International Journal of Safety and Security Engineering
Vol. 16, No. 6, June, 2026, pp. 1373-1385
Journal homepage: http://iieta.org/journals/ijsse

Role-Reversed Disaster Recovery of File System: A Faster Approach to Failback

Ravindra Reddy Sure*, Ravi Kumar Tata, Pardhasaradhi Varma Gottumukkala

Department of Computer Science and Engineering, Koneru Lakshmaiah Education Foundation, Vaddeswaram 522302, India

Corresponding Author Email: ravisure@in.ibm.com

https://doi.org/10.18280/ijsse.160618

Received: 21 April 2026
Revised: 13 June 2026
Accepted: 25 June 2026
Available online: 30 June 2026

Keywords:
disaster recovery, role reversal disaster recovery, failover, failback, file system disaster recovery

ABSTRACT

In file system-level disaster recovery (DR) configurations, asynchronous data replication is widely deployed between a local Primary file system and a remote Recovery file system to minimize application latency on the primary site. To maintain data consistency, synchronized peer snapshots are created periodically in accordance with the defined Recovery Point Objective (RPO) at both sites. In the event of a disaster impacting the Primary file system, the Recovery file system is restored to the most recent consistent snapshot, enabling application failover with assured data integrity. Once the Primary file system is restored to normal operation, the DR relationship must be re-established to maintain ongoing recoverability, followed by application failback to the primary site. This paper presents a novel and efficient failback procedure that ensures very minimal service disruption. The proposed approach establishes a temporary reverse DR relationship to synchronize the Primary file system with data changes accumulated at the Recovery file system during the outage. After synchronization is complete, applications are seamlessly failed back to the Primary file system, and the standard DR configuration is re-established between the Primary and Recovery file systems. This method significantly reduces downtime while preserving data consistency throughout the failback process.

1. INTRODUCTION

Digital information has become a foundational asset for modern enterprises, underpinning critical business operations, decision-making processes, and competitive advantage. Organizations must be prepared to restore their data following disruptive events such as natural disasters (floods, earthquakes), cyberattacks, hardware failures, human errors that may compromise the local file system and result in catastrophic data loss [1]. To maintain data availability and business continuity under such circumstances, organizations implement disaster recovery (DR) strategies that maintain a duplicate instance of the production (Primary) file system at a geographically separate (DR) site, referred to as the Recovery file system. This geographic separation ensures that regional disasters affecting the primary site do not simultaneously impact the recovery infrastructure.

Updates made on the Primary file system are continuously propagated to the Recovery file system to ensure data currency and high availability. With the rapid growth of cloud computing and the increasing adoption of hybrid cloud architectures, the Recovery file system may be deployed within a cloud infrastructure, offering advantages such as elastic scalability, reduced capital expenditure, and simplified management [2-4]. This cloud-based approach has democratized DR capabilities, making enterprise-grade protection accessible to organizations of all sizes.

Data transfer from the Primary to the Recovery file system is commonly implemented using asynchronous replication, which has become the de facto standard for geographically distributed DR configurations. This approach minimizes application I/O latency at primary site because write operations are acknowledged immediately without waiting for remote confirmation from the Recovery file system; however, this performance benefit introduces a fundamental trade-off; temporary inconsistencies may exist at the Recovery site due to replication lag [5]. The magnitude of this inconsistency window depends on factors including network bandwidth, geographic distance, write workload intensity, and the configured replication batch size.

To address these consistency concerns while maintaining the performance benefits of asynchronous replication, periodic snapshots [6] are created on both systems at coordinated intervals. These synchronized peer snapshots establish crash-consistent recovery points that are temporally aligned across the Primary and Recovery file systems, affectively defining discreate Recovery Point Objectives (RPOs). The RPO represents the maximum acceptable data loss measured in time—for example, an RPO of 15 minutes means that in a disaster scenario, up to 15 minutes of recent data changes may be lost. In asynchronous DR configurations, modified data can be transmitted continuously in the background as updates occur on the Primary file system [7], with snapshot creation serving as periodic consistency checkpoints that bound the potential data loss window.

When a disaster renders the Primary file system unavailable, – whether due to site-wide outages, storage system failures, or data corruption – applications must be rapidly redirected (failover) to the Recovery file system to maintain business continuity and minimize service disruption.

The failover process typically involves mounting the most recent consistent snapshot at the Recovery site and reconfiguring application endpoints to direct traffic to the Recovery file system. While failover procedures are well-established and frequently tested, the subsequent failback process presents more complex challenges.

After the Primary file system is restored to operational status – following hardware replacement, site recovery, or system repairs – applications should ideally be returned to it through a failback process to restore the intended DR topology and optimize performance by serving applications from the primary data center. However, before failback can occur safely, the Primary file system must be synchronized with the Recovery file system to capture all updates made during the outage period while applications were running at the recovery site. This synchronization step is critical to ensure that the restored Primary file system reflects the most recent and consistent state of the data, preventing any data loss during the transition back to normal operations. Traditional failback approaches often require extended maintenance windows, complete data replication, or application downtime, making them operationally expensive and disruptive to business operations.

This paper presents a novel and efficient failback mechanism that addresses these limitations through an innovative role-reversal approach. The proposed method begins by creating a dedicated "failback" snapshot on the Recovery file system prior to initiating synchronization, establishing a known-good baseline for the failback operation. A temporary DR relationship is then established in the reverse direction – from the Recovery file system to the Primary file system – effectively switching their roles in the replication topology. During this interval, the Primary file system operates logically as the Recovery target, receiving updates rather than sending them. Through this reversed relationship, only the modified data blocks (delta changes) accumulated during the outage are asynchronously replicated from the Recovery file system back to the Primary file system in the background, thereby optimizing data transfer efficiency and significantly reducing synchronization time compared to full data copies. This incremental approach leverages block-level change tracking to minimize network bandwidth consumption and synchronization duration.

Critically, this approach allows the Recovery file system to continue serving production application workloads during the synchronization process, eliminating the need for a maintenance window and ensuring continuous service availability. Once synchronization reaches completion and the Primary file system achieves consistency with the Recovery file system, applications can be seamlessly transitioned back to the primary site with minimal disruption—typically measured in seconds rather than hours. The original DR configuration is then re-established, restoring the Primary-to-Recovery replication topology and returning the environment to its intended operational state.

The main contributions of this work are as follows:

(1) Reverse DR Relationship Establishment: A mechanism to dynamically establish a reverse-direction DR relationship between the Recovery and Primary file systems, enabling temporary role reversal without disrupting the underlying storage infrastructure or requiring manual reconfiguration.
(2) Incremental Change Propagation: A method for efficiently propagating only delta changes accumulated on the Recovery file system back to the Primary file system once it becomes available, utilizing block-level change tracking to minimize data transfer volume and synchronization time.
(3) Non-Disruptive Synchronization: A technique that allows the Recovery file system to continue serving application workloads concurrently while synchronization to the Primary file system progresses in the background, eliminating traditional maintenance windows and ensuring continuous service availability.
(4) Automated DR Reconfiguration: An efficient procedure to automatically reinstate the original DR configuration (Primary-to-Recovery replication topology) after synchronization completes, including snapshot policy restoration and replication parameter reset.
(5) Seamless Application Failback: A failback strategy that enables applications to transition back to the Primary file system with minimal service interruption (typically sub-minute downtime), and guaranteed consistency through coordinated snapshot-based cutover.
(6) Operational Simplicity: A streamlined workflow that reduces operational complexity and human error potential compared to traditional failback procedures, which often require multiple manual steps, extended planning, and significant technical expertise.

This work assumes that the Recovery file system remains operational and continuously receives replicated updates before the disaster scenario, and that network connectivity between sites is restored before initiating the failback procedure. The proposed approach is applicable to various file system implementations and storage architectures, making it broadly relevant to enterprise DR deployments.

The remainder of this paper is structured as follows. Section 2 surveys related research. Section 3 provides background on IBM Storage Scale and its Asynchronous Disaster Recovery (ADR) capability. Section 4 describes the proposed role-reversal DR procedure, including reverse replication and failback operations. Section 5 presents experimental evaluation and results. Finally, Section 6 concludes the paper.

