

PAPER

Optimizing Off-Chain Storage in Blockchain of Things Systems: Implementing Dockerized IPFS for Enhanced Efficiency

Mamoon Aldmour¹,
Rakan Aldmour¹,
A. Y. Al-Zoubi²(✉),
Mohamed Sedky¹

¹School of Digital, Technology, Innovation and Business, The University of Staffordshire, Stoke-on-Trent, UK

²Princess Sumaya University for Technology, Amman, Jordan

zoubi@psut.edu.jo

ABSTRACT

The InterPlanetary File System (IPFS) offers decentralized storage and data sharing, which are critical for the functionality of Blockchain of Things (BCoT) systems. Despite its advantages, IPFS faces challenges such as scalability, latency, and resource management issues that hinder its effective integration into existing blockchain infrastructures. This study explores the implementation of Docker containerization to enhance IPFS performance within BCoT environments. An experimental testbed was established, comprising an IPFS node and an IPFS Cluster peer deployed as Docker containers, to evaluate the latency of file operations across various sizes and analyze containerization's impact on data storage and retrieval efficiency. The proposed Dockerized IPFS implementation demonstrates substantial performance improvements over traditional systems, achieving latency reductions of up to 75% for small files (1–256 KB) and a three-fold decrease for larger files (64 MB). Specifically, write operations were reduced from 1000 ms to 300 ms, while read operations improved by 40%, decreasing from 2500 ms to 1500 ms. Additionally, the containerized approach yielded lower latency than previous standalone IPFS deployments. The study emphasizes the significance of dynamic resource allocation in optimizing resource utilization, thereby enhancing the overall performance of IPFS Clusters within BCoT frameworks. By leveraging Dockerized IPFS, BCoT systems can achieve more efficient off-chain storage solutions, facilitating improved data management and interoperability in decentralized applications.

KEYWORDS

interplanetary file system (IPFS), IPFS-cluster, Docker

1 INTRODUCTION

Interplanetary file system (IPFS) is a decentralized file-sharing protocol with great relevance to Blockchain applications. It is a fundamental departure from the client-server model typical of protocols such as HTTP [1]. Instead, it provides

Aldmour, M., Aldmour, R., Al-Zoubi, A.Y., Sedky, M. (2025). Optimizing Off-Chain Storage in Blockchain of Things Systems: Implementing Dockerized IPFS for Enhanced Efficiency. *International Journal of Online and Biomedical Engineering (iJOE)*, 21(1), pp. 118–131. <https://doi.org/10.3991/ijoe.v21i01.53157>

Article submitted 2024-08-02. Revision uploaded 2024-11-08. Final acceptance 2024-11-09.

© 2025 by the authors of this article. Published under CC-BY.

a peer-to-peer (P2P) file-sharing method that does not rely on centralized servers. This is achieved via multiple underlying technologies, including the Distributed Hash Tables (DHT), MerkleDag data structures, and BitSwap protocol. The DHT allows for efficient discovery and retrieval of content across the network. In contrast, MerkleDag allows for content (i.e., files and directories) to be both identified and linked together in a way that is unique and efficient, using content identifiers (CIDs) [2]. BitSwap, inspired by the BitTorrent protocol, allows for cooperative data exchange between peers to improve the decentralized retrieval of data. The advantages of IPFS are numerous. Firstly, it provides a system that is resistant to censorship and has no single point of failure. Because it is a content-addressing system, every link to a file is unique and cannot be changed; the links provided are permanent and immutable, making them useful for archival purposes. Furthermore, because IPFS is efficient at content distribution, less bandwidth and storage space are necessary to retrieve files since they are distributed only when necessary, avoiding redundancy [3]. In addition, using IPFS means it is possible to use cached content offline, making it ideal for systems with limited or unstable internet access. Overall, IPFS is a pioneering solution that provides a more robust, efficient, and accessible file-sharing method.

This paper presents an experimental study to evaluate the performance of IPFS in private networks. It aims to determine how well IPFS performs in local area networks and what factors influence the performance of IPFS in private networks. It is based on a private network built using Docker-Cluster technology [4]. IPFS Cluster nodes build up a private libp2p-based network. They maintain a shared list of CIDs (content identifiers) and metadata that contains which IPFS nodes currently pin that content. When a new file is added to one of the IPFS Cluster nodes, the cluster coordinates the replication of that content by copying it to the other two IPFS nodes. IPFS Cluster nodes can be configured to pin content in specific locations or across all nodes based on available storage space. This means that the location of the content replication can be managed, and those files can be guaranteed to be kept on nodes with adequate storage [5]. Then, the paper creates files of different sizes between Docker containers while observing each operation's latency. It answers how well IPFS writes and reads the data to and from an IPFS-Cluster private network built using Docker technology and what factors affect IPFS writing and reading performance in private Docker networks. The paper aims to illustrate the feasibility of using IPFS as an alternative to client-server-based file-sharing systems. It will help to evaluate the idea of designing file-sharing applications based on IPFS-Docker for private networks. Also, it may motivate IPFS designers to try new techniques that can improve the file system's performance.

2 LITERATURE REVIEW

Several studies have been conducted to investigate the I/O performance of IPFS storage and to give us some significant insights into its characteristics and limitations. For instance, Shen et al. [6] evaluated the behavior of IPFS from the client's perspective. They investigated how the file size, concurrency, node configuration, and network topology affect the system's throughput, latency, and scalability. Shen et al. used a custom-built client and an IPFS cluster with diverse node configurations to perform their experiments. Their experiments give us essential insights

into how the decentralized storage system behaves under various workloads and scenarios. These findings significantly impact the optimization of IPFS performance and its deployment in real-world applications. One of their findings pointed out that increasing the data size has an essential effect on the IPFS performance, which leads to slower retrieval. The researchers identified that the resolving process (finding the nodes that store the data block) and the subsequent downloading of the IPFS data block are the two main bottlenecks of the system when a file is read from a remote node. IPFS was used to build a decentralized storage system called ASC (academic storage cluster), explicitly designed for the academic community. This ASC system aims to provide a secure, reliable, and scalable storage and sharing service for research data, publications, and other academic artifacts.

