SACJ 38(2) July 2026
Review Article

Filesystem Delayed Block Allocation for High-Performance Sequential Writes in Enterprise Storage Systems

Bhanuprakash Naidu Basani

Independent Researcher, USA

ABSTRACT
Delayed block allocation represents a filesystem optimization technique that addresses performance limitations inherent in traditional immediate allocation models by postponing physical block assignment until sufficient write pattern information becomes available. Enterprise storage systems deploying immediate allocation strategies experience severe fragmentation as concurrent append operations interleave spatially across available storage regions, producing data layouts where logically contiguous file regions scatter across hundreds or thousands of small extents. This fragmentation degradation manifests most acutely during backup operations, where multi-terabyte filesystem scans extending to twenty-four hours create operational constraints limiting backup frequency and elevating data loss risk. The article presents an efficient delayed allocation implementation within the Novell Storage Services filesystem that introduces a lightweight reservation mechanism using 8-byte in-memory counters to track logically committed but not yet physically allocated blocks. The implementation eliminates journal interaction during write operations, enabling reservation updates to complete at memory speeds without disk-based persistence overhead. Physical allocation occurs during controlled flush events triggered by anticipatory delays, explicit synchronization requests, or system resource conditions, allowing the allocator to assign large contiguous extents that preserve sequential locality. Performance evaluation demonstrates forty percent write throughput improvement through eliminated journal serialization overhead, up to two hundred percent read performance enhancement from reduced fragmentation, and backup duration reduction from approximately twenty-four hours to few-hour windows representing eighty to ninety percent improvement. The memory-only reservation approach preserves crash consistency guarantees through standard journal replay mechanisms without introducing new failure modes, as lost reservations represent uncommitted work discarded during recovery. The implementation demonstrates how optimized realization of established filesystem concepts produces substantial operational benefits in production enterprise deployments through careful attention to resource efficiency and architectural compatibility.

Keywords: Delayed Block Allocation, Filesystem Performance Optimization, Fragmentation Reduction, Enterprise Storage Systems, Write-Ahead Journaling.

Article history
Received: 02 June 2026
Accepted: 02 July 2026
Online: 20 July 2026

1. INTRODUCTION

Enterprise storage systems are facing increasingly stringent performance demands and diverse workload patterns, including transactional processing, analytical activities, and archival tasks. The filesystems in the modern world are the most important layer of abstraction between programs and hardware storage devices, and therefore, the allocation strategies of filesystems are the key to the overall efficiency of the systems and the effectiveness of their operation. The design decisions made as part of filesystem allocation subsystems directly impact the quality of the data layout that in turn

Basani, B.N. (2026). Filesystem Delayed Block Allocation for High-Performance Sequential Writes in Enterprise Storage Systems. *South African Computer Journal* 38(2), 52–65.

ISSN 1015-7999 (print) ISSN 2313-7835 (online)

has a ripple effect on the read performance, write throughput, and the operational attributes such as backup and disaster recovery operations.

Append-heavy workloads are an overwhelmingly popular access pattern in modern enterprise storage facilities. Systems that execute archival tasks, store backup repositories, keep logs of database transactions, run content repositories and execute log aggregation platforms will have sustained sequential write properties whereby data is continuously added to the system via append operations and not via random write or in-place updates. The provenance tracking systems which capture histories of data transformation and system operations produce continuous append streams as additional provenance records are appended to existing provenance logs, without necessarily updating previous entries in the provenance logs [1]. Such workloads generate files which grow ever larger and as time passes, new information is added at logical file boundaries and old information is left largely unaltered. The consistently chosen allocation decisions of such incremental append operations combine to define the final physical layout of data on the storage media, creating patterns which are then sustained throughout the filesystem lifecycle.

The conventional filesystem designs use immediate block allocation techniques where file blocks are allocated to physical disk blocks in the exact moment write requests are made. This method offers both easiness of implementation and predictability of behavior since each write request is handled by blocks that have been assigned and metadata that has been updated to reflect the new condition. Nonetheless, immediate allocation has a fundamental deficiency of information, which limits the quality of allocation. At the allocation point, the filesystem only knows the current write operation in its size, target file offset and the distribution of the free space. Future writes that may can logically extend the same file region in either milliseconds or seconds and is not visible to the allocation subsystem. Thus the allocator will be required to decide based on partial information regarding final access patterns and file growth curves without measuring based on overall workload behavior.

This information deficiency generates suboptimal allocation results in the append-dominated load cases where files expand in sequences of related write requests. When files grow by repeated additions to them, blocks are allocated by the filesystem in whatever free space there is at any given decision point without information about any further allocations which will lengthen the same file. Allocations of multiple applications that concurrently write to different files are spatially interspersed across available storage regions with each request being served separately by the allocator. When the file size is large, the logically contiguous file segments are physically discontinuous on storage devices over long periods of operation as the allocation decisions of separate times take blocks in separate regions of free space. The files which are supposed to occupy a continuous physical area are spread across hundreds or thousands of small extents spread across the entire storage volume and generate data layouts that are essentially orthogonal to the sequential access patterns of read operations.

Fragmentation is an obstacle to performance that is critical and impacts various operational aspects of enterprise-wide implementation. Where there is logically sequential data in physically scattered sets of small extents spread across storage media, read activity needs to have the ability to traverse discontinuous block addresses to acquire continuous logical ranges of bytes. Every jump between non-adjacent extents can impose seek overhead on mechanical disk storage or random access cost on solid-state devices, and interferes with streaming access patterns and efficacy. The degradation of the performance increases with the number of extents, because with each extent boundary, a potential discontinuity exists which, when maintained, cannot support streaming read operations by the storage subsystem.

