A Bit –Window based Algorithm for Balanced and Efficient Object Placement and Lookup in Large-Scale Object based Storage Cluster

Renuga Kanagavelu
Data Storage Institute, Singapore

A*STAR DSI

Outline

- Introduction
- Bit-Window based algorithm
- Object Placement
- Object Lookup
- Node Addition
- Node Deletion
- Performance Study
- Conclusions

Introduction

- **To develop a novel and efficient method based on Bit-Windows for object placement and lookup services**

- To distribute objects evenly across the storage nodes

- To reduce message overhead and access delay
  - No multi-hop messages

- To support node addition and deletion with low number object migrations

- To be scalable

- To reduce message overhead, no need to maintain information about neighbor nodes

Bit-Window Based Algorithm

➢ **The Bit-Window algorithm maps objects to storage nodes**

- An m-bit object identifier is produced by hashing the object
- SHA-1 is used as a basic hash function
- N : the total number of storage nodes with index from 0 to N-1
- Bit -window size $k = \lceil \log_2 (N) \rceil$. The size of the bit window depends on the number of the storage nodes. Due to addition or deletion of nodes, bit-window size may vary.
- The m-bit object identifier is divided into a number of bit windows of size k.
- The bit -windows are labeled starting from 0 from the right end, denoted as BW_0, $BW_1 \ldots BW_{v-1}$, where v is the maximum number of windows used

Object Placement

Bit-Window Algorithm

Input: O: Object Key
v: Number of bit-windows
N: Number of Storage nodes
Output: A valid node Index

Value ← BW_0
If (Value < N)   //valid node index

return Value
i← 1
While (i < v)
{
  Value = BW_i
  if (Value < N)
  return Value
  else i← i+1
}
$k = \lceil \log_2(N) \rceil$
Value = $BW_0 - 2^{k-1}$
return Value

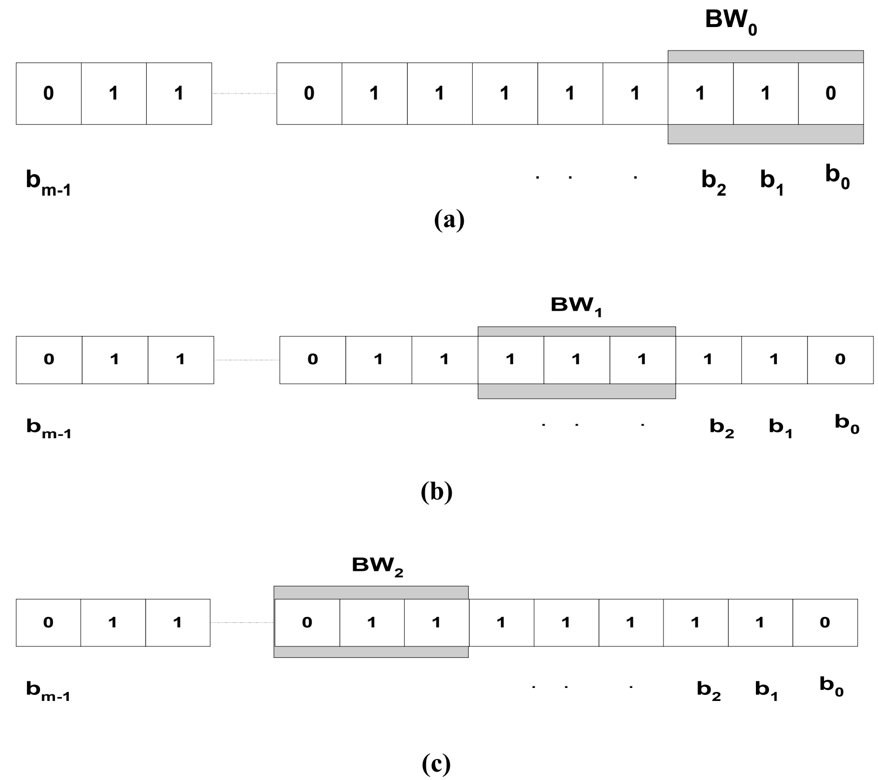

Figure : Illustration of Bit –Window algorithm (N=5). (a) BW_0 is searched. (b) BW_1 is searched. (c) BW_2 is searched.

Object Placement and Lookup

Object Placement

X : Total number of objects
N : The number of (valid) nodes
k : bit-window size
$P = 2^k$
$R = P - N$: Number of invalid nodes. ($R < N$)

Theorem-1: The bit-window algorithm evenly distributes the objects to nodes with a node having at most $\frac{X\left(\frac{R}{P}\right)^{v}}{R}$ additional number of objects than any other node and this number becomes negligibly small when the value of v tends to be large.

