An Efficient Data Location Protocol for Self-Organizing Storage Clusters

Hong Tang and Tao Yang*
Department of Computer Science
University of California, Santa Barbara
*Also affiliated with Ask Jeeves/Teoma

Cluster-Based Storage Cluster for Data-Intensive Applications and Services

- Data-intensive applications and services:
  - Scientific computing: CERN.
  - Internet services: Google, Yahoo, Teoma.
  - Data grid computing: neuroscience data analysis.

- Cluster-based storage systems: a cluster of PCs or workstations will locally attached disks.
  - Cost effective.
  - Incrementally expandable.
  - Straightforward to exploit application parallelism.

Data Location Protocol

- Objectives:
  - Organize distributed storage as virtual volumes.
  - Maintain the information of how file data are distributed on physical storage nodes.

- Previous research:
  - Fixed mapping or modulus-based hashing from logical volume addresses to physical nodes.
  - Global reconfiguration and data migration is required during node additions or departures.

- This work (part of the Sorrento project):
  - Self-organize available resources into virtual volumes.
  - Automatically adapt to node additions and departures.

Talk Outline

- Motivation.
- Problem statement.
- A taxonomy of data location schemes.
- Differentiated data location protocol.
- Experimental study.
- Related work and conclusions.

Background: Data Organization

- A logical file is represented by a tree of physical blocks.
- Index and data blocks.
- Variable-length blocks.
- Blocks are uniquely identified by BlockIDs.
- A logical file is uniquely identified by the root BlockID.
- Namespace management (filename -> root BlockID) is an orthogonal issue.
  [Anderson+00, Brend+03, Schmuck+02]

Problem Statement

\[ F : \mathcal{A} \rightarrow 2^\mathcal{A} \]

\( \mathcal{A} \): BlockID address space.
\( \mathcal{B} \): Set of storage nodes.

- Given a BlockID \( b \in \mathcal{A} \), function \( F \) returns a set of storage nodes \( \mathcal{B} \) that store \( b \), or \( \emptyset \) if \( b \) does not exist.

- Evaluation Metrics:
  - Data access performance: how fast to locate a block.
  - Storage utilization: the percentage of effectively usable storage.
  - Maintenance overhead:
    - Memory to keep the states of \( F \).
    - Data migration caused by the change of \( F \).
A Taxonomy of Data Location Schemes (I)

- How data are placed on cluster nodes?
  - Controllable: blocks can be placed on any nodes.
  - Uncontrollable: the location of a block is derived based on some hash functions.

<table>
<thead>
<tr>
<th>Controls and placement</th>
<th>Controllable placement</th>
<th>Uncontrollable (hashing-based) placement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Examples</td>
<td>GMAP, CAN, Chord.</td>
<td>Modulus-based hashing, LH, Consistent Hashing.</td>
</tr>
</tbody>
</table>

A Comparison of Data Location Protocols

- Variables:
  - N: number of blocks.
  - H: number of hosts.
  - S: total data volume.

<table>
<thead>
<tr>
<th>Memory</th>
<th>Replicated mapping table</th>
<th>Partitioned mapping table</th>
<th>Hashing-based placement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lookup (CPU)</td>
<td>O(N)</td>
<td>O(N/H)</td>
<td>O(1) or O(H)</td>
</tr>
<tr>
<td>Lookup (Net)</td>
<td>O(logN)</td>
<td>O(logN)</td>
<td>O(1) to O(logH)</td>
</tr>
<tr>
<td>Migration</td>
<td>0</td>
<td>0</td>
<td>O(1/N)</td>
</tr>
<tr>
<td>Space waste?</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Solution: Differentiated Data Location

- Small and large files/blocks are different [Baker+91, Vogels+99]:
  - User activities, life time,
  - Particularly: “Most files in a file system are small, while the majority of data are in large files.”

- Manage small blocks through a hash-based protocol:
  - Fast access performance, low memory overhead.
  - Reduced data migration overhead and space waste.

- Manage large blocks through a replicated mapping protocol:
  - Fast access performance, no migration overhead.
  - Reduced memory overhead.

Background: Consistent Hashing and Bloom Filters

- Consistent hashing [Karger+97]:
  - Mapping from BlockID addr space \( \mathcal{S} \) to the host set \( \mathcal{H} \).
  - Maintains stable hashing results for the majority of BlockIDs in \( \mathcal{S} \) when \( \mathcal{H} \) slightly changes.
  - Basic idea: establish distance relationship between \( \mathcal{S} \) and \( \mathcal{H} \) and map a BlockID to the “closest” host.
  - A client needs to maintain a view of the host set \( \mathcal{H} \).

