Scalable Metadata Service in HDFS

Lin Xiao

Wittawat Tantisiriroj, Garth Gibson

PARALLEL DATA LABORATORY
Carnegie Mellon University
HDFS architecture

Client

write 64MB

NameNode

create(pathname)

addBlock(pathname)

Datanodes[]

DataNode

DataNode

DataNode

In memory metadata

Blocks Received

create(pathname)
Single metadata server limitation

- One node is throughput limited
  - Especially problematic for small files

- Total number of files is limited by memory size
  - In memory necessary for fast access
Our goals

• Remove the limit on number of files & blocks
  • Store metadata in memory & disk
• Distribute metadata service
  • Increase the throughput without hot spot
• Keep low latency
  • When metadata fit in memory, achieve good perf
• High availability and fault tolerance

• Apply techniques to scale HDFS namenode
Related work

• Shared-disk with distributed lock: GPFS
• Static namespace partition
  • Sub-tree: Yahoo! Federation HDFS, PanFS
  • Round robin: PVFS
• Dynamic partition
  • Sub-tree: Ceph (on recent workload)
  • Directories as objects: Ursa Minor
  • Hash: Giga+
• Table for metadata: Colossus
What shall we do?

- Table stores removes HDFS limitation with parallel storage, name and block managers
What to keep in a scalable DB?

• Namespace and file metadata table
  • Pathname to row key mappings
    – May cause multiple RPCs to open a file
  • Column families: attributes, data locations
• Block metadata and location table
  • Various granularity of managing blocks
• Other tables
  • Datanode status
  • Quota
  • Both are constantly changing
Row key choices (1)

- Parent Inode + file name
  - Rename only changes inode
  - Multiple lookups for each file
  - Multiple RPCs

- Hash (path name)
  - Load balanced
  - No Locality

<table>
<thead>
<tr>
<th>Path</th>
<th>Key</th>
</tr>
</thead>
<tbody>
<tr>
<td>/</td>
<td>0</td>
</tr>
<tr>
<td>/a</td>
<td>0+a</td>
</tr>
<tr>
<td>/b</td>
<td>0+b</td>
</tr>
<tr>
<td>/a/f1</td>
<td>1+f1</td>
</tr>
<tr>
<td>/a/f2</td>
<td>1+f2</td>
</tr>
</tbody>
</table>
Row key choices (2)

- Full pathname
  - One table lookup for each file
  - Cannot just scan for just children
  - Rename a directory needs to rename every file

```
/   
/   /a  /b
  /a/f1  /a/f2
  /a/f1  /a/f2
  /b
```

<table>
<thead>
<tr>
<th>Pathname</th>
<th>Key</th>
</tr>
</thead>
<tbody>
<tr>
<td>/</td>
<td>/</td>
</tr>
<tr>
<td>/a</td>
<td>/a</td>
</tr>
<tr>
<td>/a/f1</td>
<td>/a/f1</td>
</tr>
<tr>
<td>/a/f2</td>
<td>/a/f2</td>
</tr>
<tr>
<td>/b</td>
<td>/b</td>
</tr>
</tbody>
</table>
Row key choices (3)

- Directory depth + full path name
  + Locality for every directory

Which one is better?

<table>
<thead>
<tr>
<th>Pathname</th>
<th>Key</th>
</tr>
</thead>
<tbody>
<tr>
<td>/</td>
<td>0+/</td>
</tr>
<tr>
<td>/a</td>
<td>1+/a</td>
</tr>
<tr>
<td>/b</td>
<td>1+/b</td>
</tr>
<tr>
<td>/a/f1</td>
<td>2+/a/f1</td>
</tr>
<tr>
<td>/a/f2</td>
<td>2+/a/f2</td>
</tr>
</tbody>
</table>
Opencloud operation stats

- Most operations are open
Pathnames tend to be deep

CDF

Pathname Length
Challenges

• Bootstrapping
  • Original HDFS for the scalable DB?
• Reduce latency caused by distributing metadata
  • Collocate processes with tablet server
  • Send requests in parallel
• Efficient use of memory
  • Memory overhead compared to customized service
• Constantly changing table
  • Datanode status, Quota
• Failure handling