Object Lookup

The object lookup operation of the bit-window algorithm is similar to the object placement.

$\frac{X}{2N+1}$

Node Addition

Case 1:

- Node addition does not change the bit-window size, i.e., $N < 2^k$

Claim :

When $N < 2^k$, a node addition will result in approximately $\frac{X}{N+1}$ objects migrated and this number is the minimum required to ensure balanced load distribution.

Case 2:

- An addition of a node needs the bit window to expand .i.e. $N = 2^k$

Each node (among N nodes) has approximately 50% of the objects whose bit k in BW_0 is 1. Therefore, in the worst case each node will migrate $\frac{X}{2N}$ objects.

Node Deletion

<u>Case 1:</u>

- The node with the highest index (N-1) is deleted

Claim :

When a node with highest index is deleted, approximately $\frac{X}{N}$ objects are migrated, which is the minimum required.

<u>Case 2:</u>

- The node other than the highest index node is deleted

Claim :

When a node other than the highest indexed node is deleted, approximately $\frac{2X}{N}$ objects are migrated.

Performance Study

Case 1: Small Cluster with 9-16 nodes

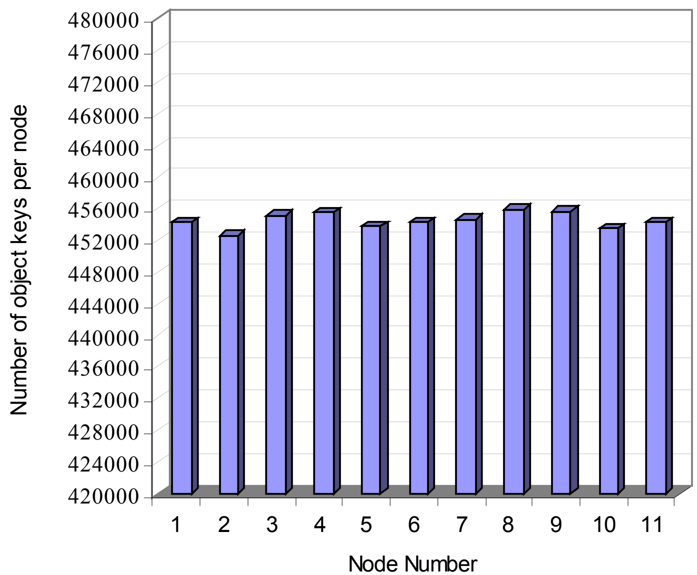

Figure . Bit –Window Algorithm: Object placement in a 11-node cluster

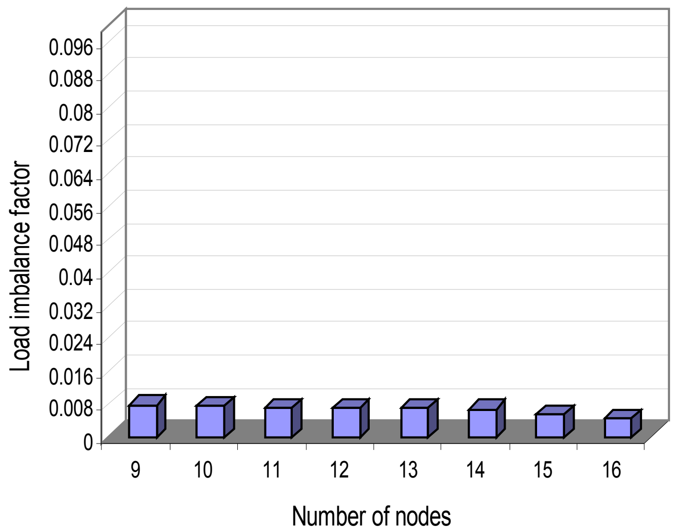

Figure : Bit –Window Algorithm: Load imbalance Index for varying number of nodes (small clusters)

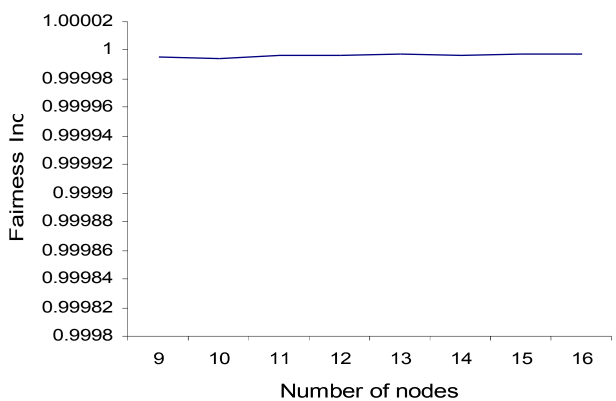

Figure : Bit –Window Algorithm: Fairness Index for varying number of nodes (small clusters).

