Journal of Systems Architecture 125 (2022) 102441

Contents lists available at ScienceDirect

Journal of Systems Architecture

journal homepage: www.elsevier.com/locate/sysarc

# DSPR: Secure decentralized storage with proof-of-replication for edge devices

Chenggang Wu *, Yongbiao Chen, Zhengwei Qi, Haibing Guan

*Shanghai Jiao Tong University, China*

ARTICLE INFO

*Keywords:*
Distributed storage
Edge computing
Internet of things

ABSTRACT

The evolving edge and IoT devices collect and process massive data and need tremendous storage space for those data. It is more efficient and reliable to establish a distributed storage system for their storage need. However, the conventional erasure codes used in distributed storage are designed for reliable data centers. Edge and IoT devices are highly vulnerable and unreliable. The conventional distributed storage systems cannot prevent malicious attackers and require very high redundancy in wild environments. This paper presents DSPR, a new decentralized distributed storage system with proof of replication for edge and IoT devices. By providing the cryptographic proofs of stored data, the system can actively find the data corruption and recover the data before it is unrecoverable in vulnerable environments. Therefore, we can apply high efficient coding schemes with low storage redundancy. Our LDPC coding achieves about 6x encoding and 20x recovering speedup than Intel ISA-L. With an IoT device Xavier AGX, the system can prove over 180GB of data per day to securely store the data, which is about 4x faster than public auditing schemes with an even stronger threat model that every device in the network can be malicious.

## 1. Introduction

Edge and IoT devices have been rapidly evolved in recent years. Many IoT devices have been deployed commercially and researched for years [1]. These devices are generating and collecting enormous data every day. To accommodate the need of processing these data, the computing power of these devices has been enormously strengthened. Nowadays, even small IoT devices can have a powerful GPU like Nvidia Xavier AGX [2]. The edge devices move the data closer to the users to reduce latency and bandwidth usage. The IoT devices collect big data with their sensors. The cameras record and identify the people to improve security. These devices need a tremendous amount of storage, but deploying them with massive storage space is inefficient since the data locality is diverse. It is more efficient and reliable to establish a distributed storage system. There are several works aim at establishing multi-layer distributed model for these devices [3].

Erasure coding is a commonly used mechanism for distributed storage systems in data centers. High efficient erasure coding can reduce disk usage while providing high redundancy against disk or server failures. Different erasure codes like Reed–Solomon (RS) code [4–7] and LRC [8] code have been widely deployed in data centers. The erasure code works for configurable number $m$ data chunks, encode them with another configurable number $k$ parity check data chunks to recover the data when failure happens. For the most commonly used RS code, the $(n, k)$ RS code stores $k$ data chunks with $n$ chunks in total. It can tolerate up to $n-k$ storage node failure with the redundancy of $\frac{n}{k}$. Many other new erasure codes aim for better performance and lower storage redundancy without sacrificing reliability in recent years [9–11]. There are also erasure codes designed for mobile devices these years [12].

However, the conventional erasure coding schemes in distributed storage are now only used in data centers. Two reasons prevent distributed storage systems from being used in commercial products. The erasure coding is used for reconstructing data as a backup during recovering the offline storage nodes [8]. Despite the high reliability of data centers, the erasure codings still need to tolerate arbitrary multiple node failures. On the contrary, the edge and IoT devices are much more unreliable due to cost efficiency. The edge and IoT devices have limited lifetimes designed for different environments, and the reliability of the devices is heavily affected by many factors like device diversity, device placement, and courier behaviors [1]. The nodes can fail anytime, but no centralized monitoring system monitors the failure nodes like the data centers. Moreover, the edge and IoT devices usually

The code (and data) in this article has been certified as Reproducible by Code Ocean: (https://codeocean.com/). More information on the Reproducibility Badge Initiative is available at https://www.elsevier.com/physical-sciences-and-engineering/computer-science/journals.

* Corresponding author.
*E-mail address:* wuchenggang@sjtu.edu.cn (C. Wu).

https://doi.org/10.1016/j.sysarc.2022.102441
Received 31 August 2021; Received in revised form 11 February 2022; Accepted 15 February 2022
Available online 26 February 2022

lack redundancy features like RAID and ECC memory, so they cannot predict or prevent its failure and data corruption like the data centers.

On the other hand, the edge devices are deployed near the users and are not physically well protected. The devices are highly vulnerable to physical attacks like classic cold boot attack [13] and DMA attack [14,15]. Erasure coding cannot detect whether the data have been maliciously constructed. The malicious data may be recovered. Therefore, the conventional erasure coding and distributed storage are hard to be applied in edge and IoT environments. Conventional erasure codes need very high redundancy and low efficient coding configuration to achieve reliability in the edge environment.

Proof of storage, proof of replication, and some other variants are several non-interactive cryptography proof systems proposed these years. They can prove that some data $D$ is physically stored on some device [16,17]. If the prover $\mathcal{P}$ has a physical copy of data $D$, $\mathcal{P}$ can generate some proof $\pi$ with the data $D$. Then $\mathcal{P}$ sends the $\pi$ to the verifiers. Any verifier $\mathcal{V}$ that receives the proof $\pi$ can verify whether the proof is valid or not. Then it knows whether $\mathcal{P}$ has the physical copy $D$.

We can combine the proof-of-replication (PoRep) and the erasure coding scheme with the proof system. We adapt a PoRep system to ensure data integrity and reliability on edge and IoT devices. The increasing computing power of the edge devices is now capable of generating the proofs. By generating and verifying the data on each storage node, the system can actively find the corrupted or forged data and recover them. Since the PoRep system provides a robust data integrity detection scheme, we can use some erasure coding schemes with much lower storage overhead, like the Low-Density Parity-Check (LDPC) code we used in our system. These codes have very high efficiency and performance but cannot tolerate arbitrary multiple (over 3) node failures. Our PoRep system can actively detect and recover failed nodes and prevent the failure of 3 nodes simultaneously.

Although the ideas are straightforward, there are several challenges in achieving this system. Firstly, non-interactive proofs, including PoRep, require some "challenges" included in the proofs. The verifiers verify the proof according to the challenges. We need to construct a protocol to let every node have the same challenges at the same time to prevent attackers from sending old proofs without storing the data. And the challenges cannot be foreseen to prevent attackers from generating proofs ahead of time. Secondly, cryptographic operations are extremely computation intensive. Low power devices, especially IoT devices, are still struggling to provide enough computation power. The conventional programs are also having performance issues on IoT platforms. With the original PoRep program, the AGX can merely prove 11 GB data in one day period, which is far from usable. We optimized the generation of PoRep, especially on IoT devices, to let low-power devices have enough capability in our system. On the other hand, we do not want the PoRep computation to affect other running applications. We want our system to fully utilize the idle computing power but not affect the other applications. The detailed methods to overcome these challenges are described in 4.2 and 6.

There are similar systems that provide the public auditing schemes with cryptography like PPPA (Privacy-Preserving Public Auditing) [18,19] and some variants like Radds [20]. However, these works still cannot provide practically usable systems. Moreover, the verification or repairing process costs seconds each time. The interactive proving scheme will also cause enormous network traffic and round-trip overhead. More will be discussed in Section 2 and evaluated in 7.

This paper presents a new distributed storage system designed for edge and IoT devices with proof of replication. With the proof of replication system, the devices can verify the stored data of each other without the need to read the data. The failed data chunks or malicious data can be actively detected to warn and recover the data before the data is unrecoverable or malicious modified. Therefore we can apply high efficient coding schemes with low storage redundancy in wild environments. Our contributions include:

- We design and implement DSPR, the first practical storage system with a public auditing scheme. It realizes high efficient erasure coding storage system with a proof-of-replication (PoRep) system. The data nodes send the proof of replication to the other nodes periodically. In this way, the nodes in the system actively verify each other's data integrity and repair the failed or corrupted data. A one-day proving period can provide higher reliability than enterprise-level storage systems with unreliable edge and IoT devices.
- The failure and corruption detecting scheme relaxes the requirement of erasure codes and allows the use of high efficient erasure coding schemes. The LDPC code we used in our system achieves about 6x encoding and 20x recovering speedup than Intel ISA-L's RS code.
- A scheduler provides high efficient proving system. An edge server with 4 CPU threads and a GPU can prove over 400 GB of data per day. Small IoT devices like Xavier AGX can prove over 180 GB per day. The resource-aware congestion control unit manages the hardware resources and utilizes regularly underutilized CPUs. Therefore, the proving system can run along with the other applications without sacrificing the performance of other applications.
- DSPR is a secure decentralized system. It does not require a centralized monitoring system to provide data availability and reliability. The proof system of each storage node audits each other's data availability and integrity. We implemented the system with 100% Rust to ensure language level high reliability and security. By storing the encrypted data, DSPR allows any third-party device and anyone to join the storage system with the p2p protocol and share their spare disk space reliably and securely.

