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.

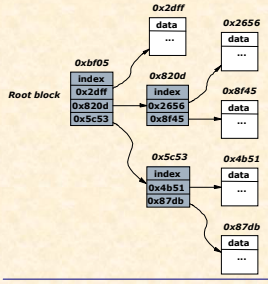
Data Location Protocol

- 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, Brandt+03, Schmuck+02]

Problem Statement

$$\mathcal{F} : \mathcal{B} \rightarrow 2^{\mathcal{H}}$$

$\mathcal{B}$: BlockID address space.
$\mathcal{H}$: Set of storage nodes.

- Given a BlockID $b \in \mathcal{B}$, function $\mathcal{F}$ returns a set of storage nodes $H \subseteq \mathcal{H}$ that store b, or ø 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 $\mathcal{F}$.
    - Data migration caused by the change of $\mathcal{H}$.

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.

	Controllable placement	Uncontrollable (hashing-based) placement
Strengths and weaknesses	High storage utilization. No migration cost. High memory cost.	Possible storage waste. High migration cost. Low memory cost.
Examples	GMAP, CANS, Chord.	Modulus-based hashing, LH*, Consistent Hashing.

A Taxonomy of Data Location Schemes (II)

- How is the mapping table maintained on cluster nodes?
  - Replicated: all cluster nodes have a copy of the table.
  - Partitioned: the table is partitioned among cluster nodes.

	Replicated mapping table	Partitioned mapping table
Strengths and weaknesses	Fast access performance. High memory cost.	Requires a network indirection to locate data. Lower and more scalable memory overhead.
Examples	GMAP, Bloom Filters	PMAP, CANS, Chord

A Comparison of Data Location Protocols

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

	Replicated mapping table	Partitioned mapping table	Hashing-based placement
Memory	*O(N)*	*O(N/H)*	*O(1)* or *O(H)*
Lookup (CPU)	*O(logN)*	*O(logN)*	*O(1)* to *O(logH)*
Lookup (Net)	0	*O(1)* to *O(logH)*	0
Migration	0	0	*O(S/H)*
Space waste?	No	No	Yes

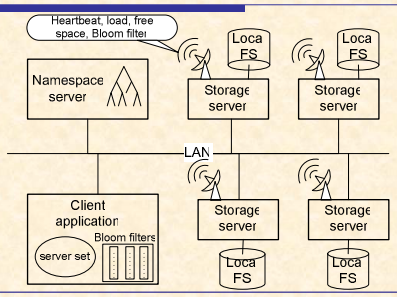
Solution: Differentiated Data Location

- 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

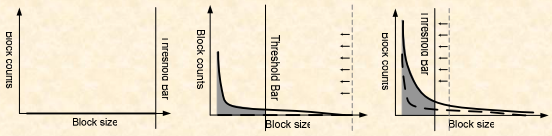
- Consistent hashing [Karger+97]:
  - Mapping from BlockID addr space $\mathcal{B}$ to the host set $\mathcal{H}$.
  - Maintains stable hashing results for the majority of BlockIDs in $\mathcal{B}$ when $\mathcal{H}$ slightly changes.
  - Basic idea: establish distance relationship between $\mathcal{B}$ 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 *H* Bloom filters, one for each storage node in the system.

System Overview



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*.



The Slide-bar Algorithm

Data Placement

```
Host Place_block (Block b)
{
   // Should b be placed using consistent hashing?
   Host h = CH_hash(servers, b.BlockID);
   if ( h.threshold >= b.size ) {
      // Yes (because b's size is below h's threshold).
      return h;
   }
   else { // No.
      // Find candidate hosts that can store the block.
      set<Host> candidates = find h in servers
                             where (h.threshold < b.size);
      // Select one host from the candidates
      // based on some placement policy.
      h = Placement_policy(candidates, b);
      return h;
   }
}
```


Data Location

```
Host Locate_block (Block b)
{
   Host h = CH_hash(servers, b.BlockID);

   // Is b managed by consistent hashing?
   if ( h.threshold >= b.size ) {
      // Yes (because b's size is below h's threshold).
      return h;
   }
   else { // No.
      // Re-do the lookup using Bloom filters.
      h = BLOOM_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.

```
Host Locate_block_modified (Block b)
{
   int n_retry=0, max_retry=3;
   while (n_retry < max_retry) {
      Host h = CH_hash(servers, b.BlockID);
      if ( h.threshold >= b.size ) {
         if ( h does not have b ) {
            servers.remove(h);
            n_retry++; continue;
         }
         else
            return h;
      }
   }
   h = BLOOM_lookup(b.BlockID);

   if (!h.valid) {
      h = mcast_lookup(b.BlockID);
   }
   return h;
}
```


Replication Support

- Replicas of a block will have same BlockID.
- Large block:
  - Replicas are placed on different servers.
  - `Bloom_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

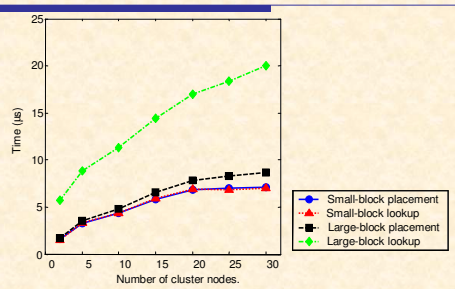
- Evaluation methodology: trace-driven simulation.
  - No need to rewrite applications.
  - Faster and cheaper than running real applications.
- Trace collection:
  - Method: modify syscall wrappers in GLIBC library.
  - Accurate, fine grain timing for operations.
- Workload:
  - Page indexing from 30 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.

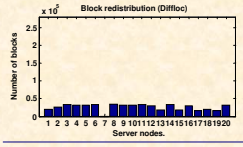
Metric	CH	BLOOM	Diffloc
Storage utilization	87.2%	98.7%	97.7%
Migration overhead	53.0GB	0	1.14GB
Memory consumption	24KB	8.3MB	190KB

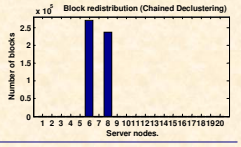
Data Access Efficiency



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?





Related Work

- Distributed and parallel file systems:
  - AFS/Coda, Petal/Frangipani, 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:

http://www.cs.ucsb.edu/projects/sorrento/