2. RELATED WORK

Most existing DR research focuses on reducing failover time, improving replication efficiency, or minimizing RPO [8-12]. Comparatively little attention has been given to failback optimization.

Patterson et al. [13] proposed a snapshot-difference (snapdiff)-based data replication mechanism designed to support asynchronous file system–level replication between geographically separated storage systems. The approach leverages native snapshot capabilities of modern file systems to efficiently detect and propagate incremental changes from a local (Primary) file system to a remote (Recovery) file system, thereby minimizing replication overhead and application impact.

2.1 Asynchronous replication using snapshot differencing

In this model, replication is driven by periodic, point-in-time snapshots. Let S_1 represent a snapshot at time t_1 and S_2 a subsequent snapshot at time t_2 (where $t_2 > t_1$) of the Primary file system. The snapdiff mechanism computes the logical

differences between S_1 and S_2 to identify all filesystem-level changes that occurred during the interval $[t_1, t_2]$.

The process typically involves:

(1) Snapshot Creation: Two consistent snapshots (S_1 and S_2) are created at different time instants. These snapshots preserve the metadata and data block mappings of the file system at those points in time, ensuring crash-consistent state capture.

(2) Inode and Directory Entry Analysis: An inode-level scan of S_2 is performed relative to S_1, typically with O(n) complexity where n represents the number of inodes in the filesystem. This scan detects:

- Newly created inodes (files or directories),
- Deleted inodes,
- Modified inodes (including metadata updates such as size, timestamps, permissions, extended attributes),
- Changes in directory entries (additions, deletions, renames).

(3) Change Set Generation: The output of the scan is a structured list of modified filesystem objects. Rather than copying entire files, the system identifies the specific data blocks and metadata elements that have changed between S_1 and S_2, minimizing data transfer volume.

(4) Operation Reconstruction and Replay: The detected differences are transformed into a sequence of filesystem operations (e.g., create, delete, write, truncate, rename). These operations are transmitted to the Recovery file system and replayed in order, ensuring logical consistency with the Primary file system state represented by S_2.

This design provides several advantages:

(1) Reduced network bandwidth consumption through incremental updates,

(2) Lower replication latency compared to full file synchronization,

(3) Minimal performance impact on applications due to asynchronous execution,

(4) Snapshot-consistent replication aligned with the configured RPO.

2.2 Failover procedure with failover snapshot

In the event of a disaster affecting the Primary file system, a controlled failover process is initiated to transition operations to the Recovery file system while preserving consistency.

(1) Failover Snapshot Creation: A failover snapshot (Sf) is identified on the Recovery file system corresponding to the most recent peer snapshot that was successfully replicated from the Primary file system before the disaster event. This snapshot represents a crash-consistent recovery point and serves as a baseline for subsequent failback operations.

(2) Activation of Recovery file system: The Recovery file system is promoted to active status, and applications are restarted using the state captured in the failover snapshot Sf.

(3) Continued Operation and Divergence: After a failover, all new data modifications occur on the Recovery file system, creating state divergence between the Primary and Recovery file systems, where the Recovery file system advances beyond the last synchronized state represented by Sf.

The failover snapshot Sf plays a critical role as a temporal reference point for tracking changes that occur during the Recovery site's active operation period.

2.3 Snapdiff-based failback procedure using failover Snapshot

Once the Primary file system is restored to operational status, a failback process is required to re-synchronize it with the changes made at the Recovery file system, ensuring no data loss. The snapdiff mechanism utilizes the failover snapshot Sf of the Recovery file system as the baseline for identifying changes made during the failover period.

The snapdiff-based failback procedure follows an iterative synchronization model.

(1) Snapshot creation on Recovery file system: A new snapshot (Sn) is taken on the active Recovery file system at the start of the failback process (and during subsequent iterations).

(2) Snapdiff Computation: The differences between the failover snapshot (Sf) and the latest new snapshot (Sn) are computed. This captures all changes that occurred on the Recovery file system since failover.

(3) Reverse Replication to Primary: The identified changes are converted into filesystem operations and applied to the restored Primary file system.

(4) Iterative Synchronization: As applications continue running on the Recovery file system, additional updates occur. Therefore, multiple iterations are performed:

- A new snapshot (Sn+1) is taken,
- Differences between the previous snapshot (Sn) and the current snapshots (Sn+1) are computed,
- Incremental updates are replicated to the Primary file system.

This iterative process progressively reduces the data gap.

(1) Final Synchronization Phase: Applications on the Recovery file system are stopped to quiesce I/O operations. A final snapshot is taken, and the remaining differences (from the last iteration) are replicated to the Primary file system, ensuring complete synchronization.

(2) Application Transition: After the final synchronization completes, applications are restarted on the Primary file system, completing the failback process and restoring the original DR topology.

2.4 Limitations of the snapdiff failback model

While effective in preserving data consistency, the iterative snapdiff-based failback approach has a critical limitation: the final synchronization phase introduces unavoidable downtime. Since applications must be stopped to capture the most recent changes without concurrent modifications, the duration of the last diff computation and replication directly determines service outage time.

For large-scale file systems with substantial write activity, the final delta can still be significant. For example, in a filesystem experiencing 100 MB/s write throughput, even a 5-minute final synchronization window would require transferring 30 GB of data, potentially resulting in 10-30 minutes of total downtime depending on available network bandwidth. This results in:

(1) Prolonged application unavailability that may violate

Service Level Agreements (SLAs).
(2) Increased operational risk during the cutover window,
(3) Measurable business impact due to delayed transaction processing and revenue loss.
(4) User experience degradation and potential customer dissatisfaction.

These limitations motivate the need for more efficient failback mechanisms that minimize or eliminate the lengthy final synchronization window while maintaining strict data consistency guarantees.

3. BACKGROUND

The failback mechanism proposed in this paper has been designed, implemented, and validated using IBM Storage Scale [14] and its ADR capability [15]. While the concepts presented are applicable to various distributed file systems, the implementation details and performance characteristics are specific to this platform. This section provides essential background on IBM Storage Scale architecture and the ADR replication framework that underpins the proposed failback procedure.

3.1 IBM storage scale

IBM Storage Scale [14], formerly known as General Parallel File System (GPFS), is a high-performance, POSIX-compliant clustered parallel file system that enables concurrent access to shared storage from multiple nodes. Originally developed by IBM Research, it has evolved into an enterprise-grade solution supporting petabyte-scale deployments with thousands of concurrent clients.

The system employs a distributed data placement strategy, striping file data across multiple disks within the file system to achieve balanced storage utilization and eliminate I/O bottlenecks. By distributing blocks from a single file across multiple physical disks, Storage Scale enables concurrent read and write operations from multiple nodes, thereby improving aggregate I/O throughput—achieving bandwidth exceeding 1 TB/s in large-scale deployments.

Storage Scale clusters consist of nodes with distinct roles. Application nodes execute user workloads and generate I/O requests, while I/O servers (also called storage nodes) provide direct access to physical storage devices. Inter-node communication utilizes TCP/IP over Ethernet networks or Remote Direct Memory Access (RDMA) over InfiniBand fabrics, with the later providing lower latency (<2 us) and higher bandwidth for data-intensive workloads.

Application nodes access remote storage through the Network Shared Disk (NSD) protocol, which abstracts physical storage devices as network-accessible block devices. NSD supports multiple transport protocols including TCP/IP for Ethernet networks and VERBS RDMA for Infiniband, enabling high-speed data access to storage attached to I/O servers as shown in Figure 1. I/O servers are typically connected to storage arrays through dedicated Storage Area Networks (SANs) using protocols such as Fiber Channel, iSCSI, FCoE (Fibre Channel over Ethernet) or NVMeOF (NVMe over Fabric). Alternatively, in a Network Attached Storage (NAS) configuration, storage may be accessed over general-purpose IP networks.

