CS244b - GFS

Learning Goals:

- Application/infrastructure co-design
- Restraint
API

Create
Snapshot
Read
Write
Append at least once
Find Matching Files
Delete
<table>
<thead>
<tr>
<th></th>
<th>Write</th>
<th>Record Append</th>
</tr>
</thead>
<tbody>
<tr>
<td>Serial success</td>
<td><em>defined</em></td>
<td><em>defined</em> interspersed with</td>
</tr>
<tr>
<td></td>
<td><em>consistent</em> but <em>undefined</em></td>
<td><em>inconsistent</em></td>
</tr>
<tr>
<td>Failure</td>
<td></td>
<td><em>inconsistent</em></td>
</tr>
</tbody>
</table>

**Table 1: File Region State After Mutation**

*undef* - data could be anything

*inconsistent* -
Master State

File Name → Chunks, Locks

Chunk → Replicas, Version, Reference Count, Lease

Log* Checkpoints*
Chunk Server State

Chunks 64MiB

Version #, checksum
serial num., leases
Figure 1: GFS Architecture
Figure 2: Write Control and Data Flow
Full path → metadata, chunks, lock

/home/user/foo

Create /home/user/foo

Locks R W
Snapshots $F_1 \rightarrow F_2$

Revoke leases

Replicate filename mapping for $F_2$

Increase chunk ref counts.

Copy on write

Tell chunk servers to make local copy
Figure 3: Aggregate Throughputs.
incremental checksum

1 2 3

bottlenecks

Master
Table 3 also shows that the rate of operations sent to the master was around 200 to 500 operations per second. The master can easily keep up with this rate, and therefore is not a bottleneck for these workloads.

In an earlier version of GFS, the master was occasionally a bottleneck for some workloads. It spent most of its time sequentially scanning through large directories (which contained hundreds of thousands of files) looking for particular files. We have since changed the master data structures to allow efficient binary searches through the namespace. It can now easily support many thousands of file accesses per second. If necessary, we could speed it up further by placing name lookup caches in front of the namespace data structures.