The consequences of fragmentation are felt most keenly in enterprise environments when they are engaged in the process of backup operations which are among the vital operations functions in terms of safeguarding their data and disaster recovery preparedness. Full filesystem scans in which all allocated blocks are read to copy the data to secondary storage systems or backup media are

Basani, B.N. (2026). Filesystem Delayed Block Allocation for High-Performance Sequential Writes in Enterprise Storage Systems. *South African Computer Journal* 38(2), 52–65.

ISSN 1015-7999 (print) ISSN 2313-7835 (online)

typically done by the backup systems. Nevertheless, fragmentation results in the deterioration of the backup operations into random read sequences because the backup process traverses disconnected extents on storage media. Physical storage devices, especially mechanical hard drives, present significantly slower throughput under random access workloads than sequential operation modes because of the mechanical cost of having to reposition the read heads in the platter surfaces. Backups have been observed to take up to twenty four hours to finish on systems that use filesystems with instant allocation, providing a strong compatibility risk with business customers who deploy filesystems with large files that demand considerable period to be restored, posing severe operational restrictions to the backup frequency, recovery point objectives, and risk of data loss during the extensive backup execution.

Delayed block allocation offers a mechanism that can be used to overcome these underlying shortcomings by delaying physical allocation decisions until we have more comprehensive information about write patterns that we can derive by observing several related operations. The filesystem architectures developed have shown that by delaying allocation until flush operations, the system can allocate much larger contiguous extents by making allocation choices based on accumulated write pattern information, rather than based on individual operations [2]. Instead of being allocated as write operations are received, each file is first deferred until natural consolidation points are provided by the trigger events that are received. The write operations during the deferral time between the write acceptance and the physical allocation are stacked together as pending requests showing sequential trends that are used to make more intelligent allocation decisions.

The delayed allocation technique works with the separation of logical allocation of the storage area and physical distribution of particular blocks of disk. When the applications make write operations, the filesystem instantaneously accepts the write requests and claims the required capacity of the existing free space without determining physical block selections. Real physical block choice and placement are deferred to the later flush operations in response to different conditions such as explicit application synchronization requests, periodic filesystem consistency operations or system shutdown procedures [2]. This postponed allocation decision allows the filesystem to have a view of series of related write operations which are logically part of the same file region and allows allocation of larger contiguous extents which would not be recognizable when each operation is done independently.

This article presents the design and implementation of an efficient delayed allocation mechanism within the Novell Storage Services filesystem deployed in Open Enterprise Server environments serving enterprise customers. The approach introduces a lightweight reservation model based on minimal in-memory state that tracks logically committed but not yet physically allocated blocks, eliminating the need for complex persistent tracking structures or heavyweight journaling during the write path. Physical allocation occurs during controlled flush operations triggered by anticipatory delays, explicit synchronization requests, unmount procedures, or memory pressure conditions, providing opportunities for optimized placement decisions that incorporate information from multiple accumulated write operations. The implementation achieves approximately forty percent improvement in write throughput by eliminating journal serialization overhead during writes, enabling the write path to proceed at memory speeds without waiting for disk-based journal updates. Read performance enhancements reach up to two hundred percent through reduced fragmentation and improved sequential access patterns, as files consist of far fewer large extents rather than many small scattered extents. Backup duration experiences dramatic reduction from approximately twenty-four hours to a few hours for representative enterprise filesystem configurations containing multiple terabytes of data across millions of files. These results demonstrate how careful implementation of established filesystem optimization concepts produces significant practical benefits in production deployments, translating theoretical advantages into

Basani, B.N. (2026). Filesystem Delayed Block Allocation for High-Performance Sequential Writes in Enterprise Storage Systems. *South African Computer Journal* 38(2), 52–65.

ISSN 1015-7999 (print) ISSN 2313-7835 (online)

measurable operational improvements for enterprise customers operating large-scale storage infrastructures.

2 BACKGROUND AND STATEMENT OF THE PROBLEM

In historical file system architectures, immediate allocation models are used in which write operations directly and synchronously cause physical blocks of the available free space pool to be assigned. Upon an application making a write request, the allocation subsystem of the filesystem consults internal data structures such as free space bitmaps, extent trees or allocation group metadata to find free blocks. The allocator chooses blocks in policies determined by a set of policies that can in turn be based on spatial locality, load-balancing across storage devices, or alignment with underlying storage geometry. After identifying candidate blocks, the blocks so selected are permanently linked with the target file by modifying the extent mapping structures that store the mapping between logical file offsets and physical block addresses. Such metadata changes should be journaled in the filesystem tree or log of transactions so that they can be treated like data after system failures or power outages and persistent write operations are put in the critical path of each allocation decision.

This is mainly because the immediate aspect of allocation decisions has structural constraints that limit the quality of the resulting data layouts. By the time a write request reaches the filesystem layer, the allocation subsystem only has full information on a particular request that includes the number of blocks needed, the target file and the target offset, and the current allocation of free space on the storage volume. But even future write operations which can come in a milliseconds or seconds later are completely unknown and not visible to the allocation decision process. The allocator will have no idea about those future writes and thus they cannot take this into account during its decision making process. Allocation decisions are made in isolation with each decision being optimized with only the available information at the time and not with regard to larger temporal events in the write workload. This shortsighted view generates decisions that might seem optimal on an individual basis but build up to generate globally suboptimal data formats as files grow over long periods of operation.