IBM Storage Scale employs a distributed token-based locking framework to coordinate access to shared storage resources. This mechanism maintains data and metadata consistency while supporting high level of concurrent operations. The locking protocol operates at byte-range granularity, allowing multiple nodes to simultaneously access non-overlapping regions of the same file. When access conflicts arise (e.g., overlapping write operations), the system enforces serialization through distributed lock arbitration, ensuring correctness while maximizing parallelism. This design enables efficient multi-node access patterns common in high-performance computing (HPC) and data analytics workloads.

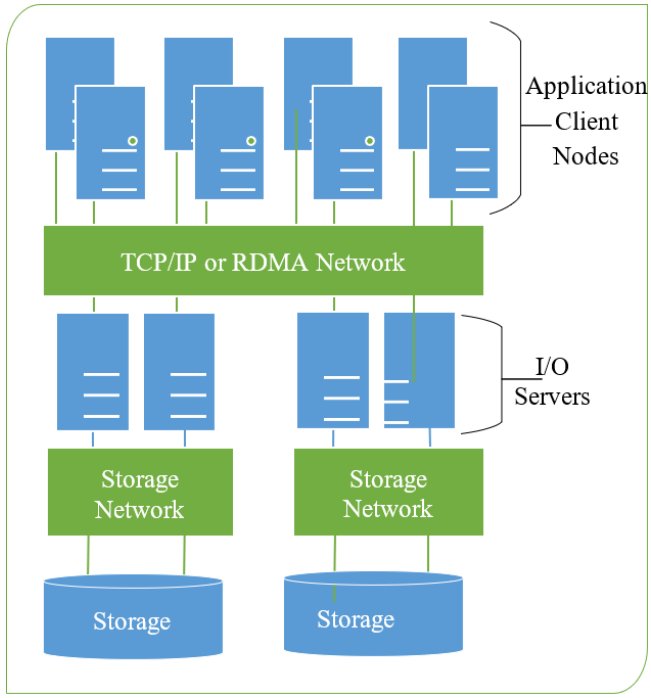

Figure 1. Network Attached Storage (NAS) configuration

3.2 Asynchronous disaster recovery

IBM Storage Scale's ADR, built upon the Active File Management (AFM) framework [15], provides a scalable and high-throughput mechanism for replicating data between geographically distributed file systems. ADR operates at either the entire file system level or at fileset granularity [16], where a fileset represents a logical partition of the file system namespace with independent management policies. This flexibility enables organizations to implement selective replication strategies based on data criticality and recovery requirements.

The ADR architecture employs a gateway-based replication model designed to minimize performance impact on production workloads. In this model, each node within the Primary file system cluster has direct local access to shared storage, enabling parallel applications to perform concurrent read and write data operations across multiple nodes without coordination overhead. Application nodes at the Primary site communicate with one or more designated gateway nodes that serve as replication coordinators. When applications perform I/O operations (create, write, delete, rename, etc.), these operations are captured by the file system and forwarded to the gateway nodes using lightweight remote procedure calls (RPC) as illustrated in Figure 2. Critically, application I/O operations complete immediately after local persistence, without waiting for remote replication acknowledgment—this

asynchronous design ensures that replication latency does not impact application performance.

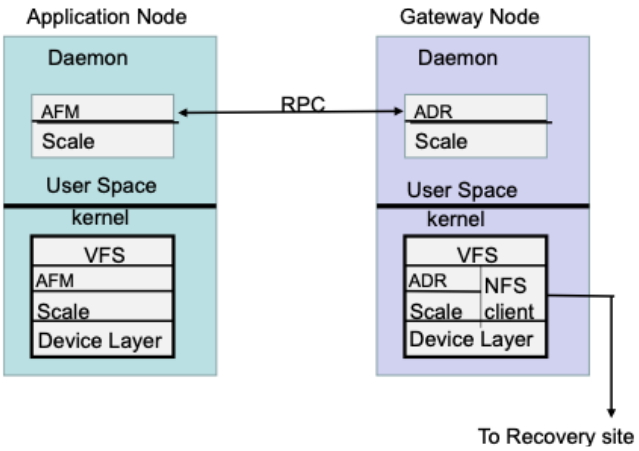

Figure 2. Replication data to recovery file system

The gateway nodes maintain operation queues and perform background data transfer to the Recovery file system [17]. To optimize throughput and accommodate diverse deployment scenarios, ADR supports multiple replication protocols:

- pNFS (Parallel NFS) [18]: Enables parallel data transfer across multiple gateway nodes, maximizing bandwidth utilization for large-scale replication.
- NFSv4: Provides standards-based replication with broad compatibility.
- Aspera FASP [19]: Delivers high-speed transfer over high-latency WAN links using UDP-based transport with forward error correction.
- S3 protocol [20]: Enables replication to cloud-based object storage targets (e.g., AWS S3, IBM Cloud Object Storage) with encryption and authentication.

The choice of protocol depends on factors including network characteristics, security requirements, and target infrastructure capabilities.

Because replication operates asynchronously, the Recovery file system may lag behind the Primary site by seconds to minutes, depending on workload intensity, network bandwidth, and geographic distance. This temporal lag introduces a consistency challenge: at any given moment, the Recovery file system may contain a partially replicated, inconsistent state. To address this challenge, ADR implements a snapshot-based consistency model aligned with the organization's RPO.

Operationally, snapshots are created periodically on the Primary file system at intervals matching the RPO requirement (e.g., every 15 minutes). Each snapshot captures a crash-consistent point-in-time state of the file system. The ADR system tracks which data blocks and metadata operations belong to each snapshot and ensures their complete transfer to the Recovery site. Only after all data associated with a Primary snapshot has been successfully replicated does the system create a corresponding peer snapshot on the Recovery file system.

These paired peer snapshots represent synchronized, crash-consistent recovery points across both sites. In the event of a disaster, the Recovery file system can be restored to the most recent peer snapshot, guaranteeing consistency within the defined RPO window.

Maintaining RPO compliance requires careful capacity planning. If replication throughput cannot keep pace with the rate of change at the Primary site, the system may fail to complete snapshot replication within the RPO interval, resulting in RPO violations. For example, if the Primary site generates 10 GB of changes every 15 minutes but available network bandwidth only supports 5 GB per 15 minutes, peer snapshots will lag increasingly behind, violating the RPO.

Prior researches [21, 22] have proposed techniques for generating near–real-time consistent snapshots in asynchronous replication environments, including adaptive snapshot scheduling and priority-based replication queuing, helping to mitigate RPO violation risks under variable workload conditions.

3.3 Failover to recovery file system

When a disaster renders the Primary file system unavailable—due to site-wide outages, storage failures, network partitions, or data corruption—a failover procedure is initiated to transition application workloads to the Recovery file system. This process involves several critical steps to ensure data consistency and minimize service disruption.

(1) Disaster Detection: Monitoring systems detect Primary site unavailability through health checks, heartbeat failures, or explicit administrator intervention.
(2) Consistency Point Selection: The most recent peer snapshot (failover snapshot) at the Recovery file system is identified. This snapshot represents the last fully synchronized, crash-consistent state between the Primary and Recovery sites.
(3) Recovery file system Activation: The Recovery file system is restored to the failover snapshot state, discarding any partially replicated, inconsistent data that arrived after the snapshot was created. This ensures applications access only consistent data.
(4) Application Redirection: Applications are reconfigured to access the Recovery file system, typically through DNS updates, load balancer reconfiguration, or application-level failover mechanisms.

The time required to complete this process defines the Recovery Time Objective (RTO)—the maximum acceptable duration of service unavailability. Modern implementations typically achieve RTOs of 5-15 minutes, though this varies based on snapshot size, storage performance, and automation sophistication.

Despite using the failover snapshot to ensure consistency, some data loss is inevitable in asynchronous replication scenarios. The magnitude of loss depends on two factors:

(1) RPO Interval: The time between peer snapshot creation (e.g., 15 minutes).
(2) Disaster Timing: When the disaster occurs relative to the last completed peer snapshot.

