockchains," in Proceedings of the Thirteenth EuroSys Conference, New York, NY, USA, Apr. 2018, pp. 1–15. doi: 10.1145/3190508.3190538.

[15] HYPERLEDGER, "Hyperledger Architecture, Volume 1 Introduction to Hyperledger Business Blockchain Design Philosophy and Consensus," Hyperledger Archit., vol. 1, 2019.

[16] J. Guo, Z.-M. Zhu, X. Zhou, and G.-X. Zhang, An instances placement algorithm based on disk I/O load for big data in private cloud. 2012, p. 290. doi: 10.1109/ICWAMTIP.2012.6413495.

[17] Hui Zhao, Q. Zheng, Weizhan Zhang, Y. Chen, and Yunhui Huang, "Virtual machine placement based on the VM performance models in cloud," in 2015 IEEE 34th International Performance Computing and Communications Conference (IPCCC), Nanjing, China, Dec. 2015, pp. 1–8. doi: 10.1109/PCCC.2015.7410296.

[18] P. N. Sayeedkhan, V. Nanded, M. S. B. S, and V. Nanded, "Virtual Machine Placement Based on Disk I/O Load in Cloud."

[19] M. Kim, J.-Y. Lee, S. Shah, T.-H. Kim, and S.-Y. Noh, "Min-max exclusive virtual machine placement in cloud computing for scientific data environment," J. Cloud Comput., vol. 10, Jan. 2021, doi: 10.1186/s13677-020-00221-7.

[20] Patterson, D.; Gibson, G.; Katz, R., "A case for Redundant Arrays of Inexpensive Disks (RAID)," ACM SIGMOD Rec., vol. 17, Jul. 1988, doi: 10.1145/50202.50214.

[21] Muppalaneni, N.; Gopinath, K., A multi-tier RAID storage system with RAID1 and RAID5. 2000, p. 671. doi: 10.1109/IPDPS.2000.846051.

[22] Microsoft Technicals Papers, "Expand virtual hard disks attached to a Windows VM in an Azure - Azure Virtual Machines." https://docs.microsoft.com/en-us/azure/virtual-machines/windows/expand-os-disk (accessed Apr. 23, 2022).