The paper [7] presents the results of a pilot deployment of this system in a few universities. The findings reveal that the system can provide a secure, decentralized alternative to the traditional centralized storage solution while encouraging collaboration and knowledge exchange in the academic community. This study contributes to developing decentralized storage solutions for academic and research-related applications. In another study, Lajam and Helmi [8] investigated the performance of IPFS in private networks and the impact of private network characteristics on the system. They identified and simulated the factors that affect the performance of IPFS in a private network, such as network size, node degree and latency, data retrieval time, bandwidth usage, node storage, and ranges. The simulation results reveal that the private network is vital in affecting the IPFS performance and can be tuned to enhance the system performance. For example, they showed that the data retrieval time decreases with an increase in the node degree. IPFS was also used to build a distributed and decentralized storage system that establishes a unified data namespace across multiple clusters [9]. Ahmad et al. studied the performance of reading data in the IPFS cluster. Their experiment showed that many connected nodes do not affect the performance, but the replication factor (how many copies of the same data block are stored in the cluster) does. The IPFS cluster represents a set of nodes forming a single distributed storage in a network, which is different from the regular IPFS private network, where each node has an independent file system in the network.

3 EXPERIMENTAL TESTBED

The experiments are conducted on a host machine with a 4-core 2.5 GHz i5 Intel processor, 12 GB RAM, and an SSD disk. The host machine operates on the operating system Windows 11. Three-node IPFS Cluster containers and three IPFS nodes are deployed on Docker Desktop. The architecture of the IPFS cluster is shown in Figure 1. An additional container for an FTP node will be implemented with VSFTP technology. The IPFS Cluster is configured, and its data storage and information retrieval efficiency are analyzed. The characteristics of the closed network environment are explored, and the roles of data storage are assigned to each node in Docker containers. The analysis might provide some clues to the performance and expansion of the IPFS system in the private network environment and advise on implementing a decentralized storage solution in an enterprise environment.

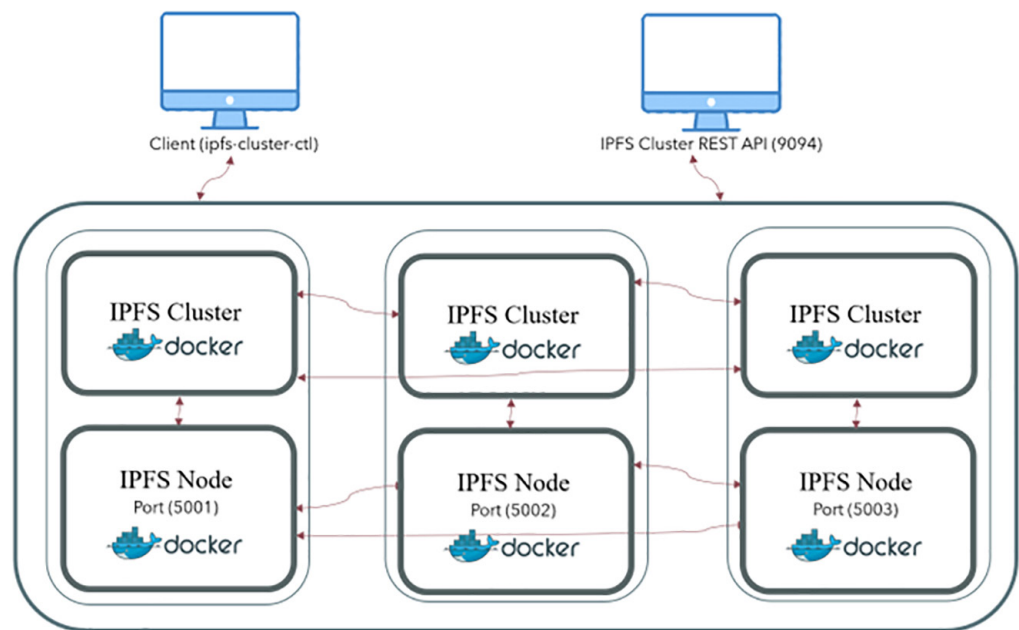


Fig. 1. The architecture of the IPFS cluster

The Docker compose file specifies multiple services corresponding to IPFS nodes (ipfs0, ipfs1, ipfs2) and IPFS Cluster P2P peers (cluster0, cluster1, cluster2). Every service must be containerized and run in isolated execution environments. Each IPFS node (ipfs0, ipfs1, ipfs2) is assigned to exposed ports corresponding to the IPFS swarm, API, and gateway. The clients use these ports to communicate with other IPFS nodes. Read operations are performed by retrieving the data from IPFS. As previously mentioned, clients can access the content stored in IPFS through the exposed gateway (8080, 8082, and 8083). These ports perform read operations on the IPFS (e.g., fetching files or performing content addressing). Write operations are performed when a new piece of data is added to IPFS. Clients need to connect to the IPFS nodes' API (5001, 5002, and 5003) to add a new piece of content. These API endpoints are primarily used to upload files or to add new content-addressed objects.

Interplanetary file system cluster peers (cluster0, cluster1, cluster2) depend on their IPFS node so that they can interact with their corresponding IPFS node to manage the data. IPFS Cluster peers help to do read operations by coordinating the data retrieval from IPFS nodes. When a client needs to get some data from the cluster, the cluster peer can retrieve the data from the IPFS node (CLUSTER_IPFSHTTP_NODEMULTIADDRESS) and then serve the data from the Cluster REST API (9094). So, the cluster REST API will query the IPFS node for the data and then return the data to the client. IPFS Cluster peers handle the write operation by coordinating the content addition among the IPFS nodes. When a client needs to add new content to the cluster, the peer will distribute it to their corresponding IPFS nodes (CLUSTER_IPFSHTTP_NODEMULTIADDRESS). After getting the data, the IPFS nodes add the actual content to the IPFS network and save the data redundantly, thus promoting the availability of the data.