## 2. Related works

**Avocado** [21] is a similar work for providing storage on untrusted cloud environments published recently. They implemented and optimized an in-memory KVS system secured by TEE, specifically Intel SGX. They use BFT as a fault-tolerant protocol. They implemented a memory management system to access the encrypted data with pointers in TEE since the limited enclave space cannot store much data. Compared to our system, Avocado cannot provide large volumes of conventional block storage. In-memory KVS is inapplicable in edge and IoT devices due to the very limited RAM size. Also, their system is limited to the Intel SGX platform, which is not the majority of the edge and IoT market. Another drawback is that the system needs.

**IPFS**: The InterPlanetary File System (IPFS) is a famous P2P file system developed and widely used in recent years [22,23]. IPFS generates a unique fingerprint of the blocks of a file. This fingerprint is used to look up the file content and remove duplications across the network. The node in the network stores only the indexing information and the content it is interested in. Therefore, only files interested by enough network nodes can be stored on the network. Otherwise, the file would be lost when the original node is down or corrupted. The reliability of the data only relies on that many reliable servers access the file and store the duplications. However, it is impractical in edge computing. The edge devices are not reliable enough. The edge data has strong localities and may not be interested in other edge devices. Moreover, the reliability provided by data duplication is low efficient.

**Nebula** [24] provided a distributed edge system for data-intensive computing. Nebula's reliability solution is adapting the reliability-based technique [25]. Their method is the conventional method, like scoring reliability scores per node and scheduling the task to multiple nodes for redundancy. Nebula's data availability is also provided by data duplications and availability scores per node. Our system also distributes the data blocks based on the reliability of the node. Moreover, simply redundancy is hard to prevent compromised nodes when malicious

users or attackers corrupt the data or the computation results. The data availability in DSPR is provided by high efficient erasure coding.

**PPPA (Privacy-Preserving Public Auditing)** [18] and many other similar works [19,20,26] provided similar attempts to store data securely on edge or untrusted with cryptographic calculation. However, they did not provide any practically usable system. The auditing speed is 6.5x slower than our scheme on a powerful server and 3.7x slower on the low-power IoT device Xavier AGX. The verification and repairing speeds are extremely slow at 0.25 MB/s on the server and 0.08 MB/s on the AGX. It is unacceptable to read files with this verification overhead. The benchmark results will be discussed in evaluations. Their schemes need to store encoded signatures, introducing much higher storage overheads (2x) than modern erasure codes. The proofs provided by storage nodes are not NIZK proofs like in our system, i.e., every verifier needs to interact with the storage server with their own keys and challenges to verify the proofs. This will cause enormous network overhead and round trip time. Furthermore, their threat model needs a semi-trusted Proxy server, which is weaker than our threat model. Moreover, one PPPA variant [19] and its variants have been proved to have security flaws in 2020 [27].

## 3. Background

### 3.1. Proof of replication

*Proof-of-Storage* works as a prover $\mathcal{P}$ , i.e. the data storage node in our system, attempts to convince a verifier $\mathcal{V}$ that $\mathcal{P}$ is storing some data $D$. $\mathcal{V}$ issues a challenge $c$ to $\mathcal{P}$ , and $\mathcal{P}$ responds with a corresponding proof $\pi$.

This scheme and the variants are NIZK proof systems (Non-interactive zero-knowledge proof) [28]. A NIZK proof system has the following properties:

- *Completeness*: The $\mathcal{P}$ can convince $\mathcal{V}$ if the statement is true, i.e., the data is stored correctly.
- *Soundness*: $\mathcal{V}$ cannot be convinced by a malicious $\mathcal{P}$ if the statement is false, i.e., the data is corrupted.
- *Zero-knowledge*: $\mathcal{V}$ cannot get any information other than that the statement is true or false, i.e., the data is stored correctly.
- *Non-interactive*: By publishing the proof $\pi$, any $\mathcal{V}$ gets the $\pi$ can verify the statement without communication with the $\mathcal{P}$ .

*Proof-of-Replication* (PoRep) schemes are introduced to prove the data $D$ is physically stored on some device based on *Proof-of-Storage*. The prover $\mathcal{P}$ generates a proof $\pi$ based on the data. The verifier $\mathcal{V}$ can verify that the $D$ has been replicated to its dedicated physical storage.

There are several scenarios and attacks of malicious users that PoRep can prevent [16]:

- *Sybil Attack*. An attacker $\mathcal{A}$ has $n$ sybil identities $\{\mathcal{P}_n\}$, each commits to store a replica of $D$. However, $\mathcal{A}$ only stores less than $n$ replica (e.g., one replica) and tries to convince verifier $\mathcal{V}$ that it stores $n$ physical replicas.
- *Outsourcing Attack*. When $\mathcal{A}$ is challenged by some $\mathcal{V}$ , $\mathcal{A}$ fetches the $D$ from other storage providers $\mathcal{P}'$ and generates the proof without physically storing the $D$ itself.
- *Generation Attack*. $\mathcal{A}$ chooses some $D$ that $\mathcal{A}$ can regenerate $D$ on demand. Upon receiving challenge $c$, $\mathcal{A}$ regenerates the $D$ and the proof without storing the original data $D$.

The detailed proof and design of the scheme to prevent these attacks can be seen in [16,17].

In our system, the data storage node generates the PoRep in a configurable period and broadcast the proof to a number of other nodes like Fig. 1 shown. With the PoRep scheme, the data stored in failed or malicious nodes can be detected in one period. Only the proof $\pi$ needs to be sent on the network periodically. The proof size is small, which is only 256 Bytes for each storage sector in our system. The storage sector size is configurable. Proof sizes are the same for any sector size.

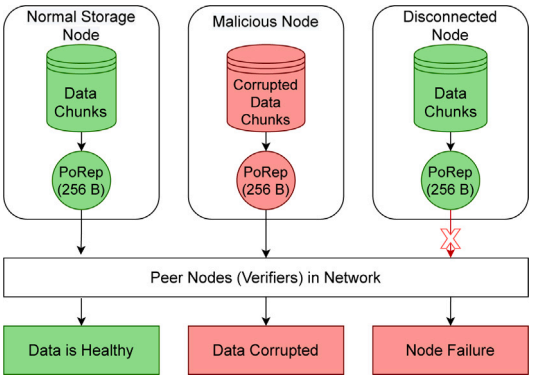



**Fig. 1.** PoRep in DSPR.

### 3.2. Erasure coding for distributed storage

Data is the most important and valuable asset in computer systems today. Data centers and HPCs are using distributed storage systems to improve performance and reliability [8,29]. Information dispersal algorithms (IDA) based methods, like erasure coding, outperform data replications in both HPC and cloud computing [30]. Therefore, in today's data centers, erasure codes are used to provide data redundancy to ensure reliability. There are also many works that have been done to accelerate the encoding and decoding of the erasure codes by using GPGPU accelerators [31] and optimizing the CPU's SIMD units [32,33].

A common distributed storage organizes and stores data in fixed-size chunks across a number of storage nodes. The erasure coding encodes the data chunks by adding parity check data chunks. The most commonly used erasure codings now are RS code and LRC code. Take RS code as an example: an (n,k) RS code encodes $k$ fixed-size data chunks into $n-k$ parity check chunks of the same size. This configuration can tolerate arbitrary $n-k$ node failures.

Different application scenarios often use different chunk sizes and different coding configurations for performance, reliability, and storage cost considerations. For example, Facebook's Tectonic system uses 100 s of MBs chunk size and RS(9,6) for cold storage and uses kB level chunk size and RS(10,4) for hot storage [29].

In history, many other erasure coding schemes with higher efficiency like LDPC have been proposed [34,35]. However, these codes are not MDS (maximum distance separable), i.e., they can recover $m$ specific failure nodes but cannot recover arbitrary $m$ failure nodes. It makes them harder to be applied in data centers [8], and even impossible in more vulnerable edge and IoT scenarios. This will no longer be a problem in our system. The failed nodes will be recovered within one proof period to prevent the arbitrary $m$ node failure problem. It makes using a high-efficiency erasure coding scheme becomes possible, like the LDPC code.

### 3.3. LDPC code

Low-Density Parity-Check (LDPC) code is originally a linear error-correcting code used in message transmission in noisy transmission channels. It is famous for transmitting messages through a channel with the noise threshold very close to the theoretical maximum (the Shannon limit). There are also several attempts to apply LDPC in distributed storage's erasure coding years before [34,35]. However, no one is applied in the real world. It has worse trade-offs than the other modern erasure codings: the distributed storage systems in data centers typically require the erasure coding scheme to tolerate arbitrary three or more node failure to prevent multiple storage node failures. LDPC needs high redundancy to achieve that. However, the idea of the LDPC code is taken by some famous erasure codings like the LRC code applied in Azure [8].