Performance Study

Case 2: Large Cluster with 2500 nodes

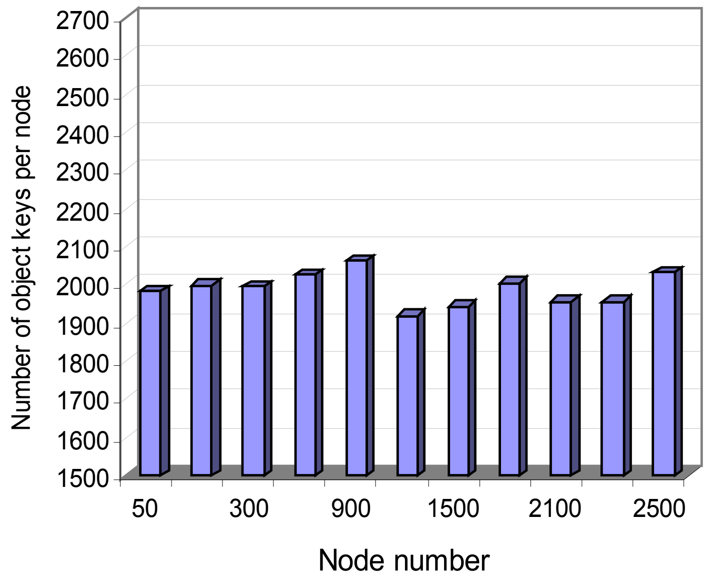

Figure : Bit –Window Algorithm: object placement for a 2500-node cluster

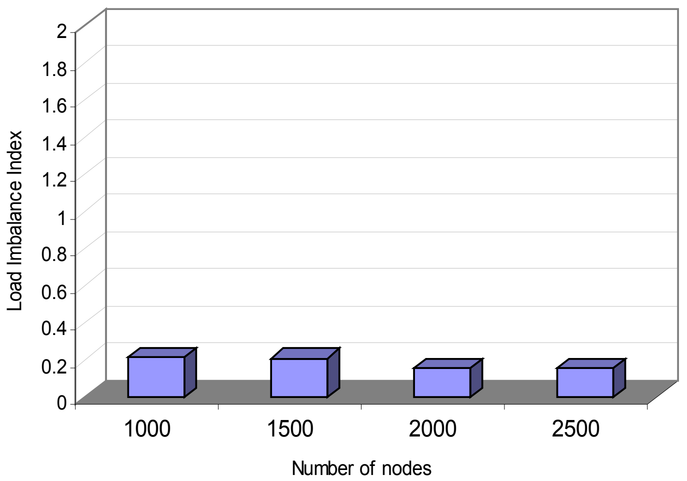

Figure : Bit –Window Algorithm: Load imbalance Index for varying number of nodes (large clusters).

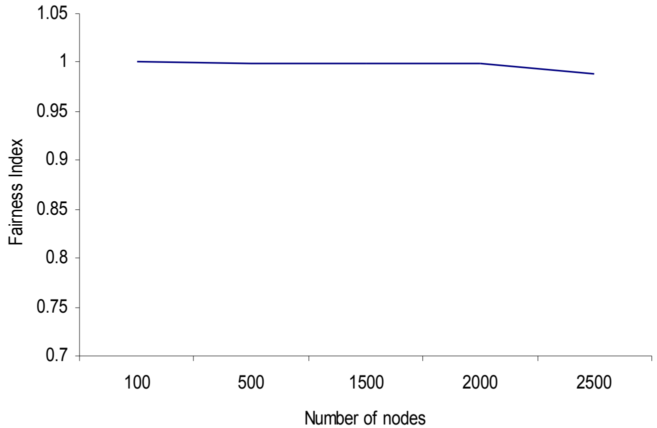

Figure : Bit –Window Algorithm: Fairness Index for varying number of nodes (large clusters).

Conclusions

- Developed a novel and efficient method based on Bit-Windows for object placement and lookup services in object-based storage clusters
- Ensures even distribution of objects
- Supports node additions and deletions with low number of object migrations
- It does not need to hop through multiple nodes for object placement and lookup, thus reducing the message overhead
- Studied the performance of the method through theoretical analysis and simulation results
- Our method is very effective in terms of the performance metrics such as load imbalance factor and fairness index