Fragmentation of files becomes the natural compound effect of these uncoordinated allocation choices taken at an independent level across time series of related processes. Take a filesystem with a number of applications, with workloads of concurrent write. A first usage stores data in a first file, and triggers the assignment of a set of sequential blocks out of the existing free space pool. A second application is later allocated the next available blocks which may be either next to or interleaved with the allocation of the first file, a few moments later. As the first application later makes further data write to its initial file, blocks which initially had the first allocation directly adjacent to it, may now be filled in by allocations to the second file. The filesystem will be required to reassign the blocks previously used by the first write of a separate physical region to the second write of the first file to cause spatial discontinuity among logically contiguous segments of the same file. With repeated occurrence of this interleaving process in many files and millions of write operations, the data layout of the filesystem becomes more and more scattered and fragmented. Logically-related files, which are made of related sequences of contiguous bytes, are physically represented as groups of numerous small extents spread in different areas of the storage volume.

The level and extent of fragmentation is associated with a number of environmental and work load factors. Age of filesystem is a determining factor, with the longer the filesystem has been in operation, the more chances of the development of fragmentation. Write concurrency increases the effects of fragmentation, since at the same time write operations of two or more applications vie to be allocated out of the common free space pool, and their allocations thus interlace spatially. The availability of free space has a significant impact on fragmentation paths, as a fuller filesystem is far more poorly fragmented because of fewer choices of allocation options. Storage device characteristics of performance are highly sensitive to the nature of access patterns with the

Basani, B.N. (2026). Filesystem Delayed Block Allocation for High-Performance Sequential Writes in Enterprise Storage Systems. *South African Computer Journal* 38(2), 52–65.

ISSN 1015-7999 (print) ISSN 2313-7835 (online)

mechanical hard drives exhibiting particularly pronounced performance degradation under workloads that cause small random accesses relative to large sequential operations [3].

Journaling overhead also adds significant additional costs of performance to the fragmentation problems of immediate allocation models. Write-ahead journaling or other transaction logging systems are universal in use in modern enterprise filesystems to ensure crash consistency. The journaling method works by initially recording metadata alterations to an exclusive journal area and then implementing the metadata alteration at their ultimate locations within the filesystem hierarchy. A journal of the changes that are meant to be made are registered before any changes are done to any filesystem metadata structures such as inode attributes, directory records, extent maps, or free space bitmaps [4]. Once journal write operations have been finished and the journal entries are persisted to stable storage the actual metadata updates are possible. As part of crash recovery following system failures, the filesystem mount operation restores journaled journal entries by processing the journal entries that were committed and incurring any changes that were journaled but not necessarily written to their ultimately allocated locations before the crash happened.

The journaling mechanism has a high performance overhead manifested especially acutely with allocation-intensive workloads. An immediate allocation model will produce journal entries which reflect the metadata changes for an allocation decision made. Characteristics of high allocation rates of write-intensive workloads create sustained journal write traffic that can saturate journal bandwidth, and add significant latency to the write path. The journal provides a point of serialization, which limits parallelism, as operations of concurrent allocation would have to coordinate their journal updates to ensure transactional consistency. A journal needs to log every transaction on disk in order to make the matching operation appear durable, which forms an inherent dependency between journal write latency and filesystem throughput [4]. The journal write overhead in workloads with small write operations which individually cause allocations to individual files may take up a large portion of the total storage bandwidth, and may become the main bottleneck exploiting overall filesystem throughput.

The best view of the practical implications of fragmentation on the real system performance is seen in enterprise backup operations. Backup systems are very important to enterprise data protection policies, and they carry out regular full scans or incremental scans of production filesystems to replicate data at secondary storage systems. In ideal environments with properly laid-out filesystems where files take up sequentially contiguous physical areas, backup operations will have a sustained sequential read throughput rate approaching the maximum sequential bandwidth specifications of the underlying storage hardware. Nonetheless, this ideal access pattern is essentially broken through fragmentation. Reading a heavily fragmented file requires accessing many small extents scattered across the storage device, with each extent transition potentially introducing substantial performance penalties.

The performance impact manifests differently depending on the characteristics of the underlying storage technology. Mechanical hard disk drives exhibit severe performance degradation under fragmented read workloads due to the physical overhead of repositioning read heads across platter surfaces. Storage system performance modeling demonstrates that request access time consists of multiple components including seek time for head positioning, rotational latency, and transfer time. For mechanical disks, seek time dominates performance characteristics for random access patterns, while sequential access patterns minimize seek overhead by maintaining head position within contiguous track regions [3]. Each transition between non-adjacent extents may require a seek operation imposing mechanical delays measured in milliseconds that dramatically reduce effective throughput. The performance differential between sequential and random access patterns becomes particularly pronounced on mechanical storage where seek times can reach ten milliseconds or more, meaning that accessing a file fragmented into thousands of small extents could introduce

Basani, B.N. (2026). Filesystem Delayed Block Allocation for High-Performance Sequential Writes in Enterprise Storage Systems. *South African Computer Journal* 38(2), 52–65.

ISSN 1015-7999 (print) ISSN 2313-7835 (online)

cumulative seek overhead measured in seconds or tens of seconds for operations that would complete in milliseconds with optimal layout [3].

Enterprise deployments utilizing immediate allocation models have reported backup windows extending to approximately twenty-four hours for typical multi-terabyte filesystem configurations, creating severe operational constraints that fundamentally limit data protection capabilities. These extended backup durations force organizations to schedule backup operations during limited maintenance windows, often restricting backups to overnight periods or weekends. Organizations must choose between accepting less frequent backup schedules that reduce operational burden but increase potential data loss in disaster scenarios, or maintaining daily backup attempts that consume substantial operational resources and may not reliably complete before the next production workload cycle begins.