In the worst case, if a disaster occurs immediately before the next scheduled peer snapshot, data loss approaches the full RPO interval. For example, with a 15-minute RPO, up to 15 minutes of recent updates may be lost.

After failover, the Recovery file system operates as the sole production system without a backup DR site—the Primary file system remains offline and out of sync. This creates a critical vulnerability window: if a second disaster affects the Recovery site during this period, applications lose access to data entirely with no fallback option.

This vulnerability is particularly concerning for mission-critical applications with strict availability requirements. For example, in financial services or healthcare environments,

even brief periods without DR protection may violate regulatory compliance requirements or expose organizations to unacceptable business risk.

To restore DR protection, the Primary file system must be repaired and returned to service. However, simply bringing the Primary file system back online is insufficient—it must first be synchronized with all data changes that occurred at the Recovery site during the outage period. This synchronization process, known as failback, presents significant technical challenges:

(1) Data Volume: Depending on outage duration and workload intensity, the Recovery site may have accumulated substantial changes (potentially terabytes).
(2) Consistency Requirements: Synchronization must ensure potentially no data loss and maintain crash consistency.
(3) Downtime Constraints: Traditional failback approaches require quiescing applications at the Recovery site during final synchronization, resulting in service disruption.

Conventional snapdiff-based failback procedures (as described in Section 2.3) address consistency requirements but suffer from prolonged downtime during the final synchronization phase. For large-scale deployments with high write throughput, this downtime may extend to hours, violating SLA commitments and impacting business operations.

To address these limitations, this paper presents an innovative procedure that re-establishes the DR relationship between the recovered Primary file system and the active Recovery file system without requiring longer application downtime. The approach leverages a temporary reverse DR configuration that enables continuous synchronization while applications remain operational at the Recovery site. Once synchronization completes, applications can be seamlessly transitioned back to the Primary file system with minimal disruption (typically < 1 minute), and the standard DR topology is restored. This method significantly reduces the vulnerability window and enables rapid restoration of full DR protection.

4. ROLE-REVERSED DISASTER RECOVERY FOR FAILBACK

When the Primary file system becomes available after a failure, normal operation should ideally be restored by migrating application workloads from the Recovery file system back to the Primary file system. However, because all updates generated during outage are serviced by Recovery file system, failback cannot be initiated immediately. The recovered Primary file system must first be synchronized with the Recovery file system so that it reflects the most recent consistent state. This synchronization step is essential to prevent data loss and to ensure correctness during the transition back to the original production configuration.

The proposed method achieves this objective through a temporary role reversal of the DR relationship. Instead of immediately reinstating the original primary-to-recovery replication direction, the system first establishes a reverse DR relationship from the Recovery file system to the Primary file system. In this temporary configuration, the Recovery file system continues to operate as the active production system, while the Primary file system temporarily assumes the role of the replication target. This design allows updates accumulated at the recovery site during the outage period to be propagated back to the recovered primary file system without interrupting application service.

The role-reversed failback procedure consists of the following four stages described in Sections 4.1 through 4.4.

4.1 Establishment of the reverse disaster recovery relationship

The procedure begins with the creation of a dedicated failback snapshot on the Recovery file system. This snapshot serves as the reference point for identifying all modifications that must be propagated back to the Primary file system. After the snapshot is created, a DR relationship is established in the reverse direction, from the Recovery file system to the Primary file system, as illustrated in Figure 3. In effect, the recovered Primary file system becomes the temporary recovery target for duration of the synchronization phase.

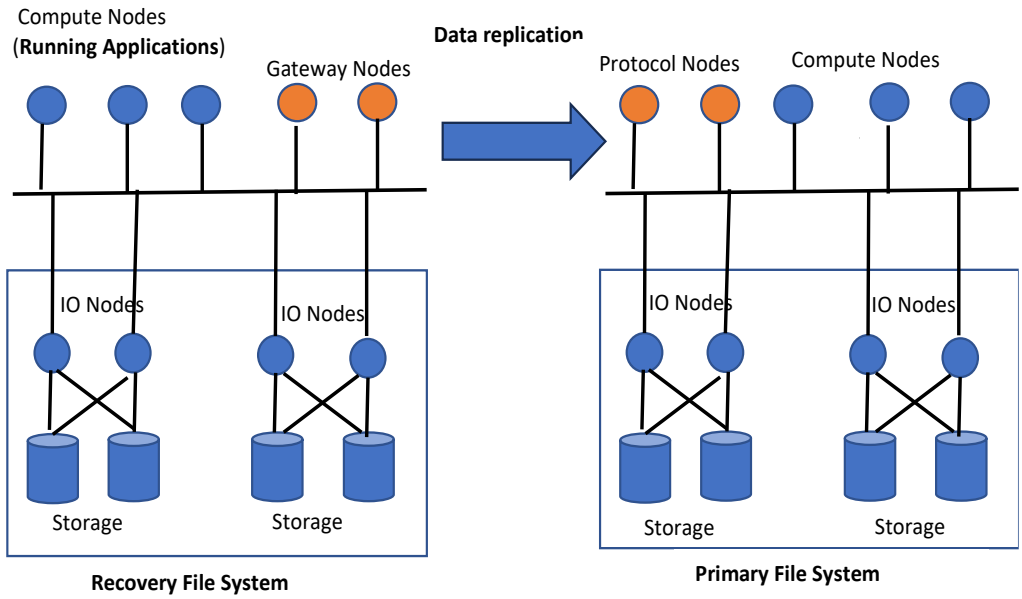

Figure 3. Role-reversed disaster recovery (DR) relation

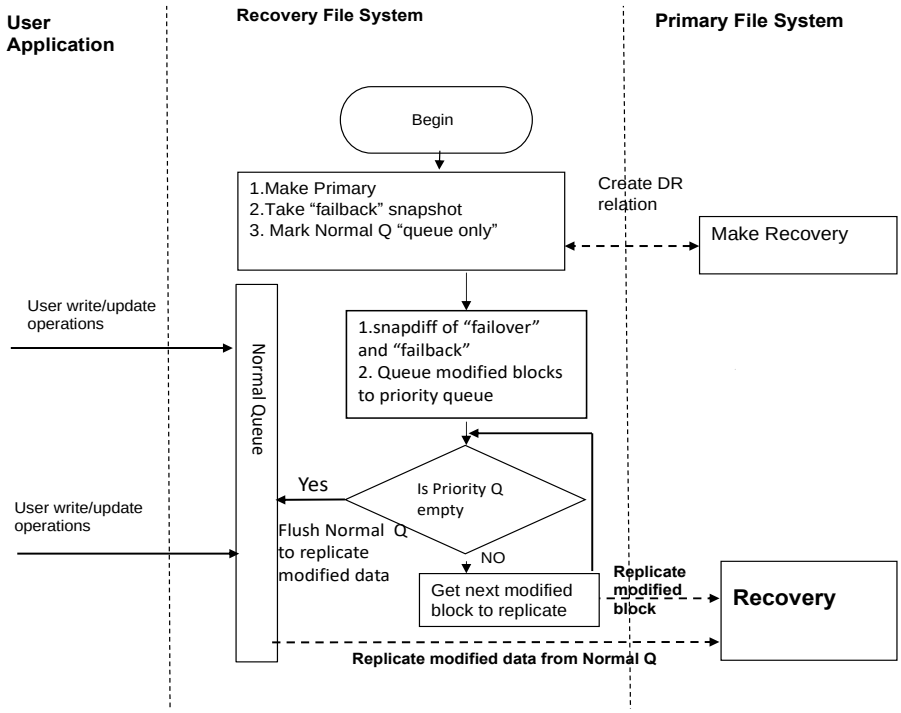

Figure 4. Synchronizing the primary file system with recovery file system updates

As illustrated in Figure 4, once the reverse DR relationship is active, the normal replication queue on the Recovery file system is temporarily paused. This pause prevents newly generated updates from interleaving with synchronization traffic, associated with failback. Importantly, pausing the normal queue does not interrupt foreground application activity. The Recovery file system continues to serve application I/O while synchronization proceeds in the background.