Clients interacted with the IPFS Cluster through the Cluster REST API (9094) to perform the following operations: adding, retrieving, and getting the latest version of data; managing peers; monitoring the cluster status; the ipfs-cluster-ctl command-line tool could also interact with IPFS Cluster to manage peers, query data, and monitor performance. Setting up IPFS in a Docker environment with a cluster

provides many benefits, such as high availability, load balancing, redundancy, fault tolerance, and scale-out. Using the IPFS Cluster, multiple nodes could collaborate and provide decentralized and distributed access to content. This situation was good for scenarios where data needs to be shared and accessed among different parties. For example, IPFS was developed to manage smart contract data. In a cluster, the nodes will play various roles. Tracker nodes track content locations and direct requests to the correct storage nodes. Pin nodes store the content and provide a way to replicate it to avoid a single point of failure. REST API nodes expose the cluster API to the outside world for management and monitoring. Together, all these nodes help improve the reliability and performance of the system.

4 EXPERIMENTAL MATERIAL AND DESIGN

The materials of experiments are a series of data files passed from node to node in a private network. These files are the ones where the writing and reading operations have been applied. There were two sizes of files, small and large. The small file sizes were 1 KB, 4 KB, 16 KB, 64 KB, and 256 KB, and the large file sizes were 1 MB, 4 MB, 16 MB, and 64 MB. The choice of these sizes was inspired by the Lajam and Helmy work. They represent the approximate values of specific data file sizes, which are the subject of their research. As IPFS splits the files into 1 MB blocks, more processing time is expected for large files. These files were created in these sizes by manually filling their contents with random alphabetical characters. In the experiments, the measurements performed for the writing and reading files were the operations carried out to and from the network by interplanetary file system.

That is, the measurements performed were the operations of reading and writing. The IPFS command ADD performed the writing operation. By adding a file, the writing operation was performed on that file. After the file was added to the local IPFS repository of a node, the file was made available to other IPFS network members. The IPFS command performed the reading operation GET. This operation downloads a file from the network to the local IPFS repository of a node and from one or more nodes that have that file. When a file was downloaded to the local IPFS repository of a node requesting the file, that file was then available to be accessed by other IPFS network members, and the number of contributors (file senders) was increased as the number of file owners was increased. Figure 2 shows the experiment operating sequentially by node 1 and node 2. Latency was measured for all operations as the time passed from the moment the user initiated the operation to its completion, i.e., the operation completion time, which included CPU time, I/O waiting time, and network delay.

In the case of writing and the reading operations related to IPFS, the experiment was started with writing content to IPFS, where the writing operation was done on an arbitrary file over three API nodes (5001, 5002, and 5003) using command-line tools such as `ipfs add` to add the content into the network. For reading operations from IPFS, first, the method considered how data would be accessed from the IPFS network, and then the content was retrieved using the command. To access the data from the IPFS network via the gateway ports, the `ipfs cat` command was followed by the CID of the content to be retrieved. Test read operations from the IPFS Cluster was performed by querying the data via the cluster REST API (9094). Every operation was executed individually on a single file, and the time taken was measured and recorded.

5 ANALYSIS AND EVALUATION

The intricacies of IPFS writing operations are shown in Figures 2 and 3, which comprehensively depict the latency for IPFS and FTP writing operations for both small and large file sizes. One of the most critical findings from this experiment is that the latency for writing the files to the local IPFS-Docker network and the FTP container shows that the IPFS latencies, measured in milliseconds (ms), are higher than for FTP. This is because of the extra functions the IPFS writing operation performs compared to the FTP writing operation. For example, the IPFS writing operation splits the large files into blocks, generating the CIDs for each block. On the other hand, the FTP writing function is a simple copy function. However, if we look at the IPFS latencies for small files, the latency variation is quite noticeable because, in this case, we only have one block. However, the latency variation is hardly noticeable for large files since the IPFS writing operation has to handle multiple blocks. Therefore, more I/O disk operations are being performed. This shows that writing a single large file in IPFS is more efficient than writing many small files. The latency graphs for each file system show that writing many large files in IPFS is less efficient. This is likely because the IPFS writing operation has to handle many blocks of a large file, resulting in more I/O disk operations being performed.

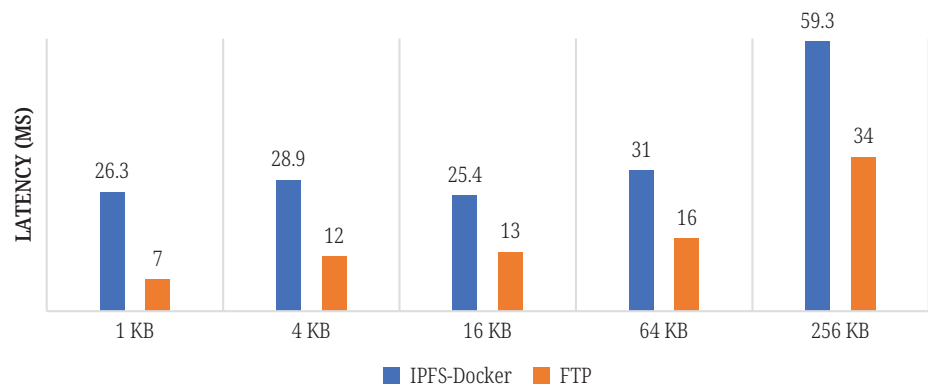


Fig. 2. The latency for the Docker cluster and FTP writing operations for the small-size files