- Bloom filters [Bloom 70]:
  - A bit array that encodes a set of BlockIDs.
  - Operations: insert, delete, test.
  - One order of magnitude reduction of memory overhead.
  - Low probability to give wrong answers (false positive).
  - A client needs to maintain a Bloom filter, one for each storage node in the system.
**How Large is Large?**

- Problem: what should be the threshold to determine small and large blocks?
  - Static threshold cannot reflect application behavior.
  - Global threshold may be hard to keep consistent.
- Solution: Host-specific threshold adjusted by a Slide-bar.

**Data Placement**

```
func Place_block (block b) {
  h = Ot_hash (servers, b.blockID);
  // Is h managed by consistent hashing?
  if (h.threshold >= h.size) {
    // yes (because h's size is below h's threshold).
    return h;
  }
  else {
    // no.
    // Find candidate hosts that can store the block.
    candidates = find h * servers
    where (h.threshold < h.size);
    // select one host from the candidates
    // based on some placement policy.
    h = Placement_policy (candidates, b);
    return h;
  }
}
```

**Data Location**

```
func locate_block (block b) {
  h = Ot_hash (servers, b.blockID);
  // Is h managed by consistent hashing?
  if (h.threshold >= h.size) {
    // yes (because h's size is below h's threshold).
    return h;
  }
  else {
    // no.
    // do the lookup using Slave filters.
    h = Slave Lookup (b.blockID);
    return h;
  }
}
```

**Handling Server Joins**

- Problem: Mapping inconsistency during block migration.
- Solution: (1) Exclude the not-found host and retry consistent hashing; (2) multicast query as backup choice.

```
func search_block (block b) {
  for e retry = max retry:
    h = Ot_hash (servers, b.blockID);
    if (h.threshold >= h.size) {
      // yes (because h's size is below h's threshold).
      return h;
    }
    else {
      // no.
      servers.remove (h);
      e.retry++;
      continue;
    }
  return h;
}
```

**Replication Support**

- Replicas of a block will have same BlockID.
- Large block:
  - Replicas are placed on different servers.
  - Slave Lookup will return multiple hosts.
- Small block:
  - Replicas are placed (located) on the BlockID’s r nearest hosts. (r: replication degree)
- Issues beyond data location protocol:
  - Replica consistency: master/slave, two-phase commit.
  - Concurrency control: locking, leasing, versioning, predicated-updates, merging procedures.

**Evaluation Settings**

- Evaluation methodology: trace-driven simulation.
  - No need to rewrite applications.
  - Faster and cheaper than running real applications.
- Trace collection:
  - Method: modify syscall wrappers in GLIBC library.
  - Accurate, fine grain timing for operations.
- Workload:
  - Page indexing from 30 machines for 15 days (production environment, Ask_jeeves/Teoma).
  - 5 million files, 1.6TB data created.
  - 5.5 million creations, 0.52 million deletions, 72.3 million lookups.
Effectiveness of Differentiated Protocol Design

- CH: consistent hashing.
- BLOOM: usage-based placement + Bloom-filter-based tracking.
- Diffloc: differentiated data location.

<table>
<thead>
<tr>
<th>Metric</th>
<th>CH</th>
<th>BLOOM</th>
<th>Diffloc</th>
</tr>
</thead>
<tbody>
<tr>
<td>Storage utilization</td>
<td>87.2%</td>
<td>98.7%</td>
<td>97.7%</td>
</tr>
<tr>
<td>Migration overhead</td>
<td>53.0GB</td>
<td>0</td>
<td>1.14GB</td>
</tr>
<tr>
<td>Memory consumption</td>
<td>24KB</td>
<td>8.3MB</td>
<td>190KB</td>
</tr>
</tbody>
</table>

Data Access Efficiency

![Graph showing data access efficiency]

Workload Redistribution after a Node Failure

- Settings:
  - Each block is replicated twice.
  - Server 7 experiences a failure.
- How does the workload destined to server 7 is redistributed to the remaining servers?

![Graph showing workload redistribution]

Related Work

- Distributed and parallel file systems:
  - AFS/Coda, Petal/Prangipani, Zebra/xFS.
  - PVFS, Google FS.
- Distributed hash table (DHT) for P2P content network:
  - CANS, Chord, Tapestry.
- Other hashing-based data location schemes:
  - Linear Hashing [Litwin 93, 96, 00].
  - SIEVE [Brinkmann+ SPAA 02].
  - Honicky et. al [IPDPS 03].
- Data location using Bloom filters:
  - Summary Cache.
  - OceanStore.

Conclusion

- A taxonomy of data location protocols.
- Differentiated data placement and location:
  - Small blocks are placed and located through consistent hashing.
  - Large blocks are placed through a usage-based policy, and their locations are tracked by Bloom filters.
- High storage utilization and low management overhead.
- Can be extended to support replication.
- Part of the Sorrento project: