Frangipani: A Scalable Distributed File System

A Paper Review

Jun 1st, 2025

Student:

Juan de Dios Fernando Lerzundi Ríos
Universidad Nacional de Ingeniería, Facultad de Ciencias,
e-mail: juan.lerzundi.r@uni.pe

Curso:

Programación Concurrente y Distribuida
Práctica Calificada 5

Abstract

The Frangripani file system has, according to its creators, achieved a distributed, easily scalable, consistent, fault tolerant and easily manageable state, which in addition doesn't compromise any performance. It is build upon Petal, a storage service that brings us a shared virtual disk workspace.

This paper reviews the Frangipani: A Scalable Distributed File System [[4]], focusing on its scalability and design trade-offs. How the filesystem is implemented, its architecture and what clever ideas were executed to be able to bring forward all these capabilities.

Keywords: file system, storage system, benchmarks, recovery, workload.

Contents

---

1 Introduction

This work stems from the growing need to manage distributed file systems that handle large numbers of users and data, without implying a heavy administrative burden or increasing technical complexity. File system management in large installations poses increasing challenges as the volume of data and the number of users increase.
Traditionally, expanding capacity and performance involved adding additional disks and servers, with the resulting operational burden of manually allocating, moving, or replicating file sets and balancing access hotspots. While technologies like RAID partially alleviate these issues at the single-server level, a truly scalable infrastructure requires a distributed approach that unifies disks and nodes under common management without degrading the user experience.

In this context, Frangipani emerges as a system that abstracts a large number of disks and servers into a single coherent, reliable, and scalable system. Its modular design, based on Petal and a top layer with

distributed locking service, makes it easily scalable and resilient to failures.

As such, Frangipani pursues the following key objectives:

1. **Incremental scalability**, where each new server or disk is added as a "building block" that increases capacity and performance without interrupting service.
2. **Global consistency**, ensuring that all clients see a single, consistent view of the data, even when multiple nodes are concurrently accessing and modifying the same files.
3. **High availability**, tolerating machine, disk, or network failures through automatic replication and log-based recovery, without requiring human intervention.
4. **Administrative simplicity**, reducing manual configuration and backup tasks, thanks to hot snapshots and a uniform resource management model.

From a theoretical perspective, Frangipani adopts modern distributed systems design principles, combining:

1. **Storage Abstraction:** Petal presents servers with a virtual disk with a sparse, replicated address space, decoupling the file system from the underlying physical hardware and enabling placement and availability optimizations.
2. **Coordination through distributed locking:** Instead of complex consistency protocols or full-file replication, Frangipani uses a locking service that offers "multiple readers/single writer" semantics, facilitating consistent data and metadata caches on each node.
3. **Private logging and local recovery:** Each server maintains its own metadata log, applying a write-ahead logging scheme that allows changes to be reordered and grouped before committing them to the virtual disk, speeding up operations and reducing the window of inconsistency.

Finally, Frangipani positions itself within the historical evolution of distributed file systems, such as **AFS**, parallelized **NFS**, and academic studies (e.g., Zebra, xFS), providing a modular design that simplifies the integration of new advances in high-speed networking, memristor storage, or cloud deployments. This approach makes Frangipani a robust and adaptable proposal, capable of serving both high-performance computing environments and private clouds and engineering clusters.

2 Theoretical Framework

Frangipani's design draws on a early tradition of research and practice in distributed file systems and virtualized storage.

Frangripani is based in three central pillars:

- **Petal**: A distributed virtual disk system that offers fault tolerance, replication, snapshots, and horizontal scalability.
- **Distributed File Systems**: These require consistency, synchronization, and performance under multiple simultaneous accesses.
- **Distributed blockchain services**: Ensure that multiple servers consistently access shared structures

Frangipani's design is inspired by previous models such as the Cambridge File Server but introduces simplicity and high integration with its underlying layer (**Petal**).

The key concepts, predecessor technologies, and theoretical principles underlying its architecture are explored in depth below.

2.1 Storage Abstraction

2.1.1 Local disk vs. distributed storage

In a traditional file system (e.g., UNIX UFS), the address space corresponds directly to the cylinders, tracks, and sectors of a physical disk. This affinity allows for optimizations based on physical location, but scales poorly when multiple disks or controllers are added. Meanwhile, virtual disks, present the operating system with a uniform interface: a contiguous address space, without exposing the underlying fragmentation or replication.

