HY590.45
Modern Topics in Scalable Storage Systems

Kostas Magoutis
magoutis@csd.uoc.gr
http://www.csd.uoc.gr/~hy590-45
Petal: Client view

- LFS (Petal Client)
- NT FS (Petal Client)
- PC FS (Petal Client)
- BSD FFS (Petal Client)

Scalable Network

Petal

/dev/vdisk11
/dev/vdisk2
/dev/vdisk3
/dev/vdisk4
/dev/vdisk5

A virtual disk
Goals of Petal

- Tolerate and recover from any component failure
- Geographically distribute to tolerate site failures
- Transparently reconfigure to expand, balance load
- Dynamically balance load
- Fast and efficient support for backups and recovery
Physical structure
Virtual to physical mapping

\(<\text{vdiskID, offset}> \rightarrow \langle \text{serverID, diskID, diskOffset} \rangle\)
Support for backup

• Snapshots
  – Petal can quickly create an exact copy of a virtual disk at a specified point in time by using copy-on-write techniques
  – A snapshot is like any other vdisk, but cannot be modified
  – VDir: vdiskID → <global-map-identifier, epoch-number>

• Crash-consistent snapshot
  – Similar to disk image left after an application crash
Reconfiguration

• Dynamic change in vdisk # of servers, redundancy

• How is it performed
  – Create new GMap with desired redundancy, server mapping
  – Change VDir entries that refer to old GMap to new one
  – Redistribute data to the servers according to new GMap, requiring substantial amounts of network and disk traffic
  – Read requests will be tried on new GMap first, then the old GMap if the translation has not yet been transferred
  – Writes are always performed on the new GMap

• Improve efficiency via fencing
Chained de-clustering

Virtual Disk

Server 0
- D0
- D3
- D4
- D7

Server 1
- D1
- D0
- D5

Server 2
- D2
- D1
- D6

Server 3
- D3
- D2
- D7
- D6
Data access and recovery

• How are reads performed

• How are writes performed
Data access and recovery

Read protocol
1. Try primary; if live, get read lock
2. If down, try secondary; if live, get read lock
3. Return block contents, release read lock

Write protocol
1. Contact primary
2. If alive, mark block busy in stable storage
3. Primary apply request locally and simultaneously send write to secondary
4. When both complete, clear busy bit, respond to client
5. If primary crashes during request, busy bit is used to recover later on
6. If primary dead, start from secondary
7. (Secondary checks primary indeed crashed)

If primary or secondary is down, on live node:
1. Mark data as *stale* before writing to disk
2. During recovery, make replicas consistent by exchanging *dirty-region log*
a read will see a state at least as recent as that produced by the most recently completed write that completed before the read started
Failure-free operation

if some reader sees the results of a particular write, then any reader that starts after that reader finishes will also see a result at least that recent.
Recovery from primary crash

write(b,6)

Primary

c1

write

busy

crash

Secondary

c2

write

check status

reconcile

restart

clear busy bit

ack

write

busy

write

ack
Recovery from primary crash

- Primary
  - write(b,6)
    - busy
    - crash
      - X
    - reboot
      - get dirty-region log
      - reconcile
      - busy
      - clear busy bit
      - ack
  - write
  - ack

- Secondary
  - write(b,5)
    - primary dead??
    - set stale
    - write
    - ack
  - write
  - busy
  - ack
Recovery from secondary crash

write(b,6) → Primary
write(write(busy)
set stale
write
clear busy bit
ack

backup dead??

write(b,5) → Secondary
write(crash)
write(busy)
get dirty-region log
reconcile
replay write
restart
clear busy bit
write
ack

ack

Primary
Secondary
Petal cannot handle partitions

write(b,5)

write(b,6)

write

set busy

backup dead??

set stale

write

clear busy bit

ack

partition

primary dead??

write

set stale

write

ack

write(b,5)
Latency, throughput results

<table>
<thead>
<tr>
<th>Client Request Latency (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Request</td>
</tr>
<tr>
<td>---------</td>
</tr>
<tr>
<td>512 byte Read</td>
</tr>
<tr>
<td>8 Kbyte Read</td>
</tr>
<tr>
<td>64 Kbyte Read</td>
</tr>
<tr>
<td>512 byte Write</td>
</tr>
<tr>
<td>8 Kbyte Write</td>
</tr>
<tr>
<td>64 Kbyte Write</td>
</tr>
</tbody>
</table>

Table 1: Latency of a Chained-Declustered Virtual Disk

<table>
<thead>
<tr>
<th>Aggregate Throughput (RZ29 Log)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Request</td>
</tr>
<tr>
<td>---------</td>
</tr>
<tr>
<td>512 byte Read</td>
</tr>
<tr>
<td>8 Kbyte Read</td>
</tr>
<tr>
<td>64 Kbyte Read</td>
</tr>
<tr>
<td>512 byte Write</td>
</tr>
<tr>
<td>8 Kbyte Write</td>
</tr>
<tr>
<td>64 Kbyte Write</td>
</tr>
</tbody>
</table>

Table 2: Normal and Failed Throughput of a Chained-Declustered Virtual Disk