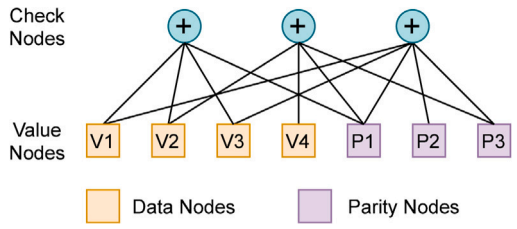



**Fig. 2.** A simplified idea of LDPC coding algorithm.

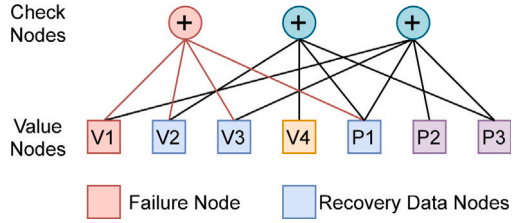



**Fig. 3.** Recover with only part of nodes when V1 failed.

The idea of LDPC coding is straightforward. It consists of a tanner graph with value nodes and parity check nodes like Fig. 2 shows. The parity nodes' values are constructed so that the xor result of all value nodes connected to the same check node is 0.

There are two kinds of algorithms to construct and decode the parity nodes. The straightforward one is selecting the nodes and xor them with 0 to get the value of the remaining node. This approach is called the hard decision algorithm. The soft decision algorithm that makes LDPC famous is only applicable in analog noise channels like Gaussian noise channels, which do not fit the erasure coding model in common distributed storage.

Despite the lack of soft decision algorithm benefits, there are still various benefits brought by the LDPC code. The locality of LDPC code allows that only the data on the other nodes connected to the same check node need to be fetched when one node fails. For the example situation in Fig. 3, when V1 failed, only the data on V2, V3, P1 or V3, P1, P2, P3 need to be fetched to recover the data on V1. This locality inspired the Azure LRC code and other local recoverable erasure codes. The pure XOR-based code makes the encoding and decoding latency very low. Furthermore, XOR operations can still have excellent performance without SIMD instructions like AVX. It can avoid lowering CPU performance on other programs due to the CPU's AVX frequency offset [36].

In this paper, the active integrity detection property of our system changes the game of storage-reliability trade-off. The multi-node failure is detected and recovered actively by the proof-of-replication scheme in our system. Therefore, the LDPC code can be applied to achieve higher efficiency in distributed storage.

## 4. Architecture

The architecture of the DSPR system is shown as Fig. 4. The storage nodes all have the same functionalities and consist of two main modules: the storage module and the PoRep module. The nodes are automatically discovered and connected by P2P protocol. The storage module is responsible for erasure coding and data distribution, just like a regular distributed system. The PoRep module ensures data integrity in the distributed system by applying the proof of replication. The system is 100% implemented with Rust to provide language level high reliability and security.

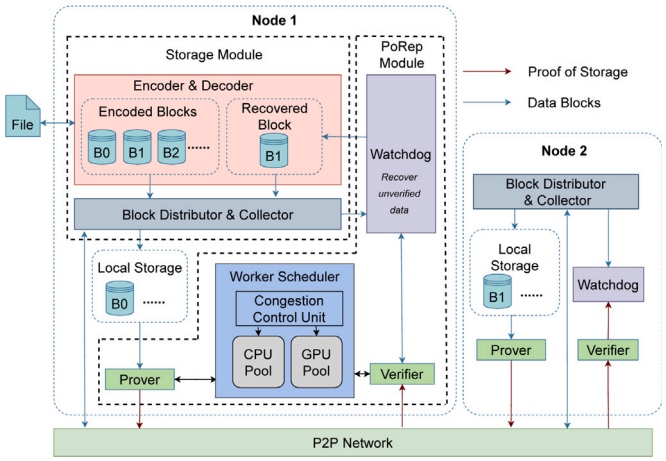



**Fig. 4.** The architecture of DSPR. The construction of node 2 is simplified.

### 4.1. Storage module

For the storage module, just like regular distributed storage systems, the input file is first encoded by chosen erasure coding scheme to encoded blocks $\{B_n\}$. Then the block distributor will decide where these blocks should be stored. Some encoded blocks will be stored locally. According to the peer nodes' remaining space, locality, and history reliability, other blocks will be distributed to several other storage nodes through the P2P network. After the data blocks are distributed and stored, the metadata will be broadcast to the network.

On the other hand, if replica-based redundancy is still needed for performance or other needs, the encoding system can specify the number of replicas of each data block to be stored. These replicas are still stored like they are different data blocks. Therefore, the number of replicas is ensured.

### 4.2. PoRep module

In recent years, many techniques use blockchains to provide security in data sharing [37,38]. We took the key idea of public consensus from the blockchains. To ensure the strong integrity of the stored data, our system introduced a PoRep module. The PoRep module periodically generates and broadcasts the proof-of-replication (PoRep) of the data stored on the node and verifies the PoRep received from other nodes. The proof-of-replication, just as described in Section 3.1, is the proof of physically stored data. If the PoRep is successfully verified, it means the corresponding data block is solidly stored by the storage node.

For a configurable period $T$, when one period starts, all nodes in the network generate a consensus seed $S$, also called the challenge, from the hash of all proofs received from the previous period. This seed will be used in PoRep generation and verification. If the nodes are functional and honest, they will have the same consensus seed since the proofs in the previous period should be the same. The seed $S$ cannot be forged by malicious nodes since the seed will be verified in the verification. The consensus seed also cannot be foreseen before a period starts. A node should get proofs from all the nodes to generate the seed. The proofs could be sent at any time in one period depending on the capability of each storage device. The malicious nodes do not know when all the proofs will be generated. Therefore, the malicious nodes cannot get the consensus seed before a period starts. This consensus protocol is Byzantine fault tolerance.

On the other hand, since the nodes are discovered and transmitted through P2P network, a malicious node cannot just send its proofs to selected nodes to break the consensus. If a proof is sent to an honest node, the honest node will send the proof to other nodes. The honest

nodes will achieve consensus sooner or later. If the proofs are only sent to malicious nodes, the proofs are still regarded as failed proofs and recovered by honest nodes.

After having the consensus seed, the node generates the proofs of replication $\pi_n = P(B_n, S)$ by the node prover. After generating a proof of replication $\pi$, the proof is broadcasted to the network. The prover can also detect data corruption and recover the data by checking the hash of the data itself during the proving phases. The proof can also be sent to only the local P2P network or some sufficient peer nodes to avoid flooding the proof to the whole network.

When the node receives a proof $\pi$, the verifier verifies the proof with $V(\pi_n, S)$ and tells the watchdog whether the corresponding data block $B_n$ is still valid. After a period ends, if the watchdog finds that some node cannot provide valid proofs or fails to provide the proofs, the data blocks of this node are marked as corrupted. The chosen erasure code will recover the data blocks on this node if this storage node has available storage space and is not storing other data blocks of the same file. After recovering, the proofs of the recovered blocks will be generated and broadcasted so that other storage nodes can verify that the data block is securely stored by another node and modify the metadata that the new node becomes the new owner of the data block. If more than one node recovered this data block, the first finished proving and broadcasted the proofs will become the new owner. The other nodes can either discard the block or keep the copy as redundancy. The data block can be stored by multiple storage nodes to enhance stability.

Since the nodes are auto-discovered by mDNS and interconnected by the P2P network, anyone can join the storage network with their device running this system. We can fully utilize the surplus storage of different devices and ensure reliability and security with this system.

However, the generation of PoRep requires complicated cryptographic computation and consists of several computing phases with different hardware resources needs. It is highly inefficient to compute the PoReps without scheduling the phases. This will be discussed in detail in Section 6. Also, the PoRep is only required to be generated within one period, which may not be as urgent as some other jobs on the storage node. On the other hand, the underutilization of hardware resources has been a problem for decades. Computation power is usually underutilized due to time variety and other varieties. Our goal is to fully utilize that surplus capacity of the computing power but still keep the other jobs unaffected, which is also done by the scheduler with the congestion controller. The congestion controller allocates the CPU and GPU resource pool by additive increase/multiplicative decrease (AIMD) algorithm according to the system's load. The prover and verifier commit works to the worker scheduler, and the scheduler schedules and issues the works to the resource pools. The details are discussed in Section 6.

Since the PoRep module ensures data integrity, the storage system can actively find corrupted data with a low network overhead. This method reduces the need for a reliable hardware system and saves the cost of edge and IoT devices. Since the PoRep module can detect malicious nodes effectively, the DSPR system can even allow everyone's devices to join the storage system and provide their spare disk space.