2.1.2 Petal: scattered and replicated storage

Petal offers a 2^{64} byte address space, physically allocating blocks only when they are written, and freeing them with a decommit operation. Additionally, automatic block replication across multiple nodes, ensuring high availability if one replica fails. It also has copy-on-write snapshot support, facilitating consistent "hot" backups without stopping the system.

2.1.3 Theoretical advantages

- **Decoupling:** The file system layer should not be aware of the physical topology, simplifying recovery logic and allowing dynamic allocation policies.
- **Scalability:** Adding nodes to Petal increases capacity and throughput without reconfiguring the file system.

2.2 Consistency and Coherence Models in Distributed Systems

2.2.1 Sequential vs. causal vs. eventual consistency

Frangipani guarantees sequential consistency (Strong Consistency): all read operations see a linearizable history of writes. And, it contrasts this with systems based on eventual replication (e.g., Dynamo), where updates can take time to propagate.

2.2.2 Cache coherence protocols

Inspired by the MESI protocol of shared memory architectures: Modified, Exclusive, Shared, Invalid.

It uses a multiple-reader/single-writer locking scheme: servers acquire read locks for cache and write locks for modifications, invalidating or writing caches as required.

2.2.3 CAP Theorem and Frangipani Design

The CAP theorem states that in the presence of network partition (P), a system must choose between Consistency (C) and Availability (A). In this case, Frangipani opts for CM (Partition Consistency and Tolerance): in the event of partitions, unreachable nodes lose their leases and stop serving, preserving global consistency.

2.3 Coordination: Distributed Blocking Services

2.3.1 Locking models

- **Pessimistic blocks:** prevent conflicts before they occur.
- **Optimistic locks:** allow conflicts and resolve them after detecting them (e.g., Coda).

2.3.2 Implementation with Paxos

Frangipani's lock service distributes replicated lock tables using Paxos, ensuring fault tolerance of lock servers. In which each table (identified by an ASCII string) manages locks named by 64-bit integers. This way, customers maintain renewable 30-second leases, ensuring that any lock released due to expiration is recoverable.

2.3.3 Lessons from previous systems

- **Andrew File System** (AFS) and DFS used callbacks to invalidate caches, but with more rigid client-server architectures.
- **Echo** and **Sprite** introduced similar invalidation and downgrade schemes, but Frangipani integrates them directly into the kernel on top of Petal.

2.4 Logging and Recovery: Private Redo Logging

2.4.1 Write-ahead logging (WAL) basics

Database-based technique (ARIES), where all modifications are recorded before being applied, allowing transactional recovery in which Frangipani logs only metadata, not user data, balancing performance vs. fsync semantics.

2.4.2 Logs partitioned by server

Each server gets a 128KB log region in Petal, managed as a circular buffer. In the event of a crash, another node recovers the log and applies only the entries whose version is higher than the current one (using version numbers in blocks).

2.4.3 Autonomous recovery

Recovery is triggered when a lease expires or when a lock is needed from a downed server. Avoids online fsck, reducing downtime.

2.5 Layered Design and Historical Comparisons

System	Storage	Consistency	Logging	Scalability
NFS	Centralized server	Weak (close-to-open)	None	Limited
AFS	Client replication	Callbacks (FSM)	None	Moderate
Coda	Offline cache	Eventual + reconciliation	Journaling	Moderate
Zebra/xFS	Distributed RAID	Block-level locks	Log-structured	High
Frangipani	Petal (virtual disk)	Reader/writer locks	Metadata WAL	Incrementally high

Table 1: Comparison of distributed file systems

- **CFS** (Cambridge File Server) introduced the idea of separating storage and VF-layer, but Frangipani goes further by integrating snapshots and automatic failover.
- **NASD** and **Object Storage** offer fine-grained block access, but require specialized APIs; Frangipani maintains the POSIX interface.

2.6 Principles of Scalability and Load Balancing

2.6.1 Multi-node parallelism

Multiple Frangipani servers can serve reads and writes in parallel, taking advantage of multiple network links and disk heads.Then, Petal applies 64KB striping across distributed physical disks.

2.6.2 Adaptive balancing

Adding a new node is transparent, simply configure it with the virtual disk location and the lock service. Also, automatic redistribution of logs and locks when lock servers change.

