$Peta{l} \quad C_1 \ldots C_n$

$disk \Rightarrow$ more b/w, more reliability limits

$\Rightarrow$ scale $\Rightarrow$ reconfig

$\Rightarrow$ usability $\Rightarrow$ block-based

File System: Frangipani

Layering

$\overset{\text{does work for storage?}}{\text{nets}}$ $\overset{\text{HTTP}}{\text{TCP}}$ $\overset{\text{IP}}{\text{ether}}$

(benefits? costs?)
Frangipani | "cluster" FS
---|---
Classic | FS
Cluster | NFS/AES

Client/"Servers" \(\rightarrow\) Lock service

Client-side: memory (cache) (consistency)
Crash recovery: logging (WAL)

[How to build a dist FS on a shared virt disk?]
C \rightarrow \text{Local FS} \leftarrow \text{Linux exty}

\Rightarrow \text{Petal}

\text{works \Rightarrow not a dist fs}

\text{Layout} \left[ \text{Data Structures} \right]

\text{Superblock: config info}

1 \text{ TB alloc, 32-bit 64-bit}

\text{only 3 KB used} \Rightarrow \text{tennis court} \Rightarrow \text{Europe}

\text{key aspect of Petal?}

\text{sparse, lazily allocated alloc: 64 KB}
Logs: one per client

=) crash recovery =) many logs

=) log is in Petal (not local to clients)

(others can access useful for recovery?)

File: up to 16 4KB blocks (small) + one additional 1TB block

Logging / Recovery:

=) each client =) own log (exclusive)

=) metadata only WAL
writes:

\[ \Rightarrow \text{buffer updates in client mem} \]
\[ \Rightarrow \text{fills: write in order to Petal Log} \]
\[ \Rightarrow \text{after: update metadata in place} \]

\[ \text{crash} \rightarrow \text{detect?} \]
\[ \text{real client lock service} \]
\[ \text{recover?} \]
\[ \text{recovery demon:} \]
Tricky: serialization of updates

\[ L_1, \quad L_2 \]

\[ \rightarrow \text{created file "foo" } v_1 \]

\[ \vdots \]

\[ \rightarrow \text{deleted file "foo" } v_2 \]

\[ \text{Time} \]

\[ \text{reply} \rightarrow \text{crash} \]

Solution: version \( \neq 5 \)

\[ \Rightarrow \text{ in log} \]

\[ \Rightarrow \text{ w/ metadata in FS} \]

one other problem: \{ have to be careful when reusing metadata \}
Monday: send email

Exam solution: email

Decide on last 2 weeks (→) ? until weekend

4th .... 11th x [ ]

(Final?) → Yes ?

No ?

Synch + Cache Coherence

multi-reader, single writer

locks

read: read lock
allows: cache block in mem, read
invalidate: ⇒ flush cache
write: write lock
   allows: cache, buffer
   release: \rightarrow downgrado 
            \nearrow to read

\rightarrow lose a altogether
\rightarrow flush dirty \rightarrow delete

Workload patterns?
(supported)
\rightarrow may be read sharing,
not much write sharing

Granularity: logical "segments"
            \rightarrow 1 lock/segment

\rightarrow log
\rightarrow bitmap split into
    \rightarrow N segments
    each client \rightarrow one
each file/dir (inode + data blocks) one lock each

Problem: Deadlock

operations that acquire 71 lock

= figure out which locks needed

= acquire in inode address order

Lock Service: (Leases)

each client -> one lease (all locks)
expires after 30 secs
Many implementations of lock service
=> distributed, replicated
(paxos)

Lease expiration:
=> $C_1$ has lease $LS \rightarrow C_1$, is dead
(writes: (and will be for $x$ secs))
check lease is still valid
do write
Performance

- Write fence in Petal

Layering

- Simpler to build
- Costs
- Write fence
- Read ahead

Storage

- Where layering is costly

\[
\begin{align*}
\Rightarrow & \quad \text{(=} \text{ double logging)}
\end{align*}
\]