Frangipani / Cache Consistency

Premise

- We want to build a distributed file system that:
  - Gives users a consistent view of the same set of files
  - Scalable by just adding more servers
  - Fault tolerant without intervention
  - Can be backed up
- Frangipani builds on top of Petal, which is a distributed storage service
  - Petal provides:
    - A virtual disk to clients that is written in blocks
    - 2^{64} bytes of storage
    - Optionally replicate data for availability
    - Snapshots for consistent backups
  - The scalability and fault tolerance comes primarily from Petal
  - Multiple disks can be used to make up the physical space behind the humongous Petal virtual disk, but everything appears as if they were on the same disk
  - Frangipani builds on top of Petal to add:
    - A file-like interface instead of disk-like interface
    - Coordination / sharing of the file system across different clients
  - Frangipani can then use multiple Petal virtual disks for even more space
    - All of this is abstracted away by the Petal driver

- Also builds on a distributed locking service for synchronizing access to files
  - Provides multiple-reader, single-writer locks to clients
- Designed to run in a cluster of machines all under the same administrator (relies on being able to trust each server)
  - The network connections must be made private
  - Outside programs should not be able to send instructions to Petal
- Generally the Petal and Frangipani can be run on the same machines
  - The distributed lock service is independent of the rest of the system, but they can still be run on the same machines
- The user can view Frangipani as just a normal mountable UNIX filesystem with the inodes that represent files or directories
- Downsides of using Frangipani as built on top of Petal:
  - Logging occurs twice, once in Frangipani and another in Petal
  - Frangipani can't use physical disk location information in placing data because Petal virtualizes it
  - Frangipani locks entire files and directories rather than blocks

Petal Disk Layout

- Petal has 2^{64} bytes of address space
  - Only commits physical space to virtual addresses that are written to
  - Provides a decommit primitive to free physical space backing virtual addresses
  - Commits and decommits space in chunks of 64 KB
- Frangipani divides up the virtual disk space as:
  - 1 TB for configuration parameters
    - In reality, only a few kilobytes are used
  - 1 TB for logs, spread among 256 regions
    - Each server will use one of these regions

        - This technically limits to 256 servers but this can easily be changed
    - 3 TB for allocation bitmaps to specify which blocks in the remaining regions are free
        - Each server uses a portion of this space for exclusive use
        - Allocation bitmaps point to both space for inodes and point to space in the rest of memory that is free for storing file data
    - 1 TB for inode space
        - Each inode is 512 bytes long, which is the same size as a disk block
            - This avoids conflicts of two servers attempting to access inodes in the same block
            - Has space for 2^{31} inodes
        - The allocation bitmap maps bits there to inodes here
            - This mapping is fixed, so each server only allocates inodes in its part of the allocation bitmap
            - This doesn't mean servers can't modify other inodes, it just means that each server when making new inodes only does it through here
    - 2^{47} bytes for small blocks
        - Each small block is 4 KB (2^{12} bytes)
        - The first 64 KB of a file is stored in small blocks
        - Anything past that is stored in a large block
    - The rest of the space is for large blocks
        - Each large block is 1 TB
- This leads to the following limitations:
    - $2^{24} \approx 16M$ large files
    - A file can be no larger than 1TB + 64KB
    - This can be easily modified
    - Each Frangipani file system uses only 1 Petal virtual disk, but multiple Frangipani file systems can be mounted by a single user

Logging and Recovery

- Each Frangipani server stores a log in Petal that stores metadata updates
    - When a server needs to make update, it first creates a record describing the update and appends it to its log
    - These log records are periodically written to Petal in the same order that they were requested
        - Rather than writing these periodically, we could allow them to be written synchronously, which offers better failure semantics at the cost of increased latency
        - Only after a log record is written to Petal does the server modify the metadata
- Logs are limited to be at most 128 KB in the given implementation
    - This is maintained as a circular buffer, so Frangipani reclaims the oldest 25% of log space whenever the log fills
    - All of these entries must be written to Petal before they are reclaimed
- If a server crashes, the system eventually detects the failure and runs recovery
    - Failure can be detected by either a client or by the lock service
    - Recovery then receives ownership of the log and locks and then carries out all log operations to ensure they are done
    - After log processing, the recovery demon releases all locks and frees the log
    - As long as Petal remains available, the system can tolerate any number of Frangipani failures
- Locking ensures that updates to the same data are serialized
- We keep a version number on every metadata block to ensure that we never replay a log record for an already completed update

Synchronization and Cache Coherence

- We use multiple-reader / single-writer locks to implement synchronization
    - A read lock allows a server to read the associated data and then cache it
        - If a server is asked to release a read-lock, it must invalidate its cache entry before doing so

    - A write lock allows a server to read or write the associated data and then cache it
        - A server's cached copy of a disk can only be different from the on-disk version if it holds the write lock
        - This data must be written to disk before it can release the lock
            - Can keep the entry in its cache if it is downgrading to reader
- Each log is a single lockable segment
- Bitmap space is divided into segments that are also locked
    - This protects space that hasn't been allocated yet
- Each file / directory / symbolic link is locked across the entire file
- Deadlock is avoided by globally ordering locks and acquiring them in two phases:
    - First a server determines what locks it needs (i.e. by looking up files in a directory, which may acquiring and releasing other locks)
    - Second it attempts to acquire all of the locks it needs in order
    - Afterwards, it checks if any of the objects it examined in phase 1 (i.e. directory contents) were modified after the lock was released
        - If they were modified, it releases all locks and restarts

Lock Service

- Generally all locks will be sticky
    - A lock will be held until it is asked to be released
    - This helps improve performance on rare locks
- Client failure is dealt with leases
    - When a client (Frangipani server) contacts the lock service, it obtains a lease with an expiration time
    - This must be renewed or else the lock service will consider the client as failed
- If a network failure occurs and a Frangipani cannot renew its lease, then the server has to discard all locks / data in its cache
    - If the cache was dirty, then all subsequent requests from the user must return an error
- The lock service must be fault tolerant to avoid issues
- There is one small hazard when a server lease expires:
    - If the server did not actually crash, it may try to access Petal even after its lease has expired
    - A Frangipani server checks that its lease is still valid before attempting any write to Petal
    - Petal doesn't do any checking when a write request arrives
    - Therefore if a lease check occurs, the write request is sent, the network is partitioned, write request is very delayed, then the write request could arrive when the lease had expired
        - What the authors did to bandaid fix this was to set a huge margin of time so we only send when we still have 15 seconds remaining in our lease
        - This doesn't fully solve the issue, which would require something like putting timestamps on write requests so Petal can ignore expired ones

Adding / Removing Frangipani Servers

- Very simple
- Added server is told which Petal virtual disk to use and where to find lock service
    - Afterwards, the lock service provides lease, and this ID is used to find where to work in Petal
- Removed server can just be shut off
    - Preferable if all dirty data is flushed and locks released
    - If not, then the recovery service will just grab this
- Petal and lock servers can be added as described in their implementations

Consistency Guarantees

- One possibility of a write saying that it is successful but not actually going off is if the write only goes to the cache
    - In Frangipani, the cache is only written to the Petal log when we lose the writer lock or periodically the cache is flushed
    - If the system crashes while still holding the writer lock, then this write could be loss forever