

6.824 2020 Lecture 11: Frangipani

Frangipani: A Scalable Distributed File System
 Thekkath, Mann, Lee
 SOSP 1997

why are we reading this paper?

- cache coherence
- distributed transactions
- distributed crash recovery
- and the interaction among the three

what's the overall design?

- a network file system
 - works transparently with existing apps (text editors &c)
 - much like Athena's AFS
- [users; workstations + Frangipani; network; petal]
- Petal: block storage service; replicated; striped+sharded for performance
- What does Frangipani store in Petal?
 - directories, i-nodes, file content blocks, free bitmaps
 - just like an ordinary hard disk file system
- Frangipani: decentralized file service; cache for performance

what's the intended use?

- environment: single lab with collaborating engineers
 - the authors' research lab
 - programming, text processing, e-mail, &c
- workstations in offices
- most file access is to user's own files
- need to potentially share any file among any workstations
 - user/user collaboration
 - one user logging into multiple workstations
- so:
 - common case is exclusive access; want that to be fast
 - but files sometimes need to be shared; want that to be correct
- this was a common scenario when the paper was written

why is Frangipani's design good for the intended use?

- strong consistency, which humans expect from a file system
- caching in each workstation -- write-back
 - all updates initially applied just in workstation's cache -- fast
 - including e.g. creating files, creating directories, rename, &c
 - updates proceed without any RPCs if everything already cached
 - so file system code must reside in the workstation, not server
- most complexity is in clients, not the shared Petal servers
 - more clients -> more CPU power
 - complex servers were a serious bottleneck in previous systems e.g. NFS, AFS

what's in the Frangipani workstation cache?

- what if WS1 wants to create and write /grades?
- read / information from Petal into WS1's cache
- add entry for "grades" just in the cache
- don't immediately write back to Petal!
 - in case WS1 wants to do more modifications

challenges

- WS2 runs "ls /" or "cat /grades"
 - will WS2 see WS1's writes?
 - write-back cache, so WS's writes aren't in Petal
 - caches make stale reads a serious threat
 - "coherence"
- WS1 and WS2 concurrently try to create /a and /b
 - in the same directory
 - will they overwrite each others' changes to /?
 - there's no central file server to sort this out!

"atomicity"

WS1 crashes while creating a file

many steps: allocate i-node, initialize i-node, update directory

leaves a mess if incomplete

how to ensure other workstations don't see the mess? how to clean up?

"crash recovery"

"cache coherence" solves the "read sees write" problem

the goal is linearizability AND caching

many systems use "cache coherence protocols"

multi-core, file servers, distributed shared memory

Frangipani's coherence protocol (simplified):

lock server (LS), with one lock per file/directory

file owner

x WS1

y WS1

workstation (WS) Frangipani cache:

file/dir lock content

x busy ...

y idle ...

if WS holds lock,

busy: using data right now

idle: holds lock but not using the cached data right now

workstation rules:

don't cache unless you hold the lock

acquire lock, then read from Petal

write to Petal, then release lock

coherence protocol messages:

request (WS -> LS)

grant (LS -> WS)

revoke (LS -> WS)

release (WS -> LS)

the locks are named by files/directories (really i-numbers),
though the lock server doesn't actually understand anything
about file systems or Petal.

example: WS1 changes file z, then WS2 reads z

WS1	LS	WS2
read z		
--request(z)-->		
	owner(z)=WS1	
<--grant(z)---		
(read+cache z data from Petal)		
(modify z locally)		
(when done, cached lock in state)		
		read z
		<--request(z)--
<--revoke(z)--		
(write modified z to Petal)		
--release(z)-->		
	owner(z)=WS2	
	--grant(z)-->	
		(read z from Petal)

the point:

locks and rules force reads to see last write

locks ensure that "last write" is well-defined

note:

before a workstation releases a lock on a file,

it must write modified file content to Petal, as well as meta-data

so workstations get coherence for file content, as well as meta-data
i.e. file reads see most recent content

coherence optimizations

the "state is already an optimization

Frangipani has shared read locks, as well as exclusive write locks

next challenge: atomicity

what if two workstations try to create the same file at the same time?

are partially complete multi-write operations visible?

e.g. file create initializes i-node, adds directory entry

e.g. rename (both names visible? neither?)

Frangipani implements transactional file-system operations:

operation corresponds to a system call (create file, remove file, rename, &c)

WS acquires locks on all file system data that it will modify

performs operation with all locks held

only releases when finished

only responds to a "revoke" after entire operation is complete

thus no other WS can see partially-completed operations

and no other WS can simultaneously perform conflicting updates

note Frangipani's locks are doing two different things:

cache coherence (revealing writes)

atomic transactions (concealing writes)

next challenge: crash recovery

What if a Frangipani workstation dies while holding locks?

other workstations will want to continue operating...

can we just revoke dead WS's locks?

what if dead WS had modified data in its cache?

what if dead WS had started to write back modified data to Petal?

e.g. WS wrote new directory entry to Petal, but not initialized i-node

this is the troubling case

Is it OK to just wait until a crashed workstation reboots?

Frangipani uses write-ahead logging for crash recovery

Before writing any of op's cached blocks to Petal, first write log to Petal

So if a crashed workstation has done some Petal writes for an operation,

but not all, the writes can be completed from the log in Petal

Very traditional -- but...