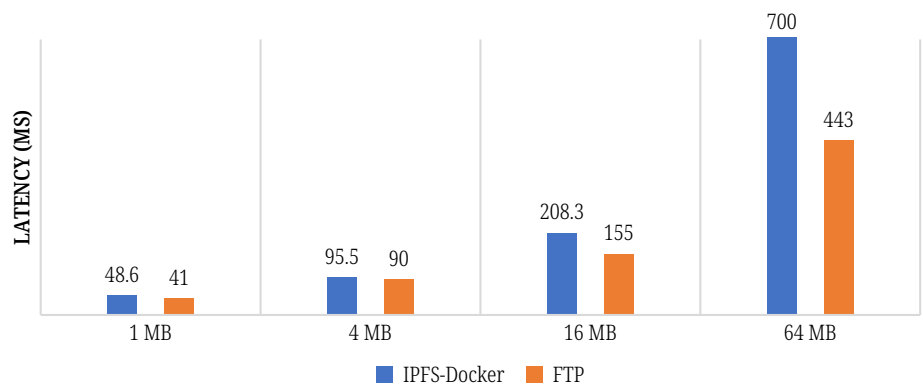


Fig. 3. The latency for Docker cluster and FTP writing operations for large files

Figures 4 and 5 provide a comparative analysis of latency measurements for read operations in IPFS-Docker with Docker vs FTP and for file sizes < 256 KB and 1–64 MB, respectively. In the case of a small file size range (see Figure 4), the latency in FTP is only slightly less than IPFS-Docker (although this difference appears insignificant),

which is most pronounced for files up to 16 KB. However, its latency increases noticeably for sizes up to 256 KB, reducing the performance differential to almost negligible. For larger file sizes in (see Figure 5), the latency through IPFS-Docker is significantly lower than FTP, specifically in the range of files up to 16 MB. This performance differential is most pronounced. By the time we started taking measurements for 64 MB size, the gap due to performance difference between both the systems narrowed down slightly, with FTP having a slight advantage over IPFS-Docker.

These results suggest that, while FTP shows a lower edge for smaller file sizes, IPFS-Docker appears more efficient for larger file sizes closer to 64 MB. It is worth mentioning that this performance characteristic stems from the content-addressed storage system, organized in a distributed architecture of IPFS, which becomes useful as the sizes of file transfers continue to grow. The emerging trends imply that, from a user perspective and depending on the target use case, selecting either file transfer protocol should necessarily consider file size. If it is data transfers of sizes closer to 64 MB, IPFS-Docker shows more significant promise.

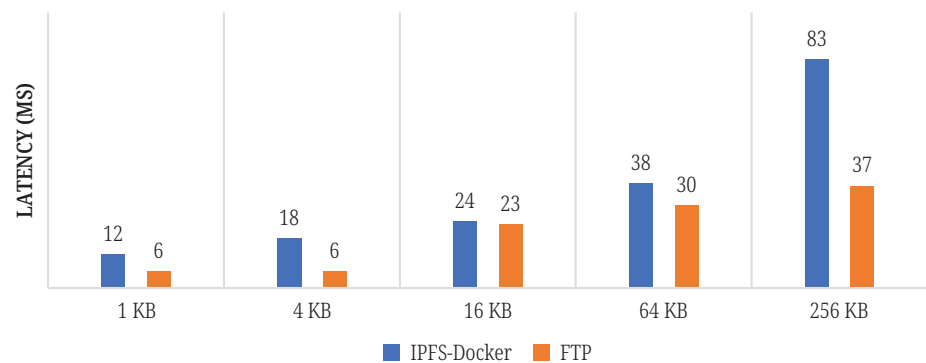


Fig. 4. The latency for the Docker cluster and FTP reading operations for the small-size files