Read performance degradation affects not only backup operations but also routine application workloads that access filesystem data during normal production operation. Database management systems reading transaction log files experience increased latency when logs fragment across many small extents. Analytical platforms and data processing frameworks that scan large datasets observe reduced job completion times when input data files exhibit high fragmentation. Content management systems serving large media files deliver decreased performance to end users when media assets scatter across fragmented storage layouts.

The problem formulation for addressing these limitations requires developing allocation mechanisms that defer decisions sufficiently to observe write patterns while avoiding introduction of unacceptable complexity, memory overhead, or compromise of crash consistency guarantees that enterprise deployments depend upon. A successful delayed allocation approach must provide adequate deferral windows to allow multiple related write operations to accumulate before allocation occurs, enabling the filesystem to identify sequential patterns and allocate appropriately sized contiguous extents. The mechanism must maintain reasonable memory overhead for tracking deferred allocations. Integration with existing filesystem architectures should avoid requiring disruptive on-disk format changes that would prevent in-place upgrades or complicate recovery procedures. Most critically, any delayed allocation mechanism must preserve or enhance crash consistency properties. The journaling system must ensure atomicity of compound operations, guaranteeing that either all components of a multi-step operation complete or none do, preventing partial updates that could leave metadata structures in inconsistent states [4].

3 DELAYED ALLOCATION DESIGN AND IMPLEMENTATION ARCHITECTURE

The implementation of the NSS filesystem delayed allocation has provided a reservation-based implementation, which is a fundamentally different implementation of logical space commitment versus physical block assignment using a two-phase architecture. The lightweight reservation in the first phase is performed in the course of write operations where the filesystem only recognizes that write has occurred and confirms logical space reserved but does not choose physical blocks. The second stage does real physical allocation at controlled flush events which offer natural consolidation points at which several accrued writes can be allocated simultaneously. Such a separation of architecture enables allocation decisions to be deferred without blocking write operations or adding user-perceived latency since reservations can be made quickly through purely in-memory operations not requiring costly disk accesses or journal interactions.

The reservation mechanism is based on the simplest in-memory counter system that is kept in the inode of each file to record the number of reserved blocks that have not been allocated yet. The implementation uses an 8-byte counter field which is the cumulative number of blocks that are dedicated to pending writes that have been acknowledged to applications but without assigning them specific physical addresses. The very lightweight nature of this solution means that even in filesystems with millions of files the amount of memory overhead is negligible since the per-file reservation state is represented by a single scalar value instead of complicated data structures. This

ISSN 1015-7999 (print) ISSN 2313-7835 (online)

small reservation counter fits seamlessly into the existing inode layouts, and does not need new on-disk format changes to enable the delayed allocation feature, so the delayed allocation feature can be rolled out via software updates that do not need filesystem reformatting and data migration operations.

In case an append write operation is directed to a file, the filesystem completes a reservation sequence to ensure that there is available space and to update the reservation state without physically allocating space. There is a first calculation of the required number of blocks in the filesystem to meet the write based on the write size and existing file offset alignment. The system will ensure that there is enough free space before the reservation is accepted by looking at global free space counters which monitor blocks that have been physically free and those that have been reserved but not allocated. This check eliminates the over-commitment cases where the cumulative reservations in all the files may be more than the total available capacity. In case there is sufficient unreserved space, the system adds the number of blocks needed to the file per-inode reservation counter and at the same time subtracts the same number of blocks to the global unreserved free space counter. These counter updates are atomic to ensure that there is consistency between per-file reservation state and global capacity tracking.

The reservation phase does not at all access the filesystem journal or the persistent metadata structures, which is a very important design decision that makes possible the main performance advantages of delayed allocation. To make allocation decisions crash consistent, models of immediate allocation of the past must journal all allocation decisions, and the allocation metadata updates must be written to the journal region and be kept in disk before the allocation can be said to be complete. Designs of log-structured filesystem have shown that write operations can be in fact much faster when permitted to be asynchronous with respect to disk access since memory access speed is many orders of magnitude greater than disk access speed with memory access times in tens of nanoseconds and disk access times in milliseconds. Delayed allocation enables the write operations to execute at memory speeds, without receiving journal updates on the disk, and does so by deferring the physical allocation as well as removing journal interaction during the write phase. The low-latency write completions in the applications are in the microsecond range as compared to milliseconds, which significantly enhances interactive response times and total write throughput.

Physical allocation takes place by use of a number of well-structured trigger pathways which give controlled chances of optimal block allocation using accrued write patterns. The major trigger is anticipatory write delays and it adds a configurable delay between the acknowledgment of the write and the actual allocation. This delay is normally set at one to two seconds, striking a balance between the competing goals of providing as many opportunities as possible to combine write batches and reducing the amount of time in a window between the time a write is registered in physical storage as uncommitted and the time it is actually written to physical storage. Upon expiry of the delay period, the filesystem will launch a flush operation which will allocate the reservation of all the reservations made on the affected file. This batching scheme enables the allocator to view sequences of connected write operations prior to making commitments to physical block allocations, so that it can allocate larger extents contiguously with sequential locality.

Explicit flush operations can offer application-controlled triggers of immediate allocation that guarantee the appropriate durability semantics to applications with data persistence guarantees. Applications which issue synchronization system calls such as fsync, fdatasync or sync operations are answered by the filesystem by deferring all pending reservations to the files which are the subject of the operations. This behavior guarantees that delayed allocation preserves the standard POSIX filesystem semantics in which data is rendered durable and is not lost on crashes until deterministic synchronization operations complete successfully. The cost of metadata update can greatly differ between the synchronous and asynchronous operations whereby synchronous metadata update will

ISSN 1015-7999 (print) ISSN 2313-7835 (online)

require full disk write operations, which could take tens of milliseconds, asynchronous metadata update can proceed at memory speeds and can be accumulated and later persisted [6].