Moreover, the erasure coding no longer needs high storage redundancy. The active integrity detection property allows high efficient erasure coding schemes like LDPC code, which is hard to tolerate arbitrary three or more nodes failure. Therefore, the system can provide higher performance in data fetching, recovering, and keeping very low storage redundancy without sacrificing reliability.

### 4.3. Decentralized P2P network

Our system is decentralized. The nodes all have the same functionalities and are connected by a P2P network. We implemented the network communication with the Rust's *libp2p* library [39] to achieve high performance, security and reliability. The *libp2p* is already widely used in P2P applications like IPFS, a well-known P2P file system [23].

The P2P network will transmit the data blocks and metadata from the block distributors to the block collectors, and transmit the proofs generated by the provers to all the verifiers. However, it is inefficient and impractical to broadcast every proof to every node in the whole network for large-scale deployment of edge and IoT devices. Therefore, the network is partitioned into sectors, called prover sectors. A storage node joins one or more prover sectors. A proof generated by the storage node will only be sent to the other nodes in the same prover sector. Thus different sectors will have different consensus seeds and can have different proving period times.

The concept of the prover sector is only a partition in the nodes' PoRep modules. The data blocks are ensured to be stored correctly by the peer nodes in the prover sector. While the nodes from different prover sectors can still be interconnected in the network. And the encoded data blocks of a single file can also be distributed to different prover sectors.

The network usage of publishing proofs is little. For one PB data with a moderate 4 GB sector size, only 64 MB of proofs will be needed per day. However, the 64 MB proofs need to be received and verified by every node in the prover sector. Therefore the topology of prover sectors should be well designed, which will be our future work.

### 4.4. Reliability & security

The devices are less controlled and less stable in edge and near user environments due to cost efficiency. We need to assess the stability of the edge devices to design the suitable erasure coding scheme and proving period in our system to ensure reliability.

Based on the datasheet of Nvidia's edge platform: Nvidia Xavier AGX [40], the failure rate of Xavier AGX in the ground mobile environment is 1756 FIT. So for $10^5$ devices deployed like the scale of the aBeacon system [1], there will be approximately 1540 failures per year, 4.21 failures per day. Therefore, the failure probability of k storage nodes in an erasure coding scheme with $N$ nodes in one day will be:

$$\binom{N}{k} \cdot (\frac{4.21}{10^5})^2$$

For $k = 2$ and a typical erasure coding configuration $N = 10$, the failure possibility is approximate $1 \times 10^{-7}$, i.e., the availability is 99.99999% for this configuration, which is more reliable than AWS EBS's 99.999 percent availability [41]. Therefore, for an erasure coding scheme with $N \approx 10$ and one-day proving period configuration, it is reliable enough to use any erasure code that can tolerate arbitrary two nodes failure. This requirement is far less strict than the arbitrary three nodes failure like Azure-LRC [8] to achieve much higher efficiency and lower storage overhead.

For the security aspect, we first define a threat model. The edge and IoT devices can be easily physically accessed. They are highly vulnerable to attacks. The threat model is based on that every node in the network in our system may be a compromised and malicious node. Ideally, as long as there exists one honest and working node in the network, the data on the whole network can be available and reliable. However, in reality, due to the limited computing power of edge devices, there need to be enough honest and working nodes at the same time in the network.

RSA key pairs certificate the storage nodes. For systems that need higher security can use private network parameters and disable and block unauthorized nodes. To enhance the security when possible malicious users join, the sensitive data may be encrypted by the application and stored as an encrypted file by the system.

Since the proofs of replication are verified by multiple nodes, and every node can verify the proof and recover the data if the proof is invalid, the well-known 51% attack in the blockchains will not be effective in our system. As long as there are honest nodes in the network, the data corruption can be detected and recovered.

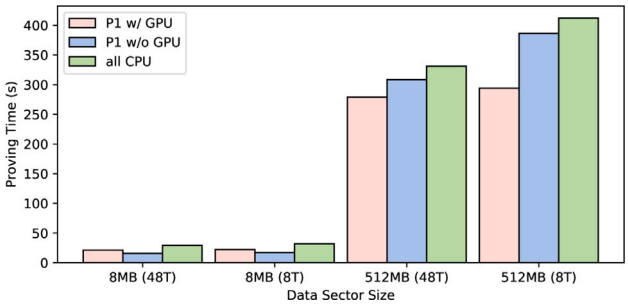



**Fig. 5.** The benchmark results of prover with different configuration.

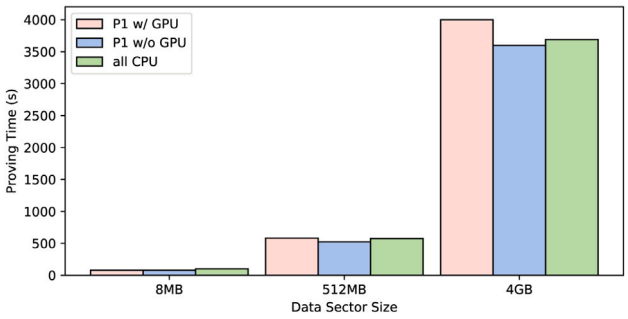



**Fig. 6.** The benchmark results of prover on AGX.

## 5. Prover and verifier

The proof of replication scheme in our system is based on the proving library of Filecoin. The proof scheme consists of 4 phases:

*PreCommit Phase 1*: In this phase, the data sector is encoded by Stacked DRG (SDR) encoding and encoded into a replica $\mathcal{R}$. An ID for this replica will be generated to identify that the data sector contains the data correctly. The SDR ID can allow the storage to detect the corrupted data itself and recover them like a hash code.

*PreCommit Phase 2*: In this phase, a Merkle tree is generated by the Poseidon hashing algorithm.

*Commit Phase 1*: In this phase, the hashed Merkle and the encoded replica $\mathcal{R}$ are processed and prepared for the next phase to generate a proof.

*Commit Phase 2*: In this phase, the hashes are compressed to generate a zk-SNARK proof. The proof is compressed to a small size and broadcast to the network.

The SDR encoding algorithm needs some fixed-size data sector. Our prover and verifier support various data sector sizes from 2 kB to 64 GB. The larger data sector size needs more RAM and more time to prove, but it is more cost-efficient. For the best practice, we are using 8 MB, 512 MB, and 32 GB sector sizes for conventional x86 machines based on our benchmark results. For ARM IoT devices with limited RAM and resources, we are using customized sector sizes based on the device's resources. On the Xavier AGX we tested, we are using 4 GB maximum sector size instead of 32 GB. The files that need to be proven should choose a data sector size larger than the file size and be padded with 0 in proving. Therefore, no matter how small the actual file is, the proving time of the same data sector size is the same, which is inefficient for small files. Therefore, instead of proving each file, our prover groups the files into proving groups and prove the group within one data sector to improve the efficiency and reduce the number of proof that needs to be broadcasted. Similarly, the verifier will also verify the files as groups.

To group the data blocks into sector groups efficiently, we first benchmarked the performance of the prover with different sector sizes and configurations. If the data sector is small, it will be inefficient to use GPU since the GPU context initialization and data transmission will introduce significant overhead. Therefore, the proving speed of with and without GPU is tested. We benchmarked proving one sector of 8 MB and 512 MB on a server with 24 Xeon v4 cores (48 threads). We also benchmarked the proving time with a lower limited core count (8 cores) to see the performance on less powerful edge servers. The results are shown in Fig. 5.

The benchmark results show that it is faster for 8 MB sector size to prove the sector without GPU. The proving of 512 MB data sector spends 17.75x of the 8 MB proving time on our machine but can prove 64 times more size the 8 MB data sectors. Therefore, it is more efficient to gather 18 groups of 8 MB group into one 512 MB group on our testing machine. Since the result may vary on different machines, the benchmark will be firstly employed, and the result will be stored in the configuration to improve proving efficiency. Then based on the benchmark result, the prover will arrange the stored data blocks into the proving group with the optimized configuration.

For ARM IoT devices, the benchmarks are also performed on the Xavier AGX. The results on AGX are shown in Fig. 6.

From the benchmarks, we can see that using GPU in P1 still negatively impacts the performance of larger data sectors due to the limited resource on AGX. However, since multiple provers will be executed in parallel, offloading the works to GPU can still improve overall performance.