Any new updates generated during this interval are buffered in the normal replication queue and are not transmitted immediately. Their propagation is deferred until the initial synchronization phase has completed. This separation between synchronization traffic and newly generated updates ensures that the recovered Primary file system first receives a well-defined and consistent set of changes corresponding to the outage period.

4.2 Background synchronization of the primary file system

Synchronization is driven by comparing the failback snapshot created on the Recovery file system with the most recent peer-consistent failover snapshot previously established across the primary and recovery file systems. The differences between these snapshots are identified using a snapdiff-based mechanism. The resulting set of modified data blocks represents the updates accumulated while the Primary file system was unavailable.

These modified blocks are inserted into a priority queue that is processed ahead of the normal replication queue. Each priority-queue entry is transmitted asynchronously to the Primary file system in the background. Because this transfer occurs asynchronously, the Recovery file system can continue serving production workloads with minimal performance disruption.

After all priority queue entries have been processed, the normal replication queue is resumed. At this point, all updates buffered during synchronization interval are propagated to the Primary file system in the usual asynchronous manner. Periodic RPO snapshots continue to be maintained throughout this phase so that consistency guarantees remain intact even while the reverse DR relationship is active.

This design shifts most of the failback-related data transfer to a background synchronization phase. As a result, the amount of work that must be completed during the final application cutover is substantially reduced.

4.3 Optional retention of the reversed configuration

Immediate migration of application workloads back to the original Primary file system may not always be necessary. In some deployment scenarios, the Recovery file system may continue to provide adequate performance and capacity, making it acceptable to remain as the active production system for an extended period.

In such cases, the reversed DR configuration can be retained. Operationally, this means that the Recovery file system is effectively promoted to the role of Primary file system, while the original Primary file system assumes the role of the Recovery file system.

The reversed configuration may remain in effect until operational conditions, such as workload intensity, throughput requirements, maintenance planning, or administrative policy, justify restoration of the original DR direction. This flexibility reduces operational disruption and allows failback timing to be aligned with business requirements rather than being forced immediately after recovery of the original primary site.

4.4 Final failback to primary file system

Once the Primary file system has been substantially synchronized with the Recovery file system and the system activity has decreased to an acceptable level, the final failback step can be initiated, as illustrated in Figure 5. This step transfers active application workloads from the Recovery file system back to the Primary file system.

To complete failback safely, application activity on the Recovery file system is briefly paused. During this short interval, any remaining buffered updates are flushed to the Primary file system so that synchronization becomes complete. After all outstanding updates have been transmitted, a peer-consistent snapshot, denoted as psnap0, is created to establish a stable and consistent checkpoint across both systems.

Once psnap0 has been successfully created on the Primary file system, applications can be redirected back to the primary site with minimal disruption. Because the bulk of synchronization has already been completed in the background, the duration of this final cutover phase is short compared with conventional failback procedures.

After applications resume on the Primary file system, the standard DR configuration is reinstated by re-establishing the replication relationship from the Primary file system to the Recovery file system. In this restored configuration, the Primary file system resumes its role as the active production system, while the Recovery file system returns to its standby role. This transition restores the original operational topology that existed before the failure event, as illustrated in Figure 6.

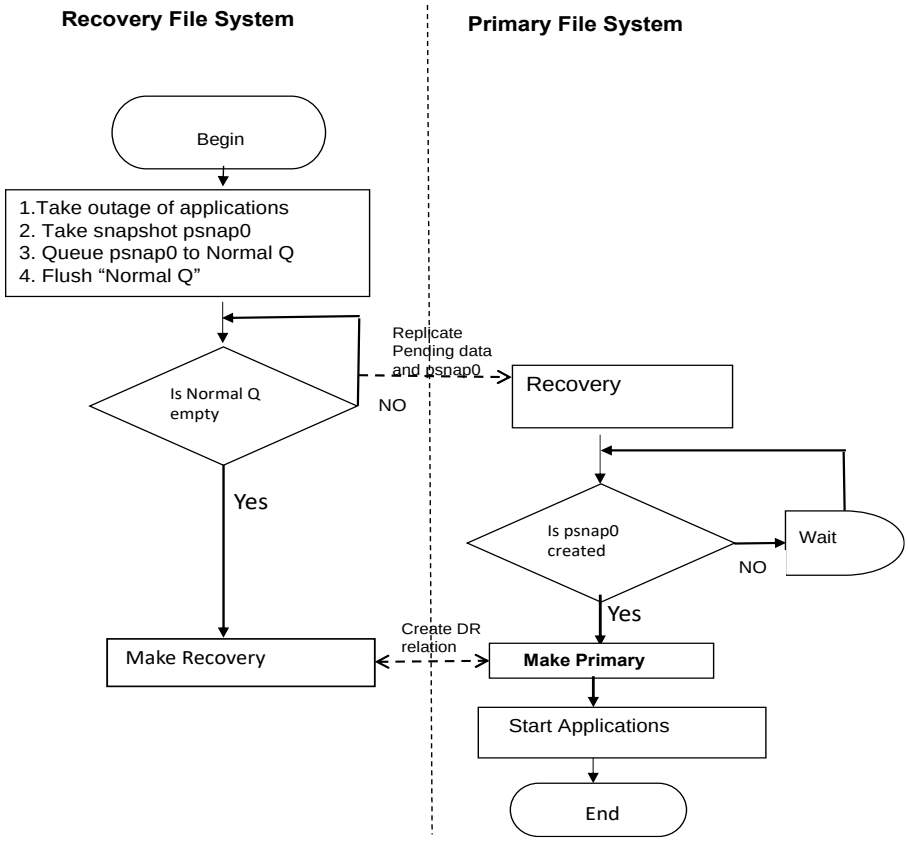

Figure 5. Failback to primary file system

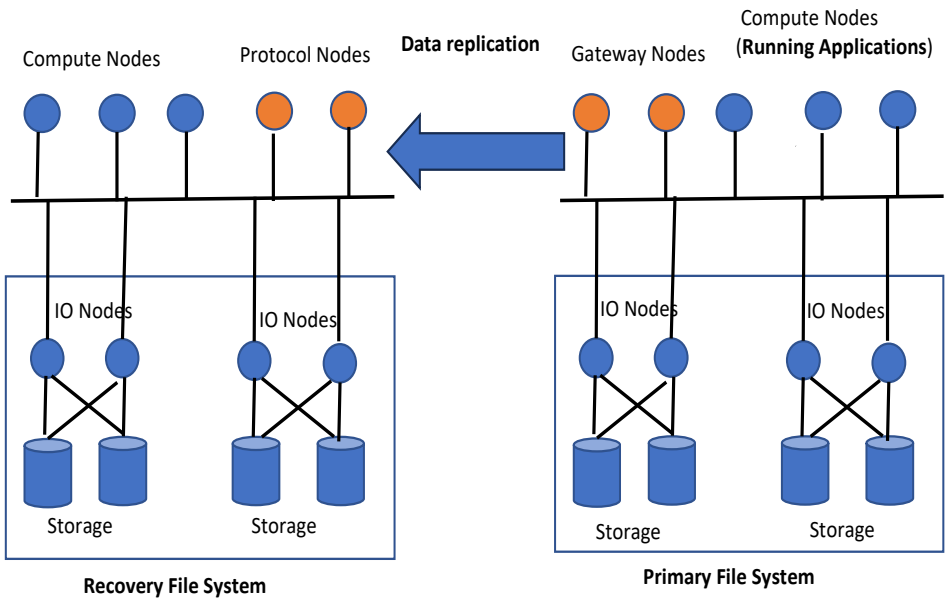

Figure 6. Reestablishing disaster recovery (DR) relations between the primary file system and Recovery file system

Overall, the proposed role-reversed DR procedure enables efficient failback in distributed replication environments affected by either planned or unplanned disruptions. By moving most synchronization work to an asynchronous background phase, the method substantially reduces application downtime during failback. Consequently, the Primary file system can be returned to service with minimal interruption, while full DR protection is restored in a timely and operationally efficient manner.