The clean shutdown sequence is done by allocating all outstanding reservations through unmount operations to the entire filesystem. The memory pressure conditions also offer another trigger pathway, which allows the filesystem to react adaptively to the constraints in system resources. When events of memory pressure occur, delayed allocation implementation will react by selectively initiating allocation of files with the largest pending reservations, since allocation of reservations enables the system to release the corresponding tracking structures and minimise memory footprint. On-demand allocation of particular files or filesystem areas can be caused by internal filesystem consistency operations which need a stable metadata state.

The allocation phase steps are used to allocate accumulated reservations to assign physical blocks based on strategies that are optimized in terms of sequential extent allocation. On a file each having outstanding reservations, the allocator looks at the accumulation of the total number of reserved blocks in all pending writes and then tries to complete the entire reservation with one contiguous extent. The allocator scans free space management structures such as extent trees or bitmap representations of free spaces which are contiguous and whose size is at least equal to the reservation size. By identifying suitable contiguous regions, the whole reservation is mapped to a single extent which covers all the reserved blocks and the best sequential layout is obtained in which logically related file data is allocated to physically adjacent storage locations. Understanding disk geometry and performance characteristics becomes essential for optimizing allocation strategies, as disk drives exhibit complex performance behaviors influenced by factors including rotational latency, seek time, and transfer rate characteristics [7]. If no single contiguous region exists large enough to satisfy the full reservation, the allocator may split the reservation across multiple extents, though the implementation prioritizes allocating the largest possible extents rather than many small fragments.

During the allocation process, the filesystem updates metadata structures including extent maps that record physical block assignments, free space bitmaps or trees that mark newly allocated blocks as occupied, and inode attributes. These metadata modifications flow through the standard journaling subsystem to ensure crash consistency. However, the journal activity pattern differs dramatically between immediate and delayed allocation approaches. Under immediate allocation, each write operation generates separate journal entries for its allocation, producing continuous journal traffic proportional to write operation rate. Delayed allocation coalesces many write operations into single allocation transactions, generating journal entries only during flush events rather than for each individual write. Metadata update performance research demonstrates that synchronous metadata updates impose substantial overhead, with each synchronous operation potentially requiring full disk rotation latency plus seek time, whereas batched asynchronous updates can reduce the effective per-operation cost by amortizing disk access overhead across many updates [6].

Crash consistency preservation relies fundamentally on the ephemeral nature of reservation state that exists exclusively in volatile memory without any persistent representation. Reservations never appear in the filesystem journal, on-disk metadata structures, or any other persistent storage location. In crash scenarios where power loss, kernel panics, or hardware failures cause unexpected system termination, all volatile memory contents including reservation counters are lost irreversibly. During the subsequent recovery process when the operating system reboots and mounts the filesystem, the recovery subsystem replays committed journal transactions to restore consistent metadata state. Lost reservations are simply discarded and ignored, as they represented uncommitted work from the application perspective. Log-structured filesystem designs have established that crash recovery can proceed efficiently by replaying committed operations from persistent logs without requiring complex analysis of partially completed operations, as operations

ISSN 1015-7999 (print) ISSN 2313-7835 (online)

become durable only when explicitly committed to the log [5]. This recovery behavior maintains standard POSIX semantics where writes become durable only after successful completion of synchronization system calls. The memory-only reservation approach introduces no new failure modes or recovery complexities beyond those already present in traditional immediate allocation systems, preserving the strong reliability characteristics essential for enterprise deployments while enabling substantial performance improvements.

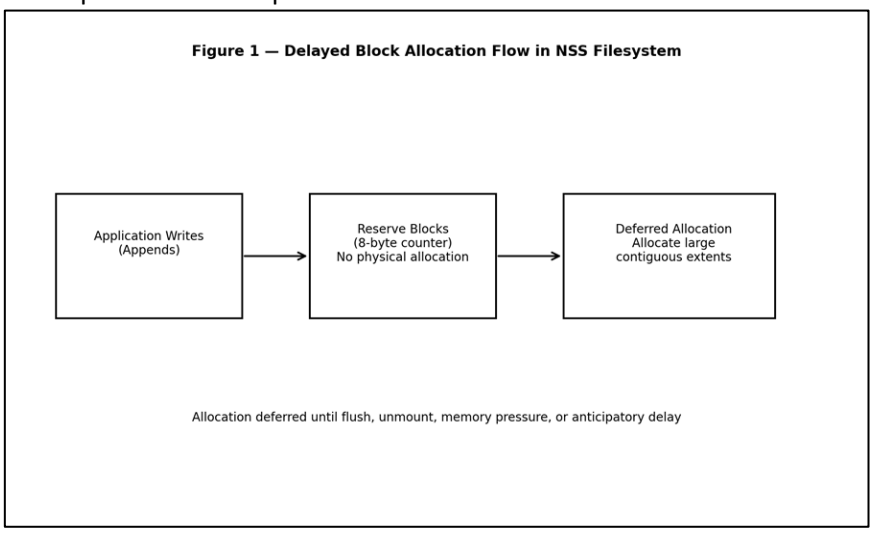

Figure 1: Delayed allocation flow showing lightweight reservation followed by deferred physical allocation.

