LFS- Log Structured File System

Last class:

Crash Consistency

- fsck
- Journaling
  - Data J.
    1. Journal Write
    2. "Commit"
    3. Checkpoint
  - Metadata J.
    1. Journal metadata Write
    2. Journal Commit
    3. Checkpoint the metadata.

Observations

1. System memory (RAM) is growing.
   - Opt: FS writes
2. Seq. I/O vs Rand. I/O.
3. FS - did poorly for common workloads.
   e.g., create a file,
   Write a block of data.
   Writes:
   1. inode.
   2. ibmap
   3. dir inode + dir data
   4. dmap
   5. data.
4. FS - not RAID aware.

Single disk

RAID
LFS:

\[
\begin{array}{|c|c|c|c|c|}
\hline
D_F & I_F & D_F & I_F \\
A_0 & A_1 & A_2 & A_3 \\
\hline
\end{array}
\]

update the data in \( D_F \).

Segments

\[
\begin{array}{c}
\downarrow \\
W_1 \\
\downarrow \\
\downarrow \\
\downarrow \\
W_n \\
\downarrow \\
\end{array}
\]

Segment (4MB)

\[
\begin{array}{c}
W_1 \ldots W_n \\
\end{array}
\]

Main Memory

Inode Map (imap)

\[
\begin{array}{c}
\text{imap}[F \ldots, F[n]] \\
\text{cr} = A_2 \\
\text{phv}[A_0] = A_0 \\
\text{map}[E] = A_1 \\
\text{imap} \\
\end{array}
\]

0

A_0

A_1

A_2
How much to buffer?

\( T_{\text{pos}} \) - time to position, (s)

\( R_{\text{peak}} \) - max transfer rate (MB/s)

Write \( D \) MB of data.

\[
T_{\text{write}} = T_{\text{pos}} + \frac{D}{R_{\text{peak}}} \quad (1)
\]

\[
R_{\text{effective}} = \frac{D}{T_{\text{write}}} = \frac{D}{T_{\text{pos}} + \frac{D}{R_{\text{peak}}}} \quad (2)
\]

What we want?

\[
R_{\text{eff.}} = F \times R_{\text{peak}} \quad (0.9) \quad (90\%)
\]

\[
F \times R_{\text{peak}} = \frac{D}{T_{\text{pos}} + \frac{D}{R_{\text{peak}}}}
\]

\[
\Rightarrow \quad D = F \times R_{\text{peak}} \left( T_{\text{pos}} + \frac{D}{R_{\text{peak}}} \right)
\]

\[
\Rightarrow \quad D = \left( \frac{F}{1-F} \right) \times R_{\text{peak}} \times T_{\text{pos}}
\]

\[
F = 90\% = 0.9 \\
R_{\text{peak}} = 100 \text{MB/s} \\
T_{\text{pos}} = 10 \text{ms} \\
D = \frac{0.9 \times 100 \text{MB/s} \times 10 \text{ms}}{1} = 9 \text{MB}
\]
Directories?

/home/gerald/foo

inode # of file foo.

CR
DF
If
Dt
If
I dir
D dir
I dir
map
map[If]: A1
map[If]: A3

Read File F

A4, A3, A2, A4, A1, A0
DONE.

/a/b/C/ foo

map[F] = A11
map[Ci] = A1 A13

Recursive update problem – prevented!
Garbage Collection

1. modify/update a block

2. append a block to a file

Garbage

Mechanisms

Policy
Mechanism

Block Liveness

Segment Summary Block

\[ A_0 : (F, D) \]

\[ D_0 \]

\[ I_F \]

\[ \text{map}[F] = A_1 \]

\[ \text{SS} \]

\[ \text{Addr} : (\text{inode \#, offset}) \]

\[ (N, T) = \text{Segment Summary}[A] \]

\[ \text{inode} = \text{Read}(\text{imap}[N]) \]

\[ \text{if} \ (\text{inode}[T] = A) \]

\[ \text{// live block} \]

\[ \text{else} \]

\[ \text{// garbage!} \]
Policies
Which block(s) to garbage collect?

Segments
- Hot: freq. accessed, updated
- Cold: NOT freq. updated

Crash?

CR | Do | If | imap | CR

Written only after 30s.

Start time | CR | End ts.

(last addr)