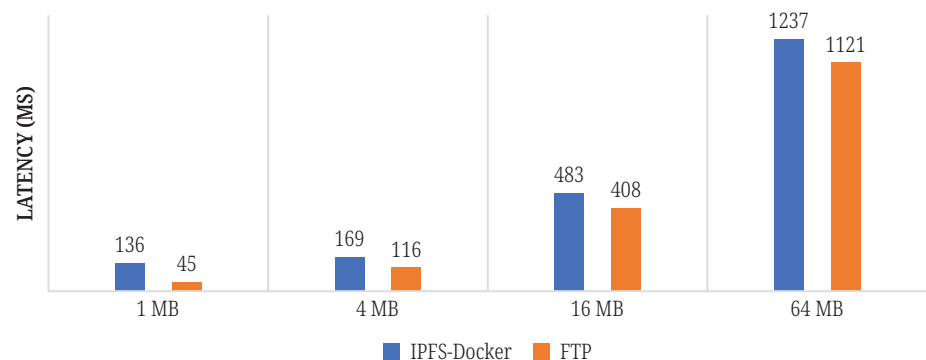


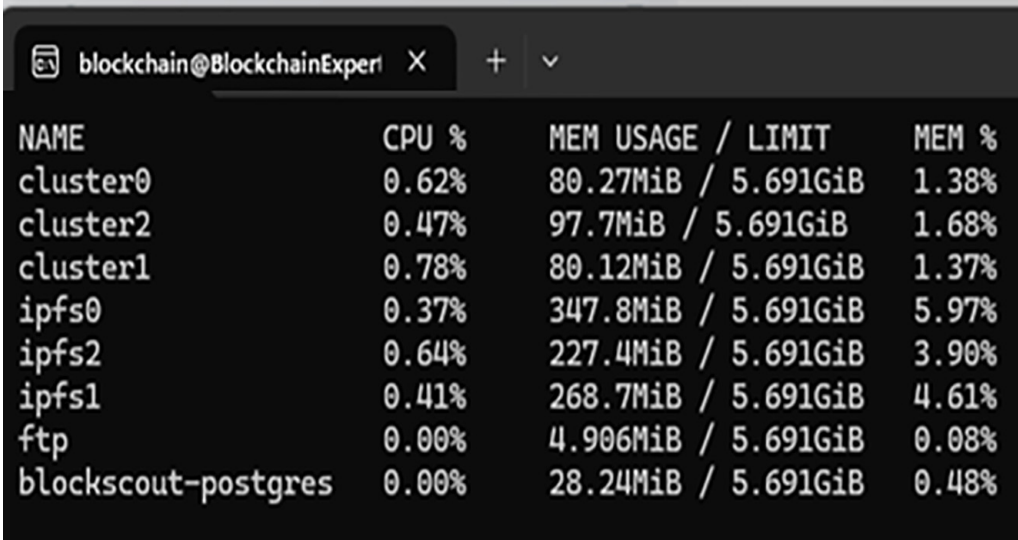
Fig. 5. Docker, cluster, and FTP reading operations latency for large-size files

Latency depends on several different parameters in a clustered, Docker-hosted IPFS setup. Although a broadcast mechanism like that in the traditional BitSwap protocol isn't necessary—the cluster takes care of the replication anyway—nodes create and send information to each other using BitSwap. The most crucial factor in latency is the cluster replication strategy. When a new file is added to the cluster, it will be replicated on several nodes, making it redundant and allowing for fault tolerance. The behavior of the cluster will reduce the risk of every node receiving the same data multiple times in a row, which can otherwise lead to higher latency [10].

There are also dedicated gateway ports and a cluster REST API, but they are an access point for data retrieval, independent of the cluster's number of nodes.

The efficiency of replication and data distribution in a cluster has a higher impact on latency than the number of nodes. For example, with replication managed by the cluster, the probability of receiving redundant data packets is much lower than without replication. Latency may improve because, dependent on the node, the redundant dataset may not be retrieved. Other factors are the overall load placed on the cluster, which can impact latency when the cluster must manage requests and perform replication, and the bandwidth between Docker containers and the host machine, which can influence latency due to network communication. IPFS Cluster improves data availability, redundancy, and fault tolerance by creating a distributed IPFS cluster architecture. Rather than a single IPFS node, the cluster has multiple coordinated IPFS nodes collaborating to replicate data, thus maintaining a distributed data replication system. When a new file is added to the IPFS Cluster, the cluster will coordinate the replication of the new file onto multiple nodes. This coordinated replication decreases the probability of nodes receiving redundant data packets. This is because the cluster controls the distribution of file chunks across the cluster instead of each node distributing data. The cluster's replication approach is a primary mechanism that prevents redundant data retrieval. With the cluster intelligently distributing file chunks to different nodes, the probability of nodes receiving redundant data when serving content requests is low. This differs from the traditional IPFS broadcast model, where more redundant data can be transferred between nodes. In addition to replication, a cluster REST API is another access point. This further reduces latency and has a higher impact on latency than the number of nodes [11].

Docker provides a natural way to deploy IPFS nodes, where developers can spin up IPFS nodes inside the containers with little effort. They can leverage the power of Docker on resource management. Overall, Docker simplifies how IPFS nodes are running while allowing more granular control of the resource allocation for these IPFS nodes. By setting hard and soft limits on the memory and CPU usage of IPFS containers, one can prevent IPFS nodes from consuming more resources than specified, as illustrated in Figure 6. This action prevents the IPFS nodes from degrading the host system performance or exhausting the resources [12].



NAME	CPU %	MEM USAGE / LIMIT	MEM %
cluster0	0.62%	80.27MiB / 5.691GiB	1.38%
cluster2	0.47%	97.7MiB / 5.691GiB	1.68%
cluster1	0.78%	80.12MiB / 5.691GiB	1.37%
ipfs0	0.37%	347.8MiB / 5.691GiB	5.97%
ipfs2	0.64%	227.4MiB / 5.691GiB	3.90%
ipfs1	0.41%	268.7MiB / 5.691GiB	4.61%
ftp	0.00%	4.906MiB / 5.691GiB	0.08%
blockscout-postgres	0.00%	28.24MiB / 5.691GiB	0.48%

Fig. 6. The Docker stats command returns a live data stream for running containers

This is particularly important when running multiple IPFS nodes, as this configuration allows more efficient resource utilization. Furthermore, IPFS Docker

can be hosted on Kubernetes, the container orchestration system. This will enable Kubernetes to dynamically allocate resources to the IPFS nodes based on runtime demand, automatically scaling up or down the node count to optimize the IPFS cluster for performance and resource efficiency. This dynamic allocation spreads the load across the available nodes so that a single node never becomes a bottleneck. The combination of IPFS, Docker, and Kubernetes enables scalable and robust deployment of IPFS-based containerized applications. A closer look at the resource utilization of containers running in an IPFS Cluster shows a more complex picture of CPU and memory utilization, reflecting the different roles and workloads of IPFS nodes and cluster containers. This analysis is based on our literature review and experimental studies, demonstrating that dynamic resource allocation mechanisms are crucial in a Docker cluster's efficiency and scalability [13].