4 OPERATIONAL IMPACT AND PERFORMANCE ANALYSIS

The performance characterization of the delayed allocation implementation will include several metrics applicable to the enterprise storage implementations where implementation will be evaluated using controlled benchmark environment as well as in the production customer installations. It is an evaluation of the behavior of a filesystem, when it is configured using an immediate allocation scheme, versus when it is configured using a delayed allocation scheme, using the same hardware platform, storage device, and workload parameter settings. Testing measures write throughput, read performance on sequential access and random access patterns, duration of backup operation and journal subsystem behavior measured in terms of transaction rates and bandwidth used.

Write performance measures show about forty percent throughput enhancement with delayed allocation about sustained append workloads typical of enterprise storage conditions. Under the immediate allocation, write throughput is limited by journal bandwidth constraints since every write operation creates allocation metadata entries that need to be journaled prior to that write being completed. The journal can serve as a bottleneck to parallelism in serialization as well as to add latency based on the time to journal write completion. The implementation of log-structured filesystems has shown that metadata updates are a major cost in terms of performance (metadata operations incur time cost), and that the conventional filesystem design would spend large amounts of time on a synchronous metadata operation that would have to wait until disk writes had completed before continuing to execute the metadata operation [8]. Delayed allocation is essentially a change to this performance profile, as it removes journal interaction at write, and enabling reservation updates to be completely in memory without persistence overhead. The resulting throughput increase of about forty percent is a direct conversion to lesser application response times of write-dominated workloads such as database transaction processing, log aggregation, content ingestion pipelines and archival operations.

ISSN 1015-7999 (print) ISSN 2313-7835 (online)

Table 1: Performance Impact Summary of Delayed Allocation Implementation

Performance Metric	Immediate Allocation	Delayed Allocation
Write Throughput	Baseline	~40% improvement
Read Performance (Sequential)	Baseline	Up to 200% improvement
Backup Duration	~24 hours	Few hours
Backup Duration Reduction	—	80–90% reduction
Journal Transaction Volume	High (per-operation)	Low (batched)

The write performance advantage is exaggerated in situations with many small write operations in which metadata overhead is a more significant portion of work. A workload that is executing one thousand little writes will produce one thousand different allocation choices and one thousand different journal entries under instantaneous allocation. Delayed allocation alters this behavior by grouping these one thousand small operations into one allocation transaction in flush operations, three orders of magnitude in journal traffic, and the same data persistence guarantees. The use of log-structured methods has demonstrated that combining multiple operations into one operation can greatly decrease the per-operation overhead, by amortizing large and expensive disk operations over large numbers of logical operations and measurements have shown that a batch size that is hundreds of operations can decrease the individual operation costs by a large factor.

The performance of reads improves significantly over that of writes because write performance is helped by the high data layout quality delivered by the delayed allocation capability to observe write sequences and respond to them. Immediately allocated files are fragmented into many small extents as incremental appends are allocated to the available free space in an uncoordinated manner. The files written using delayed allocation have significantly fewer and much larger extents since the allocation decisions use the information of many accumulated writes to allow the allocator to allocate large contiguous regions, which maintain sequential locality.

Quantitative tests demonstrate to a maximum of two hundred percent read performance advantage in files generated under delayed allocation versus files generated under immediate allocation, which symbolizes throughput augmentations in which delayed allocation may be 3 times more effective than immediate allocation in addressing a comparable read workload. Such a dramatic improvement represents the compounding benefits of reduced fragmentation in many layers of the system. Real-world filesystem workload analysis has found that read operations are prevalent in most enterprise systems, and research has demonstrated that in most deployment settings, reads can comprise two-thirds or more of all filesystem operations [9]. Large extent files allow continuous sequential access during a majority of the read operation whereas large extent files can only allow random access patterns with constant overhead. Through workload characterization studies, it is shown that file access patterns are highly temporal localized, with frequently accessed files frequently accessed by multiple accesses in short timeframe intervals thus layout optimization becomes especially important when dealing with frequently accessed data [9].

The most operationally critical metric of enterprise customers is the duration of backup operation because the main defense against hardware failures, software errors, security incidents, or operational mistakes as the primary cause of data loss is the backup systems. Production deployments with instant allocation recorded backup windows of nearly twenty-four hours when

ISSN 1015-7999 (print) ISSN 2313-7835 (online)

the enterprise filesystem was characterized by multiple terabytes of data spread out among millions of files. Such long-lasting backup times posed serious operation limitations which essentially restricted the ability of the enterprise to provide data protection and had to make uncomfortable trade-offs between frequency of the operation and the weight of the operational burden.

Following the deployment of delayed allocation, the process of backup was completed in hours on the same filesystems that it had taken about twenty-four hours earlier, which is comparable to the duration cuts of about eighty to ninety percent on average enterprise setups. This significant decrease in performance is directly due to better read performance achieved through less fragmentation and better sequential data layout. In practice, filesystem experiments show that the vast majority of read requests read complete files sequentially (i.e. either forward or backward) instead of requesting read offsets, and the predominant read pattern in many production workloads is sequential reads instead of random reads [9]. The reduced backup time allows various transformative operational enhancements. Daily backup schedules allow organizations to adopt a backup schedule where weekly or bi-weekly backups were the only viable option before which recovery point objectives and possible loss of data in the event of a disaster are significantly lowered. Backup windows are moved to off-peak time but not extending to business periods and therefore conflicts with production workloads are eliminated.