4.5 Security and reliability considerations

The proposed role-reversal DR failback mechanism is designed to minimize application downtime while maintaining data consistency and operational reliability throughout the recovery process. This mechanism leverages IBM Spectrum Scale AFM replication and snapshot capabilities to ensure secure and reliable failback execution.

4.5.1 Data integrity and consistency

Data integrity and consistency during the failback process are ensured through the coordinated operation of two complementary mechanisms: snapshot differential-based replication and asynchronous DR replication. Initially, snapshot differential-based replication identifies and transfers only the data blocks modified between the failover snapshot and the failback snapshot on the Recovery file system. This synchronization phase propagates all updates accumulated while the Primary file system was unavailable, thereby restoring data consistency between the two sites with minimal transfer overhead.

Following the completion of differential synchronization, asynchronous DR replication continues to transmit newly generated updates from the Recovery file system to the Primary file system. This replication framework preserves write ordering and consistency semantics across both file systems, ensuring that ongoing application activity remains protected throughout the failback operation. Prior to final role restoration, peer snapshots are created on both the Recovery and Primary file systems to establish a common synchronization checkpoint. These snapshots serve as a consistent recovery reference, enabling verification that all updates accumulated at the Recovery site have been successfully propagated and committed to the Primary site.

4.5.2 Secure replication communication

The proposed mechanism preserves the security model of the underlying replication infrastructure without introducing additional vulnerabilities. Replication traffic between the Recovery and Primary sites can be protected using industry-standard encrypted communication channels, including TLS-secured connections, IPsec VPN tunnels, or dedicated private network links. These security controls protect replicated data from unauthorized interception, tampering, or modification during transmission, ensuring confidentiality and integrity of data in transit.

4.5.3 Access control and authorization

Role-reversal operations require administrative authorization on both participating systems, enforcing the principle of least privilege. Existing authentication and access-control mechanisms, including role-based access control (RBAC) and multi-factor authentication where applicable, remain in effect throughout the failback process. This ensures that only authorized administrators with appropriate credentials can establish reverse replication relationships, initiate failback procedures, or modify replication configurations.

4.5.4 Audit logging and traceability

All operational activities associated with failback, including replication state transitions, role reversals, snapshot creation, synchronization completion events, and administrative actions, are recorded through standard storage-system and operating-system logging facilities. These comprehensive audit records provide complete traceability for troubleshooting, compliance verification, forensic analysis, and post-recovery assessment. The audit trail enables administrators to reconstruct the sequence of events during failback operations and verify adherence to organizational policies and regulatory requirements.

4.5.5 Rollback and recovery protection

To mitigate operational risks and provide fault tolerance, the proposed mechanism retains synchronization checkpoints prior to final failback completion. If validation procedures detect inconsistencies or unexpected failures occur during failback execution, administrators can revert to the last verified recovery state using previously created snapshots and reinitiate the synchronization process. This rollback capability minimizes the risk of data loss, provides a controlled recovery path, and ensures that the system can be restored to a known-good state without compromising data integrity. The checkpoint-based approach enables graceful degradation and recovery from transient failures without requiring complete failback restart.

4.5.6 Summary

The proposed role-reversal failback mechanism preserves the integrity, confidentiality, and recoverability of replicated data while significantly reducing application downtime compared with conventional snapshot-based failback approaches. By integrating security controls at multiple layers—data protection, communication security, access control, audit logging, and recovery safeguards—the mechanism provides a comprehensive security posture suitable for enterprise-grade DR deployments. The design ensures that security and reliability considerations are not compromised in pursuit of reduced application downtime, thereby meeting both operational efficiency and data protection requirements.

5. RESULTS AND DISCUSSIONS

This section evaluates the effectiveness of the proposed role-revered DR failback mechanism and compares it with conventional snapdiff-based failback approach. The evaluation focuses on failback delay, and application downtime under varying workload characteristics. Two distinct experimental scenarios are investigated: (i) creation of new files, and (ii) modification of existing files. The primary objective is to determine whether the proposed method can reduce service interruption while maintaining efficient synchronization of updates accumulated at the recovery site.

5.1 Experimental setup

To evaluate the proposed mechanism, we deployed a

Primary file system and a remote Recovery file system using IBM Storage Scale. Each system was configured on a single node running Red Hat Enterprise Linux with directly attached HDD storage. The two nodes were interconnected through a TCP/IP local-area network.

The hardware configuration of each server is as follows:

(1) CPU: AMD EPYC 7542 32-Core Processor operating at 2.694 GHz
(2) Memory: 256 GB DDR4 RAM
(3) Storage: 1 TB SAS HDD
(4) Network: 20 Gbps Ethernet connection between sites
(5) Operating System: Red Hat Enterprise Linux 8.6
(6) IBM Storage Scale Version: 5.1.5
(7) Number of Storage Scale Nodes: 1 per site
(8) Snapshot Interval: On-demand snapshots created before failover and failback operations

Although this testbed does not fully capture the scale of a production multi-node deployment, it provides a controlled environment for evaluating the relative performance of different failback mechanisms under identical hardware and network conditions.

A fileset was created on both the Primary and Recovery file systems, and asynchronous replication was configured between them. Two workload scenarios were designed to represent common DR use cases:

(1) New files Creation: Synthetic datasets consisting of uniformly sized 2 MB files were generated to emulate workloads dominated by file creation operations.
(2) Existing files modification: Existing files were modified to emulate workloads characterized primarily by in-place file updates.

Experiments were conducted across multiple workload scales to assess scalability corresponding to low, moderate, and high I/O pressure. These scenarios were intended to evaluate the scalability of the failback procedure under increasing metadata and data synchronization requirements:

The evaluation compares two failback strategies:

(1) Baseline method: Conventional snapdiff-based failback.
(2) Proposed method: Role-reversed DR failback mechanism introduced in Section 4.

The primary performance metric considered in this study is application downtime during failback, defined as the duration for which application services are unavailable to end users during the restoration of operations to the Primary site. To further characterize failback performance, two additional metrics are evaluated: metadata scan time, representing the time required to identify modified files or data blocks, and synchronization transfer time, representing the duration required to propagate the identified changes between sites. These metrics provide insight into the major contributors to overall failback latency.

For the conventional snapdiff-based failback approach, application services are suspended prior to initiating snapshot comparison and differential analysis. As a result, the total application downtime encompasses both the metadata scanning phase and the subsequent data synchronization phase required to restore consistency between the Recovery and Primary file systems.

In contrast, the proposed role-reversal DR failback mechanism performs reverse synchronization asynchronously while applications continue to operate on the Recovery file system. Consequently, the metadata scanning and bulk synchronization activities are removed from the application downtime window. Application interruption is restricted to the final consistency phase, during which application I/O is temporarily quiesced, synchronized peer snapshots are created on both sites, and the final cutover to the Primary file system is executed. This design substantially reduces service disruption by limiting downtime to a brief consistency-verification and transition interval, thereby improving application availability during failback operations.

5.2 New file creation workload

5.2.1 Baseline: Snapdiff-based failback

In the baseline approach, failback is performed using a snapdiff-based synchronization procedure. To ensure a consistent state, applications running on the Recovery system are first stopped. A new snapshot is then created and compared with a reference snapshot (failover snapshot) to identify newly created and modified data blocks. The identified changes are subsequently replicated to the Primary file system.

The baseline procedure consists of two main phases:

(1) **Metadata Scan Phase** – Detection of modified and newly created files through snapshot comparison.
(2) **Data Transfer Phase** – Replication of the identified changes to the Primary file system.

The total failback time is measured as the interval from application shutdown at the Recovery until completion of synchronization at the Primary site. Table 1 summarizes the observed results.

The results indicate that failback duration increases significantly with workload size. For large scale file systems the metadata scan phase becomes a primary bottleneck, while disk and network throughput further contribute to increased synchronization time, particularly as data volume grows.