For consistent snapshots, it manages by using a global lock as a barrier to creating Petal snapshots without subsequent recovery.

2.7 Modern Extensions and Current Relevance

1. **Cloud storage (S3, EBS):** Frangipani anticipates block-layer virtualization and snapshots, principles widely used today.
2. **High-speed networks (RDMA, InfiniBand):** could be integrated into Petal to reduce write latencies and accelerate coherence.
3. **Global file systems (e.g., Ceph, Lustre):** inherit from Frangipani the storage-file system separation and locking services, but add distributed metadata and object hashing.

3 Methodology

3.0.1 System Structure

Frangipani's design is based on a two-tier architecture. At the bottom layer, Petal provides a replicated, sparse virtual disk, automatically managed among multiple cooperating storage servers to ensure high availability and allow for consistent snapshots without interfering with ongoing operations. On top of this common disk, each Frangipani server runs a kernel module that intercepts POSIX calls and performs reads and writes using the Petal driver, while employing a distributed lock service to coordinate concurrent access by multiple readers and a single writer without requiring direct communication between the nodes. This modular separation makes it easy to implement new storage policies in Petal without affecting the file system logic.

3.0.2 Disk Organization

Taking advantage of Petal's vast 2^{64} byte address space, Frangipani reserves fixed regions for different internal structures. A first section stores shared configuration parameters; then, each server receives a private 128KB log space for its metadata records. This is followed immediately by the allocation bitmaps and the 512B inode area, followed by 4KB small blocks and 1TB large blocks for hosting files larger than 64KB. This simplified scheme reduces space management complexity and allows capacity to be scaled by adding nodes without having to reorganize existing data.

3.0.3 Logging and Recovery

To achieve rapid, intervention-free recovery, Frangipani employs write-ahead logging exclusively on metadata. Each relevant change is first recorded in the circular log before updating the persistent structures on the virtual disk. When a server fails or its lock lease expires, a recovery daemon takes over execution of the outstanding logs, applying only those with version numbers higher than those stored in each block. This database-inspired technique eliminates the need to run fsck after each crash and ensures that metadata consistency is restored within seconds.

3.0.4 Cache Synchronization and Coherence

Frangipani guarantees global consistency through a multiple-reader/single-writer locking protocol applied to logical segments including logs, bitmaps, inodes, and entire files. When a server receives a lock downgrade or release request, it immediately writes dirty cache portions or invalidates them as appropriate, thus avoiding inconsistent views. To prevent deadlocks in operations spanning multiple segments, the server first determines all lock candidates, sorts them by inode address, and then acquires the locks in that sequence, repeating the process if it detects changes during the verification phase.

3.0.5 Distributed Blocking Service

The lock service used by Frangipani is backed by a set of replicated servers using Paxos. Each Frangipani server maintains a "clerk" that manages 30-second leases and sends asynchronous messages (request, grant, revoke, release) to acquire or release locks. The use of Paxos ensures that, in the event of lock node failures, the global state is automatically recovered and the lock tables are rebalanced, allocating segments to new servers with minimal lock redistribution.

3.0.6 Dynamic Administration and Backups

Both Frangipani servers and Petal and lock nodes can be hot-added or hot-removed. When a new Frangipani server starts, it simply tells it the location of the virtual disk and lock service; it then obtains its log region and starts operating without disturbing the others. For live backups, crash-consistent Petal snapshots are created and can be copied to tape or mounted directly as read-only volumes. Optionally, a global lock in exclusive mode is used to enforce a barrier, ensuring that the snapshot reflects a file system state without the need for subsequent recovery.

3.0.7 Performance Evaluation

Empirical validation included the Modified Andrew Benchmark, which took the system down after each phase to account for all deferred I/O and compare latencies with AdvFS; the Connectathon Benchmark, which measured fine-grained metadata operations and 1MB data; throughput tests with 350MB files; and contention scenarios where readers and writers competed for the same file across various block and read-ahead sizes. These tests demonstrated that Frangipani delivers performance comparable to advanced local file systems, scales nearly linearly to network saturation, and recovers quickly from failures without human intervention.