The activity analysis of journal subsystem proves the high rate of metadata update overhead reduction that was obtained by a batching approach of delayed allocation. By tracking journal write rates in conditions of sustained append workload, it can be demonstrated that journal traffic is dramatically reduced in delayed allocation than in immediate allocation. Immediacy allocates to generate journal continuity directly proportional to write operation rate. A delayed allocation causes the journaling activity to occur on an intermittent basis because of periodic flush operations, as opposed to continuous journaling per-operation, and changes the journal traffic characteristics essentially to periodic batched operations instead of continuous high-frequency operations. Research on log-structured filesystems has demonstrated that by clustering together several related operations into compound operations metadata overhead can be decreased by ten to many times since the cost of a single operation managing journals is amortized over the entire set of operations in the batch [8].

Journal bandwidth demands are proportional to the corresponding reduction in the number of journal transactions, which removes journal resource contention and makes journal bandwidth no longer a bottleneck to filesystem throughput. The secondary performance payoffs of the decrease in journal pressure are several. The use of the CPU to manage the journal reduces when fewer journal operations are to be processed.

This freed CPU capacity becomes available for application workloads, improving overall system efficiency. Storage devices hosting journal regions experience reduced write activity, decreasing wear on solid-state storage and reducing contention on mechanical disks. Journal recovery times after crashes also improve, as fewer journal entries require replay during filesystem mount operations [10].

Fragmentation analysis quantifies layout improvements through extent count metrics. File extent distributions under delayed allocation show dramatically fewer extents per file compared to immediate allocation. Large files that fragment into thousands of small extents under immediate allocation instead consist of dozens of large extents under delayed allocation. Median extent sizes increase by multiple orders of magnitude. These improvements persist throughout filesystem lifetime rather than requiring periodic defragmentation maintenance. The allocation mechanism continuously produces superior layouts as files grow, maintaining performance characteristics over long operational periods.

Enterprise operational practices benefit beyond raw performance metrics. Compressed backup windows enable more flexible scheduling that accommodates business requirements [10]. Backup

Basani, B.N. (2026). Filesystem Delayed Block Allocation for High-Performance Sequential Writes in Enterprise Storage Systems. *South African Computer Journal* 38(2), 52–65.

ISSN 1015-7999 (print) ISSN 2313-7835 (online)

operations can align with actual data change patterns rather than being constrained by duration limitations. The ability to perform frequent backups reduces recovery point objectives and improves overall data protection posture. Application workloads experience improved performance when accessing filesystem data, as reduced fragmentation benefits all sequential read operations. Database systems, analytics platforms, content delivery systems, and user applications all realize performance improvements from better data layout.

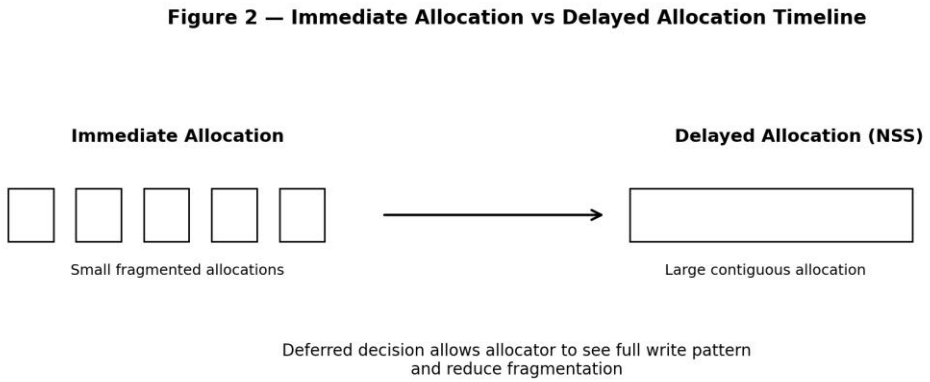

Figure 2: Comparison of immediate allocation versus delayed allocation, demonstrating reduced fragmentation through larger contiguous allocation.

5 CONCLUSION

The delayed block allocation implementation within the Novell Storage Services filesystem demonstrates how optimized realization of established filesystem techniques produces measurable benefits in production enterprise environments. The design introduces a lightweight reservation mechanism based on 8-byte in-memory counters that decouples logical space commitment from physical block assignment, enabling allocation decisions to incorporate information from multiple accumulated write operations rather than proceeding independently for each individual operation. This architectural separation allows the filesystem to defer block assignment until sufficient write pattern information becomes available, enabling allocation of larger contiguous extents that preserve sequential locality and minimize fragmentation. The reservation mechanism operates with negligible memory overhead and computational cost, requiring only simple counter arithmetic without complex data structures, demonstrating that effective delayed allocation does not necessitate heavyweight implementation complexity.

Performance improvements span dimensions critical to enterprise storage operations. Write throughput gains of approximately forty percent result from eliminating journal serialization bottlenecks during the write path, as reservation operations complete in memory without expensive disk-based journal updates. Applications observe dramatically reduced write latencies, enabling higher sustained write rates for workloads including database transaction processing, log aggregation, and archival operations. Read performance enhancements reaching up to two hundred percent stem from superior data layouts that maintain sequential locality and reduce extent counts. Files created under delayed allocation consist of dozens of large contiguous extents rather than thousands of small scattered extents, enabling sequential read operations to maintain streaming

Basani, B.N. (2026). Filesystem Delayed Block Allocation for High-Performance Sequential Writes in Enterprise Storage Systems. *South African Computer Journal* 38(2), 52–65.

ISSN 1015-7999 (print) ISSN 2313-7835 (online)

access patterns that fully utilize storage bandwidth. This layout quality persists throughout filesystem lifetime, providing sustained performance benefits across all sequential read operations.