The CPU utilization across all three IPFS nodes (ipfs0, ipfs1, ipfs2) remains low at 1% throughout. But a closer look at the memory utilization graph indicates that the ipfs0, ipfs1, and ipfs2 nodes have used a significant fraction of their available memory: ipfs0: around 6% of the available memory, which is 5.691 GB. ipfs1: approximately 4.7% of the available memory. ipfs2: almost 4% of the available memory. This high memory consumption results from the nodes managing data in data replication, block hashing, and block management. For the CPU utilization to remain so low despite the memory utilization being high points towards the likelihood that memory resources are a far more critical bottleneck on system performance than CPU capacity. Such patterns are characteristic of a distributed storage system such as IPFS, where each node has to manage metadata and store data blocks. The containers labeled cluster nodes (cluster0, cluster1, cluster2) also turn out to be surprisingly lightweight regarding resource usage. The low CPU and memory usage across the cluster nodes implies that the container running the IPFS Cluster peers is much more efficient than the other containers running the IPFS daemons. The containers acting as cluster peers are primarily there to maintain the metadata and communicate with each other about the various tasks related to the IPFS Cluster, such as coordination, managing the replication strategy, and maintaining the cluster's health. This segregated pattern of resource utilization indicates how well the IPFS Cluster architecture distributes the computational load across its nodes based on the functional requirements of each peer. Docker containers also have the convenience of changing resource allocations dynamically. For example, CPU, memory (RAM), and storage needs can be changed on the fly and adjusted based on what is needed. This is especially critical in allocating resources optimally. If resources can be automatically scaled up or down when required, the IPFS Cluster can run at peak performance and avoid wasting resources on unnecessary things. Using automation, container orchestration tools such as Kubernetes and Docker Swarm can be leveraged to dynamically scale IPFS Clusters to adapt to workload fluctuations and operational demands. This is especially useful for IPFS clusters because the number of IPFS nodes and cluster containers can change over time. The cluster should be provisioned with computational resources to execute their tasks optimally. Using resource limits and quotas and properly deploying containers with resource profiles in the cluster can avoid potential bottlenecks and improve cluster scalability. This distribution optimizes the utilization of underlying hardware resources. It contributes to the stability and reliability of the IPFS Cluster by preventing resource contention and ensuring fair resource utilization among the containers. In summary, by looking at the resource utilization patterns in the IPFS Clusters, we can see a sophisticated allocation of computational resources mirroring the different operational demands of IPFS nodes and cluster containers. The deployment of dynamic

resource allocation mechanisms further enhances the efficiency and scalability of Docker-Cluster environments, making adaptive resource management an important consideration for maintaining the performance and reliability of distributed storage systems [14]. Meanwhile, setting proper memory limits for IP's resource management features can help to mitigate the performance issues. Improving the cluster's replication strategy and optimizing the data distribution can help alleviate the memory burden on individual nodes. Dynamic allocation of resources in the cluster can be achieved through container orchestration platforms such as Kubernetes. Another way to assign GPU resources to IPFS nodes is through a container orchestration platform such as Kubernetes.

6 RESULTS AND DISCUSSION

The latency measurements of writing the files to the FTP container and the local IPFS repository container are shown in Figures 7 and 8, with the name `ips0` and default port 5001, respectively, for the cluster implementation of IPFS. These figures depict the latency of IPFS writing operations for small and large files and compare it with the results of Lajam and Helmi [8], where the writing operations to IPFS with the current Docker implementation are much less latent than the latter.

In addition, the two implementations had significant differences in latency because the IPFS performance in the virtual machine (VM) and Docker container environments depends on several key factors. In the case of VMs, the resource allocation is done at the virtual machine level, which means that the IPFS instance is guaranteed a certain level of performance. However, the overhead involved in VMs is higher than the other two containers since VM technology employs another layer of abstraction. This virtual operating system might impact the startup time and overall efficiency.

In contrast, Docker containers share the host system's kernel, making Docker containers much lighter weight (than VMs) and, therefore, faster to start. Still, the resource allocation is more dynamic depending on the host's overall load. Storage and network performance can be an additional factor here because Docker containers can benefit from the host's direct access to the physical storage and network resources. The specific IPFS workload (data size and access) can also significantly impact the performance of both the VM and the Docker environment. Docker containers' startup time, isolation, and portability are generally more accessible than VMs and the bare metal environment. However, the host resource configuration is still the most critical factor determining IPFS deployment performance [15].

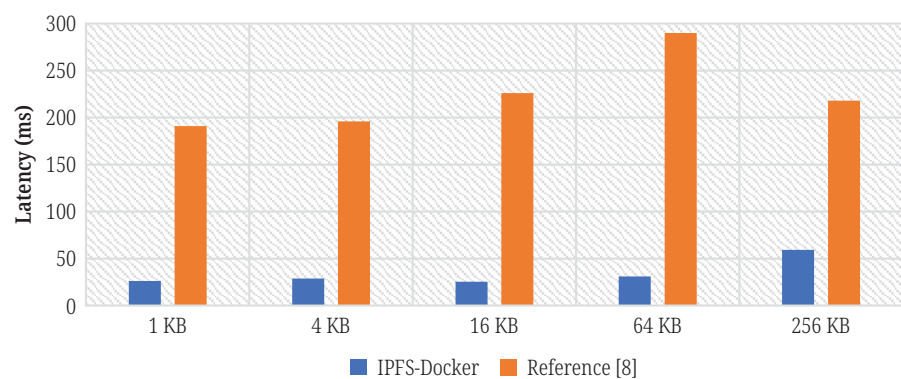


Fig. 7. Latency of IPFS writing operations of small-size files in the Docker cluster implementation

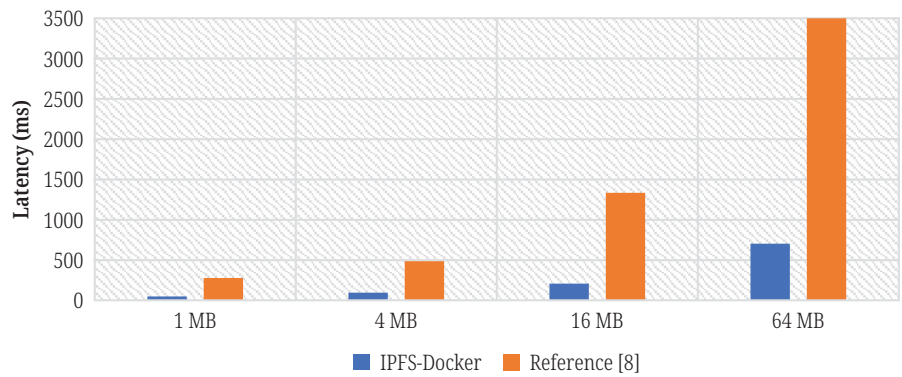


Fig. 8. IPFS Latency for writing large file operations in the Docker cluster implementation