**Table 1**
The CPU and GPU demand for different proving phases.

| Phase | CPU/GPU demand | Operation |
|---|---|---|
| preCommit Phase 1 (P1) | 1 core | PoRep SDR encoding |
| preCommit Phase 2 (P2) | GPU/All cores w/o GPU | Merkle tree generation |
| Commit Phase 1 (C1) | All cores | Proof generation |
| Commit Phase 2 (C2) | GPU/All cores w/o GPU | Proof compression |
| Verifying | 1 core | Proof verification |

## 6. Prover scheduler

The prover has different phases of computation. These phases demand different resources shown as Table 1. The SDR encoding operation in the pre-commit phase 1 cannot be parallelized due to the limitation of the cryptography algorithm. However, the following phases can fully utilize the multi-core performance. It is necessary to schedule the different phases to achieve the best performance. In most systems, the GPU resources are the most limited resource. Therefore the commit phase 2 and the pre-commit phase 2 for large sector sizes must be prioritized, and the former phases should be carefully scheduled to fully utilize the GPU. Therefore, we design and implement a scheduler to fully utilize the CPU and GPU resources and eliminate bottlenecks as much as possible. The scheduler is shown as Fig. 7.

In the system, the bottleneck is the P1 phase and the limited GPU resource. The strategy of the scheduler is a kind of multi-level priority queue. The congestion controller module in the scheduler, which will be described in the next part of this section, first creates a thread pool and a GPU pool based on the device and the load of the OS to avoid the provers affect the performance of the other applications. The scheduler has 4 CPU queues for the four corresponding prove phases and 2 GPU queues for P2 and C2 that GPU can accelerate in the pools. At the start of a proving period, the provers of the proof groups commit their current phase to the scheduler and wait for the scheduler to launch their work phases. After the current phase finishes, the next phase of the prover will be enqueued to the next phase's queue. The head in each queue has the highest priority in the queue. With higher priority, the prover has more threads to execute the proving. The queues are classified as CPU queues and GPU queues. As described in Section 5, different proof group size has different phase configuration based on the system's hardware performance. The proving phase will be

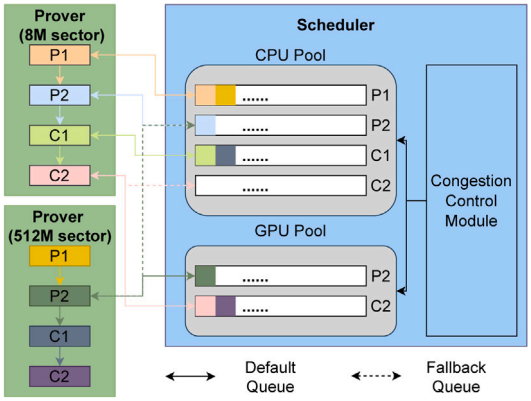



**Fig. 7.** The prover scheduler of DSPR.

enqueued to the corresponding CPU or GPU queue. If the GPU queues have too many works, but the CPU utilization is low, the provers can also be enqueued to the CPU queue to balance the load.

For GPU queues, since the GPU resource cannot be flexibly scheduled, C2 phases are first served to finish proving and send proof as soon as possible. The provers in the same queue are first come, first served. For CPU queues, the P1 queue has the lowest priority among the four queues and gives up the thread for P2 and C1 to prepare data for the phases using GPU.

At the start of a proving period, the provers of the proof groups commit their work phases to the scheduler and wait for the scheduler to launch their work phases. The congestion controller module in the scheduler, which will be described in the next part of this section, first creates a thread pool and a GPU pool based on the device and the load of the OS to avoid the provers affecting the performance of the other applications. The scheduler first launches a number of P1 based on the thread pool's size. When the P2 is committed to the scheduler after the prover finished P1,

On the other hand, the PoRep operations need a lot of computing power. Usually, the computing power of the servers is underutilized due to the load diversity of time. The edge devices have more diversity than cloud servers, like different locations and edge/IoT platforms. The surplus computing power can be used to compute the PoRep. However, we do not want the proving system to affect the device's regular applications' performance. However, it is hard to get and fit the workload of the operating system accurately. We do not want the pulse in the utilization of the regular applications affected by the proof system. Increasing the nice value of the proof system will give the CPU time to other applications but will make the prover threads in contention with each other and heavily impact the efficiency. Therefore, we employed a congestion control scheduler like the TCP congestion window to dynamically allocate the threads for provers.

In TCP connections, TCP uses an additive increase/multiplicative decrease (AIMD) scheme to limit the connection speed to avoid congestion in the network and makes all clients share the gateway bandwidth fairly. In our scheduler, a similar AIMD scheme is applied. The thread number $T$ at time $\mu$ is given as the formulas below:

$$T(\mu+1) = \begin{cases} T(\mu) + a_\mu & \text{if there are idle cores} \\ T(\mu)/2 & \text{if no idle core} \end{cases}$$

$$a_{\mu+1} = \begin{cases} 1 & \mu = 0 \\ 2 \cdot a_\mu & \mu \neq 0, \text{there are idle cores} \\ 1 & \mu \neq 0, \text{no idle core} \end{cases}$$

With the AIMD scheme, provers can reduce the threads or increase the threads according to the utilization of the system. The idle and loaded threshold and timing are carefully tuned. The scheduler can quickly give up cores for the utilization peaks and slowly grow to fully utilize the computing power.

**Table 2**
Tested x86 machine's hardware.

| | |
|---|---|
| CPU | 2 x Intel Xeon CPU E5-2650 v4 @ 2.20 GHz |
| Memory | 2 × 8 × 16G DDR4 2400 |
| GPU | Nvidia RTX3080 |
| Disk | Hikvision C2000 pro 256 GB |
| OS | Ubuntu 20.04.1 LTS |

**Table 3**
Xavier AGX's hardware.

| | |
|---|---|
| CPU | 8-core ARM v8.2 @ 1.19 GHz |
| Memory | 32 GB LPDDR4x |
| GPU | 512-core NVIDIA Volta GPU |
| Disk | Hikvision C2000 pro 256 GB |
| OS | Ubuntu 18.04.5 LTS |
| Platform power | 30 W |

## 7. Evaluation

We benchmarked several aspects to evaluate our distributed storage system: erasure coding efficiency, proving and verifying speed, scheduler performance, and efficiency. We tested the system on a traditional x86 machine with a GPU and on an IoT device, Nvidia Xavier AGX [2], which was released in 2018 and is now used in many edge and IoT environments like auto-driving. The environments of the tested systems are listed as Tables 2 and 3.

### 7.1. LDPC code

We applied a simple LDPC coding as a high efficient erasure coding scheme in our system, as we discussed in Section 3.3. We benchmarked the erasure coding scheme and compared it with the Intel ISA-L library [32] and LRC coding from Sina storage [42]. We benchmarked the average encoding time and block recovering time of ISA-L's RS coding and LRC coding with AVX2 and our LDPC coding with AVX2/Neon and without SIMD instructions. The coding parameter uses $(12, 4)$ to achieve a relatively high reliable erasure coding scheme in edge environments. The benchmark results are shown in Figs. 8 and 9.

We can see that, unlike the RS code, the pure XOR-based erasure coding scheme has good performance without SIMD instructions and performs well in decoding. In some situations when the CPU frequency is critical, and AVX needs to be avoided, the pure XOR coding scheme can still perform well.

However, for edge devices with ARM cores, the SIMD performance with Neon instructions is much lower than AVX due to its narrower SIMD width and higher memory latency. The pure xor operations also do not cost much computing power. Therefore, the performance with SIMD is even lower than without SIMD instructions due to more SIMD load and store overhead. On the other hand, Sina storage's LRC coding is specifically optimized for neon instructions. Therefore its performance is much better on ARM devices.

Overall, on the x86 server, our LDPC coding scheme achieved a 5.8x and 4.9x encoding speedup and 19.6x and 2.5x recovering speedup on average than Intel ISA-L's RS coding and LRC coding, respectively. The edge device AGX speeds up the encoding and recovering by 7.9x and 34x than ISA-L, and 3x and 1.7x than LRC coding, respectively.

### 7.2. Proving & verifying time

The storage nodes need to compute and provide the PoRep of its stored data in one period. Today's edge and IoT devices are becoming very powerful. Nvidia Xavier AGX (2018) has 8 ARM v8.2 cores and 512-Core Volta GPU with Tensor Cores [2]. Intel's edge product line Xeon D can even have 16 cores (32 threads, 2016) [43]. They are capable of proving many data within one period.