Backup duration reductions from approximately twenty-four hours to a few hours represent the most operationally transformative outcome for enterprise customers. The dramatic compression of backup windows fundamentally alters data protection capabilities and operational flexibility. Organizations can implement daily backup schedules where previously only weekly backups were feasible, dramatically reducing recovery point objectives and limiting potential data loss exposure. Compressed backup duration enables flexible scheduling that accommodates business requirements and decreases vulnerability to backup failures by shortening execution windows. Journal subsystem activity reductions eliminate a primary performance bottleneck while maintaining identical crash consistency guarantees. By coalescing many operations into periodic allocation transactions, delayed allocation reduces total journal write volume by orders of magnitude, producing benefits including decreased CPU utilization, reduced storage device wear, improved recovery times, and elimination of journal bandwidth as a throughput constraint.

The implementation maintains filesystem reliability through careful architectural decisions that preserve existing crash recovery semantics without introducing new failure modes. Reservations reside exclusively in volatile memory and never appear in persistent structures, ensuring crash recovery procedures remain unchanged. During recovery after system failures, the filesystem replays committed journal transactions while discarding lost reservations that represented uncommitted work. This recovery behavior maintains standard POSIX semantics where data durability requires explicit synchronization operations. The memory-only reservation approach introduces no additional complexity to crash recovery paths, preserving reliability characteristics essential for enterprise deployments while enabling substantial performance enhancements.

The work demonstrates the critical importance of implementation quality in realizing practical potential of filesystem optimization concepts. While delayed allocation represents an established technique in filesystem literature, the specific implementation approach determines achievable performance gains, operational applicability, and compatibility with existing architectures. The lightweight reservation mechanism minimizes runtime overhead, enables effective allocation deferral through configurable delays and multiple trigger pathways, and preserves crash consistency through memory-only state requiring no special recovery handling. Integration with existing NSS filesystem components demonstrates that substantial benefits can be achieved through focused enhancements rather than wholesale architectural redesigns. The delayed allocation mechanism integrates into existing inode structures, operates alongside existing allocation policies, and flows decisions through standard journaling mechanisms with minor modifications to support batched transactions.

Enterprise storage systems demand solutions addressing multiple competing objectives including performance across diverse workloads, reliability ensuring data protection, operational simplicity minimizing administrative burden, and economic efficiency maximizing infrastructure value. The NSS delayed allocation implementation achieves this balance by targeting append-heavy sequential workloads dominating enterprise environments while maintaining full compatibility with existing filesystem guarantees and operational practices. The resulting system delivers substantial improvements in write throughput, read performance, and backup duration while requiring no changes to administrative procedures or application interfaces. This successful translation of filesystem research concepts into production impact demonstrates that careful attention to implementation details, resource efficiency, and architectural compatibility enables delivery of optimization techniques into real-world deployments producing tangible operational and business value for enterprise customers operating large-scale storage infrastructures.

References

Basani, B.N. (2026). Filesystem Delayed Block Allocation for High-Performance Sequential Writes in Enterprise Storage Systems. *South African Computer Journal* 38(2), 52–65.

ISSN 1015-7999 (print) ISSN 2313-7835 (online)

1. R. Spillane et al., "Story Book: An Efficient Extensible Provenance Framework," 2009. Available: https://www.usenix.org/legacy/events/tapp09/tech/full_papers/spillane/spillane.pdf
2. Adam Sweeney, Silicon Graphics, "Scalability in the XFS File System," USENIX, 1996. Available: https://rcs.uwaterloo.ca/~ali/cs854-f23/papers/xfs.pdf
3. Michael P. Mesnier et al., "Modeling the relative fitness of storage," ACM Digital Library, 2007. Available: https://dl.acm.org/doi/10.1145/1254882.1254887
4. Stephen C. Tweedie, "Journaling the Linux ext2fs Filesystem," LinuxExpo ’98. Available: https://www.cs.columbia.edu/~junfeng/12sp-w4118/lectures/journal-design.pdf
5. Mendel Rosenblum and John K. Ousterhout, "The Design and Implementation of a Log-Structured File System," ACM Transactions on Computer Systems, 1992. Available: https://people.eecs.berkeley.edu/~brewer/cs262/LFS.pdf
6. Gregory R. Ganger, Yale N. Patt, "Metadata Update Performance in File Systems," USENIX Symposium on Operating Systems Design and Implementation, 1994. Available: https://users.ece.cmu.edu/~ganger/papers/osdi94.pdf
7. Chris Ruemmler and John Wilkes, "An introduction to disk drive modeling," IEEE Computer, 1994. Available: https://www.cs.cmu.edu/~natassa/courses/15-721/papers/IEEEComputer.DiskModel.pdf
8. Margo Seltzer et al., "An Implementation of a Log-Structured File System for UNIX," Winter USENIX, 1993. Available: https://www.usenix.org/legacy/publications/library/proceedings/sd93/seltzer.pdf
9. [9] "A Comparison of File System Workloads," USENIX Conference Policies, 2000. Available: https://www.usenix.org/conference/2000-usenix-annual-technical-conference/comparison-file-system-workloads
10. [10] Nitin Agrawal et al., "Design Tradeoffs for SSD Performance," USENIX ,2008. Available: https://www.usenix.org/legacy/event/usenix08/tech/full_papers/agrawal/agrawal.pdf

Basani, B.N. (2026). Filesystem Delayed Block Allocation for High-Performance Sequential Writes in Enterprise Storage Systems. *South African Computer Journal* 38(2), 52–65.

ISSN 1015-7999 (print) ISSN 2313-7835 (online)