Table 1. Snapdiff-based failback performance for new file creation workload

Number of Files	Data Volume	Scan Time (s)	Transfer Time (s)	Application Downtime (s)
1500	3 GB	135	22	157
3000	6 GB	236	39	275
4500	9 GB	435	53	488
6000	12 GB	556	79	635
7500	15 GB	756	104	275

5.2.2 Proposed method: Role-reversal disaster recovery failback

To overcome the limitations of the baseline approach, we implemented a reverse DR configuration in which the Recovery file system temporarily acts as the source for synchronization. Updates accumulated at the Recovery site are transferred asynchronously to the Primary file system while applications continue to execute.

The synchronization process operates using two queues:

(1) Data modified on the Recovery file system prior to establishing the reverse relationship is transferred asynchronously using a high priority queue.
(2) New updates generated during ongoing application execution are placed in a normal queue and propagated after the initial backlog is processed.

By performing synchronization in the background, most of the data transfer occurs concurrently with application processing. Application interruption is required only during the final consistency operation involving peer snapshot creation.

The results, summarised in Table 2, reveal that most of the data synchronization is performed in the background without impacting application availability. The final phase—where applications are briefly paused and a peer snapshot (psnap0) is created—remains nearly constant across all workloads. This presents a reduction in the application downtime of approximately 98% compared to the baseline approach for the largest workload tested.

Table 2. Role-reversed disaster recovery (DR) failback performance for new file creation workload

Number of Files	Data Volume	Final Sync Times (s)	Total Application Downtime (s)
1500	3 GB	3	3
3000	6 GB	3	3
4500	9 GB	4	4
6000	12 GB	4	4
7500	15 GB	4	4

5.3 Files modification workload

5.3.1 Baseline: Snapdiff-based failback

In this experiment, approximately 7,500 files of size 2 MB were initially created on the Primary file system and replicated to the Recovery file system. Following failover, application workloads modified varying amounts of data on the Recovery file system. Subsequently, failback was performed using the conventional snapdiff-based approach. In this approach, a failback snapshot was compared against the failover snapshot to identify modified files, after which the changes were synchronized to the Primary file system.

The Table 3 summarizes the observed performance results. Unlike the file creation workload, metadata scan time remains relatively constant because the total number of files remains unchanged. Nevertheless, the metadata comparison phase continues to dominate overall failback time, resulting in significant application downtime.

Table 3. Snapdiff-based failback performance for file modification workload

Number of Files	Updated Data Volume	Scan Time (s)	Transfer Time (s)	Application Downtime (s)
7500	3 GB	750	22	772
7500	6 GB	754	39	793
7500	9 GB	752	53	805
7500	12 GB	750	79	829
7500	15 GB	756	103	859

5.3.2 Proposed method: Role-reversal disaster recovery failback

The file modification experiments were repeated using the proposed role-reversal DR mechanism. Like the file creation scenario, updates were transferred asynchronously while applications remained operational.

The results are consistent with those obtained for the file creation workload. Application downtime remains nearly constant, ranging from 4 to 5 seconds across all evaluated data volumes. This observation indicates that the proposed role-reversal DR mechanism effectively overlaps data synchronization with normal application execution, thereby confining service interruption to a brief final consistency phase. The negligible variation in downtime across workload sizes further suggests that the failback process scales independently of the amount of modified data, as evidenced by the results presented in Table 4.

Table 4. Role-reversal disaster recovery (DR) failback performance for file modification workload

Number of Files	Updated Data Volume	Final Sync Time (s)	Application Downtime (s)
7500	3 GB	4	4
7500	6 GB	4	4
7500	9 GB	4	4
7500	12 GB	5	5
7500	15 GB	4	4

5.4 Comparative analysis

As shown in Figure 7, the failback delay of the SnapDiff-based approach increases significantly with workload size. This increase is primarily caused by the growing metadata scan overhead required to identify newly created files, together with the subsequent data transfer time. In contrast, the proposed role-reversal DR mechanism maintains a nearly constant application downtime of only 3–4 seconds across all evaluated workloads.

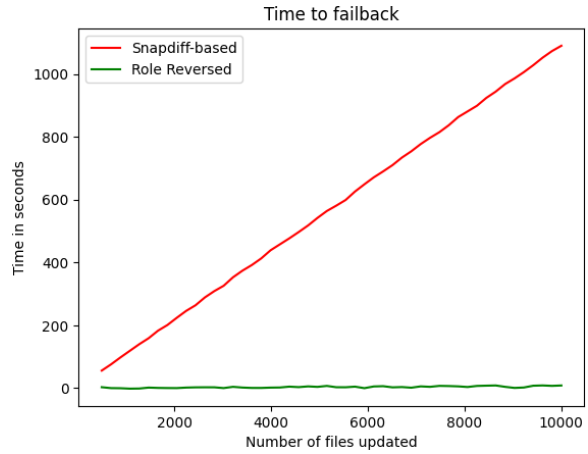

Figure 7. Compares the failback delay of the conventional snapdiff-based approach with that of the proposed role-reversal disaster recovery (DR) mechanism for the new file creation workload

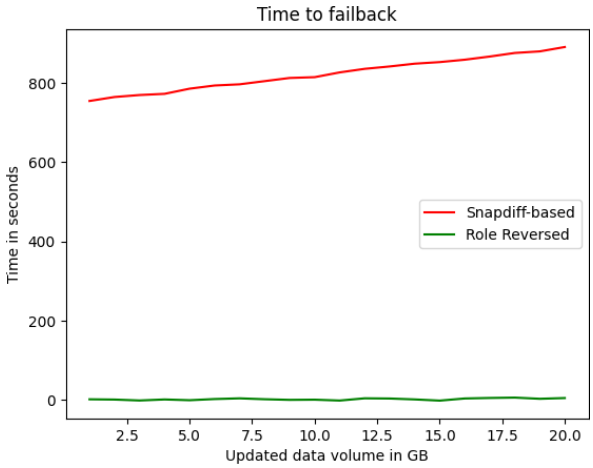

Figure 8. Compares the failback delay of the conventional snapdiff-based approach with that of the proposed role-reversal disaster recovery (DR) mechanism for the files update workload

This improvement is achieved by performing most of the synchronization asynchronously while applications continue executing on the Recovery system. Consequently, application interruption is limited to a brief final consistency operation. A similar trend is observed in the file modification experiments presented in Figure 8.

Overall, the proposed mechanism reduces application downtime by approximately 98–99.5% compared with the conventional snapdiff-based approach, demonstrating superior scalability and significantly improved service availability during DR failback operations.

5.5 Key observations

The experimental evaluation leads to the following observations:

(1) The conventional snapdiff-based failback procedure introduces substantial application downtime, particularly for large-scale workloads.
(2) Metadata scanning constitutes the dominant component of failback delay and becomes increasingly expensive as the number of files grows.
(3) The proposed role-reversal DR mechanism eliminates lengthy synchronization windows by performing replication asynchronously during normal application execution.
(4) Application downtime is reduced from several minutes to only a few seconds, representing an improvement of more than 99% for the largest workloads evaluated.
(5) The proposed approach exhibits excellent scalability, with failback downtime remaining largely independent of workload size and data volume.
(6) The results demonstrate that role-reversal DR can significantly improve recovery-site service continuity and is therefore well suited for large-scale enterprise DR environments.

5.6 Failure boundary and limitations

The proposed role-reversal DR failback mechanism assumes that the Recovery file system remains operational throughout the reverse synchronization phase and that communication between the Recovery and Primary sites remains available to support replication activities. While the mechanism substantially reduces application downtime, its effectiveness depends on several operational assumptions.

5.6.1 Network partition

If network connectivity between the Recovery and Primary sites is interrupted during reverse synchronization, data replication is temporarily suspended. Once connectivity is restored, synchronization can resume from the last successfully replicated update. However, failback completion cannot occur until communication between both sites is re-established.

5.6.2 Recovery site failure during failback

The proposed mechanism assumes that the Recovery file system remains available until reverse synchronization is completed. A catastrophic failure of the Recovery site during failback may result in loss of unsynchronized updates that have not yet been transferred to the Primary file system. In such circumstances, failback cannot be completed until the Recovery site is restored or an alternative recovery mechanism is invoked.

