CS554: FusionFS Basics

Dongfang Zhao

September 2013
Motivation

- How to achieve scalable I/O throughput for extreme scale distributed systems. Examples:
  - In cloud computing, metadata is a bottleneck
  - In high-performance computing, storage is remotely connected
Goal

• Propose a new storage architecture to achieve high I/O throughput at extreme scale:
  – Metadata management
    • Co-locate to computation
    • Evenly distributed on compute nodes
  – Data movement
    • Co-locate to computation
    • Write: to local disk; rebalancing
    • Read: schedule the job to the data; or transfer the data
Roles

A 10-node cluster

Node 1  Node 2  Node 3  Node 4  Node 5

Node 6  Node 7  Node 8  Node 9  Node 10

Client

Metadata Server

File name: ~/foo1
Location: Node 1
Owner: root

Data Server

~/foo9

```
dongfang@datasy:~$ ls -alh
总共有 172k
drwxr-xr-x 20 dongfang users 4.0K
drwxr-xr-x 43 root root 4.0K
-rw------- 1 dongfang users 18K
-rw-r--r-- 1 dongfang users 1.2K
drwxr-xr-x 2 dongfang users 4.0K
```
Architecture

Node 1

Local Write

Local Storage

Partial Metadata

Files

Remote Read

Node 2

Local Write

Local Storage

Partial Metadata

Files

Job Scheduler

Global Name Space

Periodical Load Rebalance
## Distributed Hash Table

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>~/</td>
<td>drwxrwxr-x; 4.0K; ~/homedir/subdir</td>
</tr>
<tr>
<td>~/homedir</td>
<td>drwxrwxr-x; 4.0K; ~/homedir/subdir, ~/homedir/homefile</td>
</tr>
<tr>
<td>~/homedir/subdir</td>
<td>drwxrwxr-x; 4.0K; ~/homedir/subdir/subfile</td>
</tr>
<tr>
<td>~/homedir/homefile</td>
<td>-rw-rw-r--; 423M; Node 1</td>
</tr>
<tr>
<td>~/homedir/subdir/subfile</td>
<td>-rw-rw-r--; 133M; Node 2</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Resilience

**Membership Table**

<table>
<thead>
<tr>
<th>Start</th>
<th>Node</th>
<th>Entries</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>A</td>
<td>1, 2, 3</td>
</tr>
<tr>
<td>4</td>
<td>B</td>
<td>6</td>
</tr>
</tbody>
</table>

**Stage 1: initial state**

<table>
<thead>
<tr>
<th>Start</th>
<th>Node</th>
<th>Entries</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>A</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>C</td>
<td>2, 3</td>
</tr>
<tr>
<td>4</td>
<td>B</td>
<td>6</td>
</tr>
</tbody>
</table>

**Stage 2: Node C joins**

<table>
<thead>
<tr>
<th>Start</th>
<th>Node</th>
<th>Entries</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>A</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>C</td>
<td>2, 3</td>
</tr>
<tr>
<td>4</td>
<td>A</td>
<td>6, 1</td>
</tr>
</tbody>
</table>

**Stage 3: Node B leaves**

<table>
<thead>
<tr>
<th>Start</th>
<th>Node</th>
<th>Entries</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>A</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>C</td>
<td>2, 3</td>
</tr>
<tr>
<td>4</td>
<td>A</td>
<td>6, 1</td>
</tr>
</tbody>
</table>
Data Movement

• Design principle
  – Always write to the local compute node
    • Pro: maximal aggregate throughput
    • Con: bad load balance
      – Possible solution: Asynchronous rebalancing?
  – No locality-awareness for data read
    • Transfer from the node that has the file
      – Maybe it is just loopback if we are lucky
      – Can we schedule the job on the node with the required data?
    • Then read locally
Data Replication

- **Design principle**
  - Option 1: traditional replicas
    - Pro: ease-of-use
    - Con: low storage utilization
  - Option 2: information dispersal algorithms (e.g. erasure coding)
    - Pro: small storage overhead
    - Con: compute intensive
    - Solution: GPU acceleration?
Data Provenance

• Design principle
  – Provenance data saved on compute nodes
  – Independent to file metadata
  – Provide a hybrid of fine- and coarse-grained levels of provenance
    • A summary of block-level provenance, as well as the file-level provenance
Data Compression

• Design principle
  – Customizable: users should be able to specify/choose which compression algorithms to be applied
  – Resilient: should apply different strategies for different type/size of files
  – Transparent: the compression should not require any modification to the applications and/or high level I/O libraries
Implementation

- Implementation
  - C, C++, Shell script, FUSE, socket, Pthread
FusionFS Projects

• More information from Prof. Raicu
• My suggestions in general:
  – Strong programming skills in
    • System programming in C
    • Scripting language, e.g. Shell script, perl, python
  – Familiar with at least one distributed file system, e.g. HDFS, GPFS, Lustre, PVFS
  – Knowledgeable in distributed systems and operating systems (CS550 is a good sign)
  – Comfortable with Linux terminals, e.g. vi, ssh, scp
Questions

• FusionFS document:  

• Mentor contact:
  – Name: Dongfang Zhao
  – Office: SB #002
  – Email: dzhao8@hawk.iit.edu