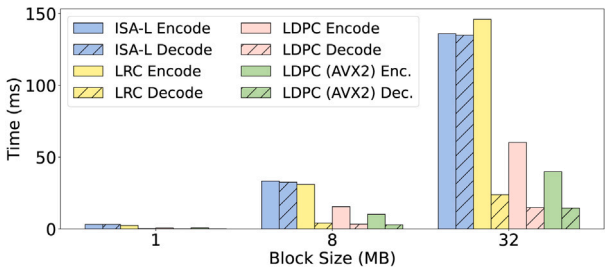



**Fig. 8.** Encoding & recovering time of different coding schemes on x86 server.

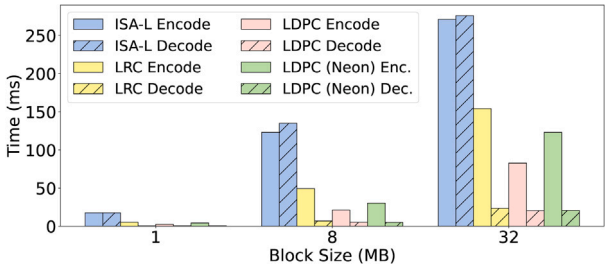



**Fig. 9.** Encoding & recovering time of different coding schemes on AGX.

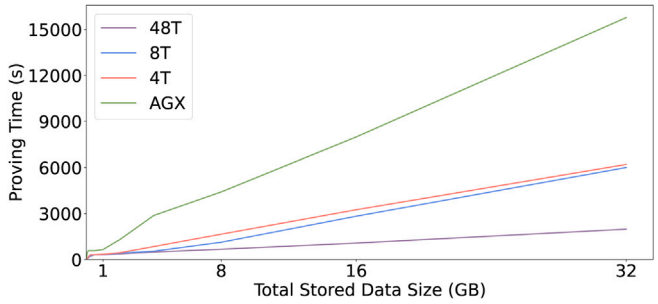



**Fig. 10.** Proving time from 8M to 32G stored data.

We evaluated the time needed to prove different sizes of data. We tested the performance on our server with 48 threads and the Nvidia Xavier AGX. To evaluate the performance on edge and IoT devices with x86 CPU like Xeon D, we also tested the system by limiting the server by 8 threads and 4 threads. The test results are shown in Fig. 10.

For stored data size lower than 256 MB, the proving time grows quicker since the proving group size uses 8 MB. For larger data size, the system will use 512 MB proving group size to improve efficiency. From 512 MB to 8G, the slope increases. It is because the P1 phases are not fully parallelized and become the bottleneck. Starting from 8 GB, the proving time becomes linearly increasing. Similarly, when thread count becomes lower, the impact of low thread count also becomes lower.

For one day period, based on the benchmark results from 4T, an x86 edge device with 4 threads can prove 445G data within one day, which is sufficient for a lightweight edge device. For the Xavier AGX, the proving work takes about 4x more time than the x86 server with the same 8 CPU threads due to the smaller and slower memory and much lower power consumption. Despite this, it can still prove and provide 180G data within the period.

The verification of the PoRep is very simple and fast. On the x86 server, verifying one PoRep with a single thread takes only 2.4 ms to verify the PoRep of the 8 MB proving group. It takes 2.79 ms to verify the PoRep of 512 MB proving group. For different sizes of proving groups, the verification time is similar. On the AGX, the verification takes 6.21 ms on average for different sizes of proving groups. The verification is used to verify the proof of replications from other nodes

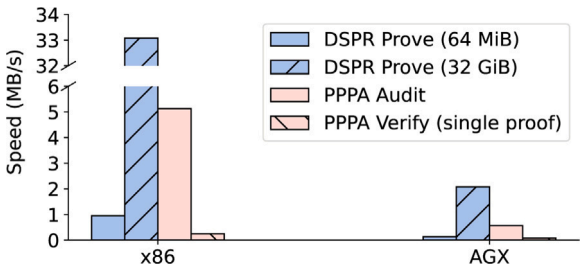



**Fig. 11.** Proving and verifying speed of DSPR and PPPA.

to ensure the data are reliably stored without downloading the entire file. An intermediate hash is checked when using the downloaded data to ensure the data integrity with negligible overhead.

The public auditing scheme PPPA [18] is also tested and compared as Fig. 11. With the same 80-bit security level, we benchmarked the proving speeds of proving 64 MB and 32 GB with the optimal proving group sizes, and the auditing speed and the speed of verifying one proof in PPPA scheme. The auditing and verifying speeds are almost constant for different sizes of data.

For low amount storage, the proving algorithm in our system has very high overhead and very low speed. However, when the amount of stored data increases, the overhead will be amortized. Therefore, the performance is much higher than the PPPA scheme for larger data storage. DSPR's proving speed is 6.5x faster than PPPA on the x86 server. And 3.7x faster on the low-power IoT device Xavier AGX.

For verification, the verifying time of DSPRis negligible, so that it is not shown in the figure. But PPPA's verification speed is extremely slow at 0.25 MB/s on the x86 server and 0.08 MB/s on the AGX.

### 7.3. Scheduler

To test the efficiency of the scheduler in our system, we benchmarked the performance of proving 100 512 MB sectors (50 GB total) and 256 8 MB sectors (2 GB total) with and without the scheduler. The scheduler achieves 13.37x speed up in proving time by improving the efficiency and utilization of the system. The utilization during proving 100 512 MB sectors with and without scheduler are shown in Fig. 12. The overall utilization without the scheduler is very low and consists of a massive amount of surges. The scheduler effectively eliminates the surges and mainly fully utilizes the CPU and GPU resources.

There are three phases in the figure. The first phase that fully utilizes the CPU is parallelized P1 provers. The second phase that CPU utilization drops and GPU is utilized is when some workers finish P1 and start to use GPU to compute P2. The CPU utilization increases in this phase because after P2 finishes, provers use the CPU to compute C1. In the last phase, CPU utilization drops because C1 workers finish, and GPU does the last C2 works.

However, the utilization of the GPU still varies a lot and is still underutilized. That is because the GPU program still lacks more optimizations for the cryptographic algorithms. Moreover, the GPU data loading and reading are still not optimized for the prover pipeline.

### 7.4. CPU congestion control

Our system goals to utilize the surplus computing power without sacrificing other running applications. The congestion control unit does this in the scheduler. We evaluated the effectiveness of our congestion control unit by running the sysbench with 48, 36, 24, and 12 CPU threads during the proving period. To visually see the effectiveness of the congestion control unit, the CPU utilization of sysbench and our DSPR system during the test is shown in Fig. 13.

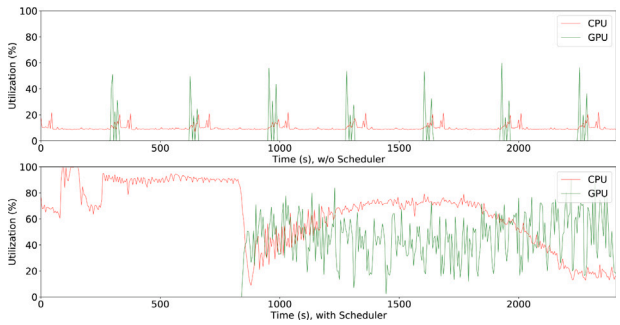



**Fig. 12.** The system resources utilization during proving (only part of the w/o scheduler condition to match the same duration).

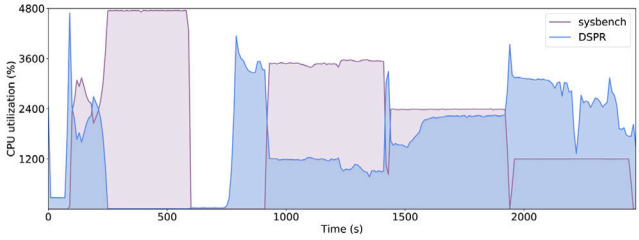



**Fig. 13.** The CPU utilization of sysbench and DSPR when running simultaneously.

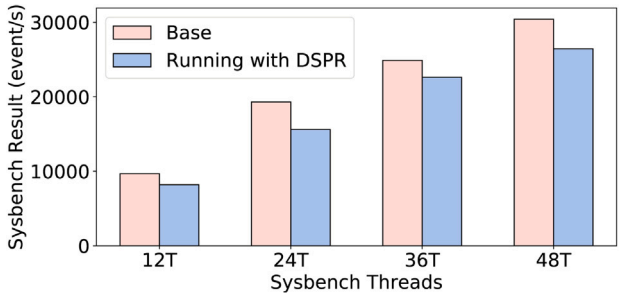



**Fig. 14.** The benchmark results of sysbench when running with DSPR.

The tests are done without setting the nice value of DSPRor sysbench. In these tests, we want to show that the DSPR system can dynamically manage the resource without affecting the other applications. By setting the nice value, the results can be better.

Fig. 13 shows the effectiveness and limitation of our congestion module. We start the proving period and run the sysbench with 48, 36, 24, 12 threads for 500 s each in sequence. At the start of proving, the DSPR system does the initializing works like reading the disk, allocating memory, and preparing the resource pool. That is why the CPU utilization behaves strangely at the beginning. And the congestion control did not act responsively initially since the DSPR system has not come to a stable state.

After that, we can see that the congestion control acts well. It quickly gives up the CPU when other application starts and takes CPU when the system has idle CPUs. We can see the DSPR takes CPUs during the gaps of sysbench and drops as soon as the next sysbench starts. The response of congestion control takes almost immediately. Around 1500 s in the figure, we can see the AIMD takes effect. DSPR first quickly drops to a level that will not affect the sysbench's performance. And then gradually increase to utilize the remaining idle CPU threads.