4 Results and Discussion

Empirical results show that:

- The authors conducted a battery of tests that included standardized benchmarks, throughput analysis under different load conditions, and simulations of failures in critical components. The results obtained not only demonstrate that Frangipani meets its goals but also illustrate the technical compromises made to achieve a simple and robust distributed file system.

- One of the main experiments involved running the Modified Andrew Benchmark (MAB), adapted to unmount the file system after each phase to ensure that all deferred writes were completed before starting the next stage. In this test, Frangipani was compared to a local AdvFS-based system, revealing that its times are, on average, as good as those of its local counterpart, with minor differences attributable to lock redirection and log writing.
- The Connectathon Benchmark, a collection of regression and consistency tests for network file systems, was used. In this case, Frangipani showed competitive performance, especially in common operations

such as stat, readdir, file creation, and reading/writing 1MB blocks. In metadata access tests, performance remained constant even under load, suggesting that the logging and locking strategy is lightweight enough not to be a bottleneck in directory control tasks.

- It was aslo revealed that certain limitations derived from design decisions. One of the most critical scenarios evaluated was concurrent writing to shared files. Since Frangipani employs full-file locking, throughput in this case drops sharply with the number of simultaneous writers. For example, when writing small data to the same file from multiple nodes, contention for the exclusive lock causes long waits and request queues. Although the system maintains consistency and stability, performance degrades in direct proportion to the number of writers.
- In concurrent read tests from multiple servers on different files, Frangipani demonstrated much more efficient performance. With low contention for read locks and the ability to leverage the page cache on each node, throughput increased almost linearly with each new server.
- Finally, the snapshot functionality for hot backups was tested, using Petal's native support. By taking snapshots without global blocking, the backups reflect a crash-consistent state, which is sufficient in most cases. For situations requiring full consistency at the file system level, the authors introduced a brief synchronized barrier using a global exclusive lock, after which the snapshot was taken. This operation did not significantly impact performance, but it ensures that the backup data does not require subsequent log re-execution, simplifying restoration.
- Overall, the experimental results confirm that Frangipani achieves its goal of providing a distributed file system that scales efficiently, preserves data consistency, recovers from failures without intervention, and maintains competitive performance.

5 Conclusions

The work presented in Frangipani: A Scalable Distributed File System convincingly demonstrates that it is possible to design a distributed file system that properly balances the fundamental pillars of distributed systems engineering: scalability, consistency, fault tolerance, and administrative simplicity. Through a clearly defined two-tier architecture—Frangipani as the file system and Petal as the storage infrastructure—the authors achieve a separation of responsibilities that favors both the independent evolution of each component and the ease of maintenance of the system as a whole.

From a performance perspective, experiments show that Frangipani offers comparable throughput to local file systems like AdvFS, despite operating on a shared virtual disk and going through an additional layer of coordination. This efficiency is maintained even under demanding massive read or write scenarios, while in cases of highly concurrent access to shared files, the system exhibits limitations derived from its decision to use full-file locks. However, these restrictions reflect a deliberate choice in favor of strong consistency and simplicity of the cache model, which avoids subtle errors and facilitates reasoning about system behavior.

Beyond its concrete implementation, Frangipani leaves a lasting lesson: that the combination of simple structures and well-founded policies can result in a distributed system that is not only effective, but also surprisingly easy to manage, maintain, and scale. This lesson remains relevant today, especially in a context where system complexity tends to grow without a commensurate gain in robustness or clarity. In that sense, Frangipani not only solves a technical problem but also sets a precedent for disciplined and pragmatic design.

References

[1] Gems of Coding. Frangipani - a scalable distributed filesystem. `https://gemsofcoding.com/Frangipani-FileSystem/`, n.d. Artículo de opinión consultado en línea.

[2] T. Harada. Application of probabilistic–statistical methods to image metrology. Presented at the International Conference on Image Processing, September 1995. pp. 83–86.

[3] Kaiyan99yankai. Distributed systems learning: Frangipani. https://medium.com/@kaiyan99yankai/distributed-systems-learning-frangipani-1421c080db, December 2022. Artículo de opinión consultado en línea.

[4] Chandramohan A. Thekkath, Timothy Mann, and Edward K. Lee. Frangipani: A scalable distributed file system. https://doi.org/10.1145/268998.266694, 1997. Presented at the 16th ACM Symposium on Operating Systems Principles (SOSP '97), Saint-Malo, France.