Frangipani:A Scalable Distributed File System

Shreepadma Venugopalan
CS 739: Distributed Systems
University of Wisconsin, Madison

17 Feb 2006
Spring 2006

1 Overview

This paper describes the design and implementation of Frangipani, a scalable distributed filesystem, built on top of Petal. Petal provides a incrementally scalable, highly available and automatically managed storage service. Multiple machines run Frangipani on top of a shared virtual disk, using a distributed locking service to ensure coherence.Even though Frangipani inherits much of its scalability, fault tolerance and easy administration from Petal,its design is constrained because of by Petal. .For example, using Frangipani with a replicated Petal disk implies logging occurs twice. *We could solve the problem of double logging by having Frangipani take care of the replication. But by doing so we may introduce too much complexity into the file system.*One other problem arises from Frangipani running on a virtual disk. Unlike other file system, it cannot exploit location information in placing data. *We see that Petal influences all most all aspects of Frangipani design.More generally speaking, in what ways does Petal makes building filesystems easier? In what ways does Petal make the file system design less ideal? Sitting on top of Petal, does Frangipani make the correct tradeoffs?*

2 System Structure

Machines running Frangipani servers access Petal virtual disk over the network. Alternatively, same machine can run both Frangipani and Petal. Clients may form the third tier in the system and access a Frangipani server over the network. Clients may also reside on the same machine as the server in which case accesses look local.

Individual servers dont talk to each other directly, but instead use Petal as the backplane. *This method of communication reduces the global state and greatly simplifies the system.* Further,locks are stored in Petal. In this way, Frangipani makes use of the reliable underlying distributed store to provide consistent data access.

Frangipani system doesnt provide any form of security to restrict accesses to sensitive data. Instead Frangipani assumes an environment of trust. *Both Petal and Frangipani take a lax view on security and provide the most basic solution - no security. Petal should probably implement some form of block level security. A higher level trusted entity could give authorization to clients and the lower levels can merely implement it at the block level.One another problem arising from the lack of block level security is a buggy implementation can lead to clients writing data to the wrong location. We could also provide security by using the sparse address space of Petal to make the location of a file inode exponentially difficult. But most of the disrtibuted file systems did not care about security enough. Even a widely used system like NFS provided no security.*

3 Design-The piece details

This section presents the discussion of various aspects of Frangipani design-Disk Layout, Logging and Recovery, Synchronization and Locking.

3.1 Disk Layout

Frangipani allows multiple file systems to access the same Petal volume. The first region of the disk holds shared configuration parameters and housekeeping information.The second region holds the logs. Each Frangipani server has a private log and the region can hold upto 256 such logs. The third region contains the allocation bitmap to indicate which blocks in the remaining region are free. The fourth region consists of the inodes. Inodes are 512 bytes long to avoid false sharing, between servers that would occur if two servers access different inodes in the same block.*However this leads to a 4x space waste. But we think this tradeoff is reasonable given the simplicity of the scheme.* The fifth region hold small blocks, each 4KB in size. The first 64KB of a file are stored in small blocks. If a file grows more than 64KB, the rest is stored in one large block. The remainder of the Petal address space holds the large blocks.

Frangipani use of 1TB block sizes greatly simplifies the design by eliminating the need for indirect pointers and indirect blocks. Indirect blocks complicate the metadata that needs to be tracked.Frangipani can afford to allocate 1 TB blocks without worrying about internal fragmentation because all the allocation is done lazily. However Petal's 64KB blocks can lead to some internal fragmentation.

One other issue is the log size.Given the large sparse address space of Petal the log could probably be bigger. However Frangipani probably wants to make use of space pressure to commit the transactions and remove them from the log. Also, having a bigger log would increase the recovery time .Hence the tradeoff that Frangipani makes doesnt seem too bad.

Frangipani also sets a static limit on almost everything. For example one of the most constrictive limits seems to be the limit on the number of servers .Even though the number of large files seems large enough at first (16 million) such a limit would not work very well for an application like a photo server. Frangipani probably needs to dynamically set the limits for number of small files and the size of large files. But to be fair to the authors the limits worked well for most workloads during the time the paper was written.

3.2 Logging and Recovery

Frangipani uses Write Ahead Logging to log metadata writes.Metadata only logging keeps log sizes small and gives reasonable write performance. *Logging metadata keeps the file system data structures consistent and reduces the cost of recovery. However metadata only logging does not help with data losses and inconsistencies.*

Log records are flushed to Petal in the same order that the updates they describe were requested. The permanent location are updated every 30 seconds by the Unix update daemon.*In presence of multiple Frangipani servers it becomes important to maintain cache consistency. Also Frangipani doesnt make sure that meta data reaches the disk before the data does. This can lead to some mis ordering and inconsistencies. Ordered journaling reduces the chances of mis ordered data at minimal performance loss. Frangipani should probably use ordered journaling to guarantee that metadata updates go first.*

Frangipani introduces an additional level of logging over Petal. *However both levels of logging serve different purposes.Petal's logging is needed to keep the mirrors in sync, whereas Frangipani's logging keeps metadata consistent. But this incurrs additional physical writes for every logical write.For example, when a dirty inode is flushed to disk, Frangipani issues two logical writes and Petal issues three physical writes for each logical write, incurring a total of six phyical writes. One possible solution to solve the problem is to have Frangipani issue the writes to the replicas. However this would introduce additional complexity into Frangipani and would need Petal to expose some of the underlying details to Frangipani.*

If a Frangipani server crashes, the failure is detected by either the client of the server or the by the lock server. The recovery daemon is given explicit ownership of the failed server's log and locks. The daemon finds the log start and runs recovery.*But the lease may have expired because of a network failure and could lead to false recovery. This is an example of an abstarction that cannot be extended from a single system because of the distributed nature of failures.*

3.3 Synchronization and Locking

Frangipani uses multiple reader/single writer locks to ensure synchronized access to the shared on disk data struc-

tures. When the lock server detects a conflict, the current holder of the lock is asked to release or downgrade it to remove a conflict. On disk structres are divided into segments with locks for each segment.

Frangipani provides locking at the granularity of files. *This could be restrictive to applications like a database that runs on top of Frangipani. Database systems may require locks at the record level to maintain enough concurrency in the system.*

The lock server deals with client failure using leases. When a client first contacts a lock server, it obtains a lease. Additional locks are merely added to the existing lease. Each lease has an expiration time after its creation or its renewal. A client must renew it lease before the expiration or else the server would consider it to have failed.

Like discussed before, a server may not be able to renew its lease because of a network failure. However it may still try to access the Petal volume. Although a Frangipani server checks that its lease is valid before issuing a write, Petal does no checking when a write request arrives. To solve this problem the authors plan to add an expiration timestamp to each write request to make sure the write is valid.

4 Summary

The Frangipani file system provides it users with coherent, shared access to the same set of files. Frangipani leaverages most of its scalability, ease of administration and availability from underlying Petal. But Frangipani's locking and synchronization is complicated by the distributed nature of the system.*However we think that Frangipani does pretty well and provides a system to scalably use the large storage space provided by Petal.*