The sysbench's results are shown in Fig. 14. When running the sysbench and DSPR's proving system simultaneously, the sysbench is affected by 13% on average. This result is not as good as expected, even though the sysbench is actually running at the specified CPU threads. There are two reasons. Sysbench requires high memory bandwidth to execute. Any program running along with it would affect the memory bandwidth and the benchmark result of sysbench. Modern CPUs have dynamic frequency scaling features like Intel Turbo Boost. The CPU can run at a higher frequency when fewer cores are running. That is why a load of other programs affected sysbench's result at the lower thread count. However, these two aspects are highly hardware and application related. It is hard to predict and still needs tuning.

## 8. Conclusion & future work

This paper introduces DSPR, a decentralized distributed storage system for edge and IoT devices with solid reliability and security provided by a proof-of-replication scheme. By periodically proving and verifying the stored data, the PoRep system enables the storage system to actively detect data corruption and storage node failure. This property makes the system reliably store and provide data to applications in unstable and compromised environments like edge and IoT devices. A one-day proving period can provide higher reliability than enterprise-level storage systems with unreliable edge and IoT devices.

Our optimized system can prove 4x faster than public auditing schemes like PPPA on low-power IoT devices with the same 80-bit security level and even stronger threat model. The NIZK proofs in our scheme also provide many more benefits than interactive challenge schemes in public auditing schemes.

Moreover, it also makes some highly efficient erasure coding schemes like the LDPC code applicable in the distributed storage, formerly inapplicable due to the lack of tolerating three or more arbitrary node failures. The LDPC code used in our system provides 5.75x encoding speedup and 19.6x recovering speedup than the RS code of the Intel ISA-L library. It can also compute efficiently without SIMD instructions, of which the performance may be weak or inapplicable in edge and IoT CPUs.

Generating the proofs needs high computing power. The edge and IoT devices are evolving quickly and can provide the computation needed for generating proofs. The scheduler and the resource-aware congestion control unit in our system can dynamically manage the computing resources to fully utilize the underutilized CPU without affecting the other applications running on the same device. A lightweight edge server with 4 CPU threads can finish proving 445G data in one day period. And a low-power IoT device can prove 180G data in the period. However, the congestion control unit cannot perfectly manage and cooperate with the memory bandwidth and the CPU's dynamic frequency. It still needs to be developed and tuned for different environments and situations. On the other hand, the system still needs many improvements. These will be our future work.

Our network protocol does not open the support of file and block deletion for the security issue for now. When updating a file, the system will store the updated blocks as new blocks, but the old blocks and the old file metadata will not be deleted. Since our threat model is that everyone can be malicious and compromised, the deleting and modifying operations cannot be verified. The old data must be kept to prevent malicious operations. In a slightly relaxed threat model that the private key and the trusted certification list on a node are protected, the storage node in our system sign the operations with its RSA private key, and then the operations can be audited. We will move these parts into TEE like Intel SGX and ARM TrustZone in our future work.

Furthermore, the system still lacks large-scale testing due to our hardware limitation. The data distribution algorithm still lacks optimizations like geometric distribution-based algorithms. The GPU program of the PoRep module still needs to be further optimized. These are the future works.

## Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

## Acknowledgments

This work is sponsored by Shanghai Pujiang Program, China 19PJ1430900, Shanghai Key Laboratory of Scalable Computing and Systems, China, and supported in part by NSF of China (NO. 61732010).

## References

[1] Y. Ding, L. Liu, Y. Yang, Y. Liu, D. Zhang, T. He, From conception to retirement: a lifetime story of a 3-year-old wireless beacon system in the wild, in: 18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21), USENIX Association, 2021, URL: https://www.usenix.org/conference/nsdi21/presentation/ding.

[2] NVIDIA Corporation, Jetson AGX xavier developer kit, 2021, https://developer.nvidia.com/embedded/jetson-agx-xavier-developer-kit.

[3] J. Xing, H. Dai, Z. Yu, A distributed multi-level model with dynamic replacement for the storage of smart edge computing, J. Syst. Archit. 83 (2018) 1–11, http://dx.doi.org/10.1016/j.sysarc.2017.11.002, URL: https://www.sciencedirect.com/science/article/pii/S1383762117303119.

[4] I.S. Reed, G. Solomon, Polynomial codes over certain finite fields, J. Soc. Ind. Appl. Math. 8 (2) (1960) 300–304.

[5] S. Muralidhar, W. Lloyd, S. Roy, C. Hill, E. Lin, W. Liu, S. Pan, S. Shankar, V. Sivakumar, L. Tang, S. Kumar, f4: Facebook's warm BLOB storage system, in: 11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14), USENIX Association, Broomfield, CO, 2014, pp. 383–398, URL: https://www.usenix.org/conference/osdi14/technical-sessions/presentation/muralidhar.

[6] D. Ford, F. Labelle, F. Popovici, M. Stokely, V.-A. Truong, L. Barroso, C. Grimes, S. Quinlan, Availability in globally distributed storage systems, 2010.

[7] S. Weil, S. Brandt, E. Miller, D. Long, A scalable, high-performance distributed file system, in: Proceedings of USENIX Symposium on Operating Systems Design and Implementation (OSDI), 2006.

[8] C. Huang, H. Simitci, Y. Xu, A. Ogus, B. Calder, P. Gopalan, J. Li, S. Yekhanin, Erasure coding in windows azure storage, in: 2012 USENIX Annual Technical Conference (USENIX ATC 12), USENIX Association, Boston, MA, 2012, pp. 15–26, URL: https://www.usenix.org/conference/atc12/technical-sessions/presentation/huang.

[9] M. Xia, M. Saxena, M. Blaum, D.A. Pease, A tale of two erasure codes in HDFS, in: 13th USENIX Conference on File and Storage Technologies (FAST 15), USENIX Association, Santa Clara, CA, 2015, pp. 213–226, URL: https://www.usenix.org/conference/fast15/technical-sessions/presentation/xia.

[10] Y. Hu, L. Cheng, Q. Yao, P.P.C. Lee, W. Wang, W. Chen, Exploiting combined locality for wide-stripe erasure coding in distributed storage, in: 19th USENIX Conference on File and Storage Technologies (FAST 21), USENIX Association, 2021, pp. 233–248, URL: https://www.usenix.org/conference/fast21/presentation/hu.

[11] P. Sobe, Failure-tolerant distributed storage with compressed (1 out-of n) codes, J. Syst. Archit. 54 (9) (2008) 861–867, http://dx.doi.org/10.1016/j.sysarc.2008.02.007, URL: https://www.sciencedirect.com/science/article/pii/S1383762108000362 Parallel, Distributed and Network-Based Processing.

[12] Y. Wu, D. Liu, X. Chen, J. Ren, R. Liu, Y. Tan, Z. Zhang, Mobilere: A replicas prioritized hybrid fault tolerance strategy for mobile distributed system, J. Syst. Archit. 118 (2021) 102217, http://dx.doi.org/10.1016/j.sysarc.2021.102217, URL: https://www.sciencedirect.com/science/article/pii/S1383762121001533.

[13] J.A. Halderman, S.D. Schoen, N. Heninger, W. Clarkson, W. Paul, J.A. Calandrino, A.J. Feldman, J. Appelbaum, E.W. Felten, Lest we remember: Cold boot attacks on encryption keys, in: 17th USENIX Security Symposium (USENIX Security 08), USENIX Association, San Jose, CA, 2008, URL: https://www.usenix.org/conference/17th-usenix-security-symposium/lest-we-remember-cold-boot-attacks-encryption-keys.

[14] E.-O. Blass, W. Robertson, TRESOR-HUNT: attacking CPU-bound encryption, in: Proceedings of the 28th Annual Computer Security Applications Conference, 2012, pp. 71–78.

[15] T. Markettos, C. Rothwell, B.F. Gutstein, A. Pearce, P.G. Neumann, S. Moore, R. Watson, Thunderclap: Exploring vulnerabilities in operating system iommu protection via DMA from untrustworthy peripherals, 2019.

[16] J. Benet, D. Dalrymple, N. Greco, Proof of replication, Protoc. Labs July 27 (2017) 20.

[17] I. Damgård, C. Ganesh, C. Orlandi, Proofs of replicated storage without timing assumptions, in: Annual International Cryptology Conference, Springer, 2019, pp. 355–380.

[18] C. Wang, S.S. Chow, Q. Wang, K. Ren, W. Lou, Privacy-preserving public auditing for secure cloud storage, IEEE Trans. Comput. 62 (2) (2013) 362–375, http://dx.doi.org/10.1109/TC.2011.245.