5.6.3 Corrupted, ransomware or malicious data

The proposed mechanism transfers updates generated at the Recovery file system but does not independently validate application-level data correctness. Similarly it does not distinguish between legitimate updates and malicious modifications. Consequently, corrupted, ransomware-encrypted, or maliciously modified data may be propagated to the Primary file system during reverse synchronization. Recovery from such events requires additional protection mechanisms such as malware detection systems, or point-in-time recovery procedures.

5.6.4 Administrative and configuration errors

Incorrect configuration of replication relationships, accidental deletion of snapshots, or improper failback execution may affect recovery operations. The use of administrative authorization controls, audit logging, and pre-failback validation procedures can reduce the likelihood of such errors.

5.6.5 Summary

The proposed role-reversal failback mechanism primarily addresses the challenge of reducing application downtime and improving service availability during failback operations. While it provides efficient synchronization and consistency preservation between the Recovery and Primary sites, it does not explicitly address advanced fault scenarios such as simultaneous site failures, malicious data modification, ransomware attacks, or human operational errors. Mitigation of these risks requires complementary security, validation, and resilience mechanisms that operate in conjunction with the proposed failback framework. These aspects represent important directions for future research.

6. CONCLUSIONS

Our observations indicate that using role-reversed DR relations to failback applications from the Recovery file system to the Primary file system results in significantly lower application downtime. Importantly, this downtime remains unaffected by the volume of data changes on the Recovery file system during the Primary file system's downtime. In contrast, traditional snapdiff-based failback procedures experience increased application downtime proportional to the data changes made on the Recovery file system while the Primary file system is unavailable. Role-reversed DR relations offer greater flexibility, as immediate failback is not necessary. The Primary file system operates as a recovery system, maintaining recoverability. This allows failback to be scheduled during periods of low system load, optimizing performance. Conversely, snapdiff-based failback requires prompt action once the Primary file system is restored to ensure system recoverability. Any delay in this process risks compromising recoverability in the event of a disaster affecting the Recovery file system where applications are running.

REFERENCES

[1] Shen, Z., Cai, Y., Cheng, K., Lee, P.P.C., Li, X., Hu, Y., Shu, J. (2025). A survey of the past, present, and future

of erasure coding for storage systems. ACM Transactions on Storage, 21(1): 1-39. https://doi.org/10.1145/3708994

[2] Deshpande, U., Linck, N., Seshadri, S. (2021). Self-service data protection for stateful containers. In Proceedings of the 13th ACM Workshop on Hot Topics in Storage and File Systems, Virtual, USA, pp. 71-76. https://doi.org/10.1145/3465332.3470876

[3] Mendonça, J., Lima, R., Queiroz, E., Andrade, E., Kim, D.S. (2019). Evaluation of a backup-as-a-service environment for disaster recovery. In 2019 IEEE Symposium on Computers and Communications (ISCC), Barcelona, Spain, pp. 1-6. https://doi.org/10.1109/ISCC47284.2019.8969658

[4] Depoutovitch, A., Chen, C., Chen, J., Larson, P., et al. (2020). Taurus database: How to be fast, available, and frugal in the cloud. In Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data, Portland, USA, pp. 1463-1478. https://doi.org/10.1145/3318464.3386129

[5] Shumway, S. (1991). Issues in online backup. In Proceedings of the Fifth Large Installation Systems Administration Conference, Berkeley, USA, p. 81. http://98.124.61.90/pub/mouse/misc/shumway-online-backup.ps.

[6] Xiao, W., Yang, Q., Ren, J., Xie, C., Li, H. (2009). Design and analysis of block-level snapshots for data protection and recovery. IEEE Transactions on Computers, 58(12): 1615-1625. https://doi.org/10.1109/TC.2009.107

[7] Chervenak, A., Vellanki, V., Kurmas, Z. (1998). Protecting file systems: A survey of backup techniques. In Joint NASA and IEEE Mass Storage Conference, pp. 17-32.

[8] Meng, K., Li, M., Ding, J., Zhou, H. (2026). Cloud disaster recovery model based on failure prediction. Journal of Cloud Computing, 15(1): 33. https://doi.org/10.1186/s13677-026-00846-0

[9] Takdir, Kitagawa, H., Amagasa, T. (2025). Local recovery and partial snapshot in distributed stateful stream processing. Knowledge and Information Systems, 67(10): 9407-9435. https://doi.org/10.1007/s10115-025-02509-z

[10] Karni, S. (2025). Seamless low-cost, low-maintenance DR orchestration with Azure. International Journal of Computational and Experimental Science and Engineering, 11(3). https://doi.org/10.22399/ijcesen.3777

[11] Liu, J., Dai, Y., Arpaci-Dusseau, A.C., Arpaci-Dusseau, R.H. (2025). Fast, transparent filesystem microkernel recovery with Ananke. In Proceedings of the 23rd USENIX Conference on File and Storage Technologies, Santa Clara, USA, pp. 1-18. https://www.usenix.org/conference/fast25/presentation/liu-jing.

[12] Shen, W., Cui, Y., Sen, S., Angel, S., Mu, S. (2025). Mako: Speculative distributed transactions with geo-replication. In Proceedings of the 19th USENIX Symposium on Operating Systems Design and Implementation, Boston, USA, pp. 129-152. https://www.usenix.org/conference/osdi25/presentation/shen-weihai.

[13] Patterson, H., Manley, S., Federwisch, M., Hitz, D., Kleiman, S., Owara, S. (2002). SnapMirror: File system based asynchronous mirroring for disaster recovery. In Proceedings of the 1st USENIX Conference on File and Storage Technologies, Monterey, CA, p. 9. https://www.usenix.org/publications/library/proceedings/fast02/full_papers/patterson/patterson.pdf.

[14] Schmuck, F., Haskin, R. (2002). GPFS: A shared-disk file system for large computing clusters. In Proceedings of the 1st USENIX Conference on File and Storage Technologies, Monterey, USA, p. 16. https://www.usenix.org/legacy/events/fast02/full_papers/schmuck/schmuck_html/.

[15] Eshel, M., Haskin, R.L., Hildebrand, D., Naik, M., Schmuck, F.B., Tewari, R. (2010). Panache: A parallel file system cache for global file access. In FAST'10: Proceedings of the 8th USENIX Conference on File and Storage Technologies, San Jose, USA, 10: 1-14. https://www.usenix.org/legacy/event/fast10/tech/full_papers/eshel.pdf.

[16] Ananthanarayanan, R., Eshel, M., Haskin, R., Naik, M., Schmuck, F., Tewari, R. (2008). Panache: A parallel WAN cache for clustered filesystems. ACM SIGOPS Operating Systems Review, 42(1): 48-53. https://doi.org/10.1145/1341312.1341322

[17] Wang, C., Li, Z., Ren, K. (2010). ARPRG: An asynchronous replication protocol with RPO guarantee. In 2010 2nd International Conference on Computer Engineering and Technology, Chengdu, China, pp. V1-611. https://doi.org/10.1109/ICCET.2010.5485935

[18] Carnegie Mellon University, Parallel Data Laboratory. (2004). Parallel NFS (pNFS). https://www.pdl.cmu.edu/pNFS/index.shtml.

[19] IBM Corporation. Aspera Transfer Software. https://www.ibm.com/products/aspera.

[20] Rajesh, P., Alam, M., Tahernezhadi, M., Kumar, T.R., Rajesh, V.P. (2020). Secure communication across the internet by encrypting the data using cryptography and image steganography. International Journal of Advanced Computer Science and Applications, 11(10).

[21] Redd, S.R., Varma, G.P.S., Rao, P.S. (2024). File system RPO snapshots in near real-time with asynchronous replication. International Journal of Intelligent Systems and Applications in Engineering, 12(3): 1971-1977.

[22] Naik, M.P., Sure, R.R. (2018). Snapshots at real time intervals on asynchronous data replication system (U.S. Patent No. 9,983,947). U.S. Patent and Trademark Office. https://patents.google.com/patent/US9983947B2/en.