In contrast, latency measurements for reading the files from the FTP and the local IPFS repository containers are measured through the gateway ports of ipfs1 (port: 5002) and the Cluster REST API (port: 9094), respectively. It is noted that whenever a new file is added to one of the IPFS Cluster nodes, the cluster replicates this content across the three IPFS nodes. The latency of the reading operations is shown in Figures 9 and 10, respectively. The degree of difference in the latencies between IPFS-Docker in the two cases was considerably high for both small and large files. However, IPFS operations latencies in the Docker environment were reduced.

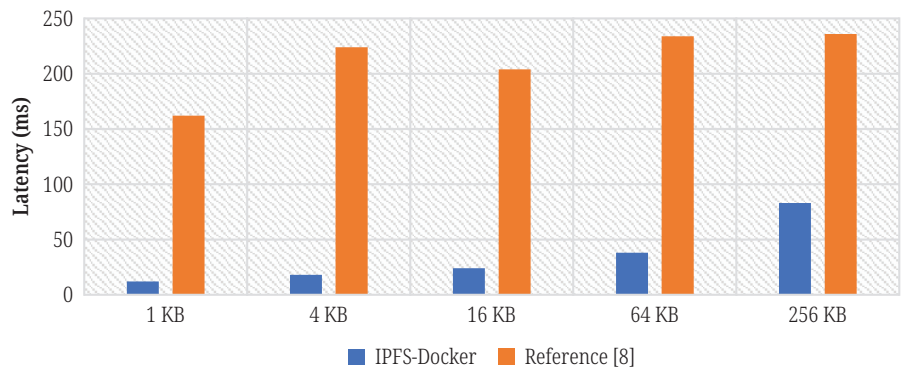


Fig. 9. The latency for IPFS reading operations of small-size files in the Docker implementation

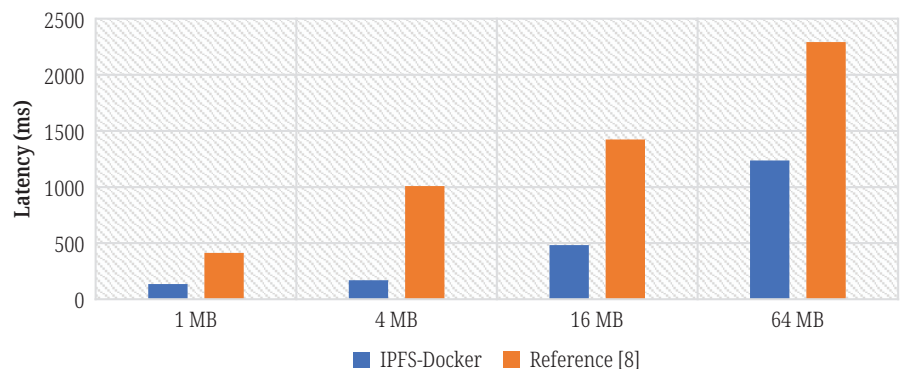


Fig. 10. The latency for IPFS reading operations of large-size files in the Docker implementation

Many factors can influence latency. With a cluster of IPFS in Docker where BitSwap broadcast is irrelevant (because the cluster manages the replication), the cluster still processes data exchanges with other nodes via BitSwap. However, how

the cluster replicates the data will most likely affect latency. Again, when you add a new file, the cluster will replicate it across many nodes, which is where BitSwap is used. Because the cluster coordinates the replication, the likelihood of receiving redundant data is much smaller than if the nodes replicated it independently, which could improve latency.

Another factor is that there are dedicated gateway ports and a Cluster REST API for data retrieval. How this would affect latency depends less on how many nodes are in the cluster and more on how well the cluster replicates and distributes the data so that nodes receive the data without redundant data packets. Because the cluster is handling the replication, the likelihood of receiving redundant data is much less than if the nodes shared it on their own, which would improve latency. Other factors include the overall load the cluster is under, which can affect latency, and the bandwidth between the Docker containers and the host, depending on the network communication between the nodes and the host. The cluster has many nodes that can influence latency, but because it manages the replication, its influence on latency is moderated by how it distributes and shares the data, which could improve latency.

7 CONCLUSION

This paper presents a novel implementation of Dockerized IPFS, demonstrating significant performance improvements in off-chain storage for Blockchain of Things (BCoT) systems. The experimental results reveal that the IPFS-Docker implementation consistently outperforms traditional IPFS systems in both read and write operations across various file sizes. Notably, the implementation achieves lower latency, with the performance gap widening as file sizes increase. For smaller files (1–256 KB), the IPFS-Docker system shows substantial benefits, particularly in writing latency, where it achieves approximately 75% lower latency than the reference system. This trend continues for larger files (1–64 MB), with a remarkable threefold reduction in latency for 64 MB file operations. Specifically, the read latency for 64 MB files is improved by 40%, while write latency is reduced to around 1000 ms—significantly faster than the over 3000 ms observed in the reference system. These findings underscore the potential of Dockerized IPFS to enhance file transfer efficiency within BCoT frameworks, where rapid data access and transfer are critical for real-time applications and decision-making processes. By optimizing off-chain storage solutions, this implementation can facilitate better data management and interoperability among connected devices in a BCoT ecosystem.