1) Frangipani has a separate log for each workstation

this avoids a logging bottleneck, eases decentralization

but scatters updates to a given file over many logs

2) Frangipani's logs are in shared Petal storage, not local disk

WS2 can read WS1's log to recover from WS1 crashing

Separate logs is an interesting and unusual arrangement

What's in the log?

log entry:

(this is a bit of guess-work, paper isn't explicit)

log sequence number

array of updates:

block #, new version #, addr, new bytes

just contains meta-data updates, not file content updates

example -- create file d/f produces a log entry:

a two-entry update array:

add an "f" entry to d's content block, with new i-number

initialize the i-node for f

initially the log entry is in WS local memory (not yet Petal)

When WS gets lock revocation on modified directory from LS:

1) force its entire log to Petal, then

2) send the cached updated blocks to Petal, then

3) release the locks to the LS

Why must WS write log to Petal before updating
i-node and directory &c in Petal?

Why delay writing the log until LS revokes locks?

What happens when WS1 crashes while holding locks?

Not much, until WS2 requests a lock that WS1 holds

LS sends revoke to WS1, gets no response

LS times out, tells WS2 to recover WS1 from its log in Petal

What does WS2 do to recover from WS1's log?

Read WS1's log from Petal

Perform Petal writes described by logged operations

Tell LS it is done, so LS can release WS1's locks

Note it's crucial that each WS log is in Petal so that it can
be read by any WS for recovery.

What if WS1 crashes before it writes its recent log to Petal?

WS1's recent operations may be totally lost if WS1 crashes.

But the file system in Petal will be internally consistent.

Why is it safe to replay just one log, despite interleaved
operations on same files by other workstations?

Example:

WS1: delete(d/f) crash

WS2: create(d/f)

WS3: recover WS1

WS3 is recovering WS1's log -- but it doesn't look at WS2's log

Will recovery re-play the delete?

This is The Question

No -- prevented by "version number" mechanism

Version number in each meta-data block (i-node) in Petal

Version number(s) in each logged op is block's version plus one

Recovery replays only if op's version > block version

i.e. only if the block hasn't yet been updated by this op

Does WS3 need to acquire the d or d/f lock?

No: if version number same as before operation, WS1 didn't

do the write, so it couldn't have released the lock,

so no-one else could have the lock, so

it's safe for WS3 safe to update in Petal

The log doesn't hold file *content* -- what are the consequences?

Workstations may send content writes to Petal in an order different
from the order in which application called write.

If no crash, this is hidden by locks, so it doesn't matter.

If crash, Petal may be left with a random subset of recent writes.

For many apps, this doesn't matter, since a crash is unlikely
to leave program output in a useful form anyway.

Careful programs (e.g. text editor, database) use fsync().

This behavior mimics e.g. Linux with a local disk.

What if:

WS1 holds a lock

Network partition

WS2 decides WS1 is dead, recovers, releases WS1's locks

If WS1 is actually alive, could it subsequently try to write data covered by the lock?

Locks have leases!

Lock owner can't use a lock past its lease period

LS doesn't start recovery until after lease expires

Is Paxos or Raft hidden somewhere here?

Yes -- Paxos-based configuration managers for lock servers, Petal servers
ensures a single lock server, despite partition
ensures a single primary for each Petal shard

Performance?

- hard to judge numbers from 1997
- do they hit hardware limits? disk b/w, net b/w
- does total throughput increase with more hardware?
 - does each added Frangipani workstation contribute more capacity as well as more load?
- what scenarios might we care about?
 - read/write lots of little files (e.g. reading my e-mail)
 - read/write huge files

Small file performance -- Figure 5

- X axis is number of active workstations
 - each workstation runs a file-intensive benchmark
 - workstations use different files and directories
- Y axis is completion time for a single workstation
- flat implies good scaling == no significant shared bottleneck
- presumably each workstation is just using its own cache
- possibly Petal's many disks also yield parallel performance

Big file performance

- each disk: 6 MB / sec
 - Petal stripes to get more than that
- 7 Petal servers, 9 disks per Petal server
 - 336 MB/s raw disk b/w, but only 100 MB/s via Petal
- a single Frangipani workstation, Table 3
 - write: 15 MB/s -- limited by network link
 - read: 10 MB/s -- limited by weak pre-fetch (?), could be 15
- lots of Frangipani workstations
 - Figure 6 -- reads scale well with more machines
 - Figure 7 -- writes hit hardware limits of Petal (2x for replication)

For what workloads is Frangipani likely to have poor performance?

- lots of read/write sharing?
- lots of small files?

Petal details

- Petal provides Frangipani w/ fault-tolerant storage
 - so it's worth discussing
- block read/write interface
 - compatible with existing file systems
- looks like single huge disk, but many servers and many many disks
 - big, high performance
 - striped, 64-KB blocks
- virtual: 64-bit sparse address space, allocate on write
 - address translation map
- primary/backup (one backup server)
 - primary sends each write to the backup
- uses Paxos to agree on primary for each virt addr range
- what about recovery after crash?
 - suppose pair is S1+S2
 - S1 fails, S2 is now sole server
 - S1 restarts, but has missed lots of updates
 - S2 remembers a list of every block it wrote!
 - so S1 only has to read those blocks, not entire disk
- logging
 - virt->phys map and missed-write info

Limitations

- Most useful for e.g. programmer workstations, not so much otherwise
 - A file system is not a great API for many applications, e.g. web site
- Frangipani enforces permissions, so workstations must be trusted
 - so Athena couldn't run Frangipani on Athena workstations
- Frangipani/Petal split is a little awkward
 - both layers log
 - Petal may accept updates from "down" Frangipani workstations

more RPC messages than a simple file server

Ideas to remember

- complex clients sharing simple storage -- maybe scalable
- cache coherence
- distributed transactions
- distributed crash recovery
- interactions among the above