[19] J. Liu, K. Huang, H. Rong, H. Wang, M. Xian, Privacy-preserving public auditing for regenerating-code-based cloud storage, IEEE Trans. Inf. Forensics Secur. 10 (7) (2015) 1513–1528, http://dx.doi.org/10.1109/TIFS.2015.2416688.

[20] J. Wu, Y. Li, F. Ren, B. Yang, Robust and auditable distributed data storage with scalability in edge computing, Ad Hoc Netw. 117 (2021) 102494, http://dx.doi.org/10.1016/j.adhoc.2021.102494, URL: https://www.sciencedirect.com/science/article/pii/S1570870521000573.

[21] M. Bailleu, D. Giantsidi, V. Gavrielatos, D.L. Quoc, V. Nagarajan, P. Bhatotia, Avocado: A secure in-memory distributed storage system, in: 2021 USENIX Annual Technical Conference (USENIX ATC 21), USENIX Association, ISBN: 978-1-939133-23-6, 2021, pp. 65–79, URL: https://www.usenix.org/conference/atc21/presentation/bailleu.

[22] Protocol Labs, Ipfs powers the distributed web, 2021, https://ipfs.io/ (Accessed on 05/19/2021).

[23] J. Benet, Ipfs-content addressed, versioned, p2p file system, 2014, arXiv preprint arXiv:1407.3561.

[24] A. Jonathan, M. Ryden, K. Oh, A. Chandra, J. Weissman, Nebula: Distributed edge cloud for data intensive computing, IEEE Trans. Parallel Distrib. Syst. 28 (11) (2017) 3229–3242.

[25] K. Budati, J. Sonnek, A. Chandra, J. Weissman, Ridge: combining reliability and performance in open grid platforms, in: Proceedings of the 16th International Symposium on High Performance Distributed Computing, 2007, pp. 55–64.

[26] T. Wang, Y. Mei, X. Liu, J. Wang, H.-N. Dai, Z. Wang, Edge-based auditing method for data security in resource-constrained internet of things, J. Syst. Archit. 114 (2021) 101971, http://dx.doi.org/10.1016/j.sysarc.2020.101971, URL: https://www.sciencedirect.com/science/article/pii/S1383762120302186.

[27] J. Zhang, R. Lu, B. Wang, X.A. Wang, Comments on "privacy-preserving public auditing protocol for regenerating-code-based cloud storage", IEEE Trans. Inf. Forensics Secur. 16 (2021) 1288–1289, http://dx.doi.org/10.1109/TIFS.2020.3032283.

[28] M. Blum, P. Feldman, S. Micali, Non-interactive zero-knowledge and its applications, in: Providing Sound Foundations for Cryptography: On the Work of Shafi Goldwasser and Silvio Micali, 2019, pp. 329–349.

[29] S. Pan, T. Stavrinos, Y. Zhang, A. Sikaria, P. Zakharov, A. Sharma, S.S. P, M. Shuey, R. Wareing, M. Gangapuram, G. Cao, C. Preseau, P. Singh, K. Patiejunas, J. Tipton, E. Katz-Bassett, W. Lloyd, Facebook's tectonic filesystem: Efficiency from exascale, in: 19th USENIX Conference on File and Storage Technologies (FAST 21), USENIX Association, 2021, pp. 217–231, URL: https://www.usenix.org/conference/fast21/presentation/pan.

[30] D. Zhao, K. Burlingame, C. Debains, P. Alvarez-Tabio, I. Raicu, Towards high-performance and cost-effective distributed storage systems with information dispersal algorithms, in: 2013 IEEE International Conference on Cluster Computing (CLUSTER), 2013, pp. 1–5, http://dx.doi.org/10.1109/CLUSTER.2013.6702655.

[31] S. Kalcher, V. Lindenstruth, Accelerating galois field arithmetic for reed-solomon erasure codes in storage applications, in: 2011 IEEE International Conference on Cluster Computing, 2011, pp. 290–298, http://dx.doi.org/10.1109/CLUSTER.2011.40.

[32] I. Center, Intel intelligent storage acceleration library, 2016, https://software.intel.com/en-us/storage/isa-l.

[33] L. Yan, J. Xing, T. Wang, Z. Huo, J. Ma, P. Zhang, Write bandwidth optimization of online erasure code based cluster file system, in: 2013 IEEE International Conference on Cluster Computing (CLUSTER), 2013, pp. 1–8, http://dx.doi.org/10.1109/CLUSTER.2013.6702661.

[34] K.M. Greenan, X. Li, J.J. Wylie, Flat XOR-based erasure codes in storage systems: Constructions, efficient recovery, and tradeoffs, in: 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST), 2010, pp. 1–14, http://dx.doi.org/10.1109/MSST.2010.5496983.

[35] J. Plank, A. Buchsbaum, R. Collins, M. Thomason, Small parity-check erasure codes - exploration and observations, in: 2005 International Conference on Dependable Systems and Networks (DSN'05), 2005, pp. 326–335, http://dx.doi.org/10.1109/DSN.2005.86.

[36] T.C. Blog, On the dangers of intel's frequency scaling, 2017, https://blog.cloudflare.com/on-the-dangers-of-intels-frequency-scaling/.

[37] N. Eltayieb, R. Elhabob, A. Hassan, F. Li, A blockchain-based attribute-based signcryption scheme to secure data sharing in the cloud, J. Syst. Archit. 102 (2020) 101653, http://dx.doi.org/10.1016/j.sysarc.2019.101653, URL: https://www.sciencedirect.com/science/article/pii/S1383762119304606.

[38] X. Qin, Y. Huang, Z. Yang, X. Li, A blockchain-based access control scheme with multiple attribute authorities for secure cloud data sharing, J. Syst. Archit. 112 (2021) 101854, http://dx.doi.org/10.1016/j.sysarc.2020.101854, URL: https://www.sciencedirect.com/science/article/pii/S1383762120301405.

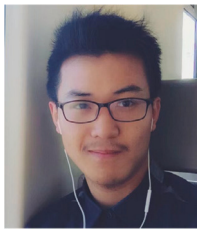
[39] Protocol Labs, Libp2p, 2021, https://libp2p.io/ (Accessed on 05/19/2021).

[40] NVIDIA Corporation, Data SHEET NVIDIA jetson AGX xavier series system-on-module, 2020, https://developer.download.nvidia.com/assets/embedded/secure/jetson/xavier/docs/JetsonAGXXavierSeriesDatasheet_DS09654001v1.2.pdf?wqjB7uykkcCPelJwzr0b4vz2bIky24ekF2r99PoaQM76JbVEJVAyyTmp7rfb4sL0vdcjgMVfFluOqbIlhlJyT_p0YIjhv1v_-XTyBvdLS8LQo3iGHl9X5ZBHzHBjjubEtjI5b1JY0EXtykK046oTSwFQbOCwrWLclkBv_lZ08S1F8XN9ELn9ZwiXXgrty0iof2VVHXFGDo9OWw.

[41] Amazon Corporation, Durability and availability, 2021, https://docs.aws.amazon.com/whitepapers/latest/aws-storage-services-overview/durability-and-availability-3.html.

[42] Y. Zhang, Drmingdrmer/lrc-erasure-code: Lrc(local reconstruction codes) erasure code based on reed-solomon with vandermonde matrix, 2016, https://github.com/drmingdrmer/lrc-erasure-code (Accessed on 08/11/2021).
[43] Intel Corporation, Intel® xeon® processor D-1587 (24m cache, 1.70 GHz) product specifications, 2016, https://ark.intel.com/content/www/us/en/ark/products/93365/intel-xeon-processor-d-1587-24m-cache-1-70-ghz.html (Accessed on 05/19/2021).

**Chenggang Wu** is currently a Ph.D. student in computer science of Shanghai Jiao Tong University. He received his B.Eng. degree in ECE from Shanghai Jiao Tong University. His research interests include edge computing and heterogeneous computing.



**Yongbiao Chen** received a Bachelor's degree in software engineering from Northwestern Polytechnical University in 2016 and he is currently pursuing his Ph.D. degree in the school of Electronic Information and Electrical Engineering, Shanghai Jiao Tong University. He is broadly interested in the application of machine learning and computer vision, with a primary focus on developing efficient algorithms for large-scale information retrieval.

**Zhengwei Qi** received his B.Eng. and M.Eng degrees from Northwestern Polytechnical University, in 1999 and 2002, and Ph.D. degree from Shanghai Jiao Tong University in 2005. He is an Associate Professor at the School of Software, Shanghai Jiao Tong University. His research interests include program analysis, model checking, virtual machines, and distributed systems.

**Haibing Guan** received Ph.D. degree from Tongji University in 1999. He is a Professor of School of Electronic, Information and Electronic Engineering, Shanghai Jiao Tong University, and the director of the Shanghai Key Laboratory of Scalable Computing and Systems. His research interests include distributed computing, network security, network storage, green IT and cloud computing.