Looking ahead, future work will focus on exploring a broader range of file and block sizes, conducting repeated operations to ensure reliability, simulating more realistic virtual environments with increased node counts, and examining how limited resources impact IPFS performance. This study aims to refine further distributed storage solutions tailored to the unique demands of BCoT systems, ultimately enhancing their scalability and efficiency.

8 REFERENCES

- [1] Q. Zheng, Y. Li, P. Chen, and X. Dong, “An innovative IPFS-based storage model for blockchain,” in *2018 IEEE/WIC/ACM International Conference on Web Intelligence (WI)*, 2018, pp. 704–708. <https://doi.org/10.1109/WI.2018.000-8>

- [2] H.-S. Huang, T.-S. Chang, and J.-Y. Wu, "A secure file sharing system based on IPFS and Blockchain," in *Proceedings of the 2020 2nd International Electronics Communication Conference*, 2020, pp. 96–100. <https://doi.org/10.1145/3409934.3409948>
- [3] G. Pandey, G. Sahu, and M. Singh, "Improving data integrity of IPFS on-chain proof," in *2023 6th International Conference on Contemporary Computing and Informatics (IC3I)*, 2023, pp. 171–177. <https://doi.org/10.1109/IC3I59117.2023.10398081>
- [4] Q. Xu, Z. Song, R. S. Mong Goh, and Y. Li, "Building an ethereum and IPFS-based decentralized social network system," in *2018 IEEE 24th International Conference on Parallel and Distributed Systems (ICPADS)*, 2018, pp. 1–6. <https://doi.org/10.1109/PADSW.2018.8645058>
- [5] S. Bhadula, S. Sharma, and A. Johri, "Hybrid blockchain and IPFS for secure industry 4.0 framework of IoT-based skin monitoring system," in *2023 9th International Conference on Advanced Computing and Communication Systems (ICACCS)*, 2023, pp. 41–47. <https://doi.org/10.1109/ICACCS57279.2023.10112751>
- [6] J. Shen, Y. Li, Y. Zhou, and X. Wang, "Understanding I/O performance of IPFS storage: A client's perspective," in *2019 IEEE/ACM 27th International Symposium on Quality of Service (IWQoS)*, 2019, no. 17, pp. 1–10. <https://doi.org/10.1145/3326285.3329052>
- [7] A. Von Tottleben, C. Ihle, M. Schubotz, and B. Gipp, "Academic storage cluster," in *2021 ACM/IEEE Joint Conference on Digital Libraries (JCDL)*, 2021, pp. 278–279. <https://doi.org/10.1109/JCDL52503.2021.00034>
- [8] O. Abdullah Lajam and T. Ahmed Helmy, "Performance evaluation of IPFS in private networks," in *2021 4th International Conference on Data Storage and Data Engineering [Preprint]*, 2021, pp. 77–84. <https://doi.org/10.1145/3456146.3456159>
- [9] A. P. Ahmad, A. A. Ilham, and A. W. Paundu, "Analysis of blockchain and interplanetary file system (IPFS) utilization for big data architecture optimization," in *2023 IEEE International Conference on Communication, Networks and Satellite (COMNETSAT)*, 2023, pp. 652–657. <https://doi.org/10.1109/COMNETSAT59769.2023.10420785>
- [10] H. Shin, M. Lee, and S. Kim, "Space and cost-efficient reed-solomon code based distributed storage mechanism for IPFS," in *2023 14th International Conference on Information and Communication Technology Convergence (ICTC)*, 2023, pp. 1165–1169. <https://doi.org/10.1109/ICTC58733.2023.10392473>
- [11] W. Kim, A. Kwak, B. Yoo, and H. Ko, "IPFS Viewer: IoT surveillance camera system using IPFS and MQTT," in *2024 IEEE International Conference on Consumer Electronics (ICCE)*, 2024, pp. 1–6. <https://doi.org/10.1109/ICCE59016.2024.10444244>
- [12] K. Sivasankari and V. S. Sathyamithran, "IPFS enabled robust mechanism for file storage and retrieval using block chain," in *2022 Fourth International Conference on Emerging Research in Electronics, Computer Science and Technology (ICERECT)*, 2022, pp. 1–5. <https://doi.org/10.1109/ICERECT56837.2022.10059644>
- [13] J. Tang, T. Jia, H. Chen, and C. Wei, "Research on big data storage methods based on IPFS and blockchain", in *Proceeding of the 2020 2nd International Conference on Video, Signal and Image Processing (VSIP '20)*, NewYork, NY, USA: Association for Computer Machinery, 2021, pp. 55–60. <https://doi.org/10.1145/3442705.3442714>
- [14] P. Kumar, M. Gupta, and R. Kumar, "Improved cloud storage system using IPFS for decentralised data storage," in *2023 International Conference on Data Science and Network Security (ICDSNS)*, 2023, pp. 1–6. <https://doi.org/10.1109/ICDSNS58469.2023.10245317>
- [15] S. Routray and R. Ganiga, "Secure storage of Electronic Medical Records (EMR) on interplanetary file system (IPFS) using cloud storage and blockchain ecosystem," in *2021 Fourth International Conference on Electrical, Computer and Communication Technologies (ICECCT)*, 2021, pp. 1–9. <https://doi.org/10.1109/ICECCT52121.2021.9616690>

9 AUTHORS

Mamoon Aldmour is with the School of Digital, Technology, Innovation, and Business. The University of Staffordshire, Stoke-on-Trent, UK.

Rakan Aldmour is with the School of Digital, Technology, Innovation, and Business. The University of Staffordshire, Stoke-on-Trent, UK.

A. Y. Al-Zoubi is with the Princess Sumaya University for Technology, Amman, Jordan (E-mail: zoubi@psut.edu.jo).

Mohamed Sedky is with the School of Digital, Technology, Innovation, and Business. The University of Staffordshire, Stoke-on-Trent, UK.