**Weighted load balancing method on data access nodes**

# **Technical field of the disclosure**

This disclosure defines a new weighted load balancing method on data access nodes, ensuring the ability to scale system’s volume horizontally and to balance the load when adding, removing data access nodes or changing nodes’ hardware configuration.

# **Technical status of the invention**

To ensure heavy load handling capability of large data access system, software developers have to solve data partitioning and load balancing problems. Furthermore, it is important to rebalance the data when adding, removing data access nodes or changing node’s hardware configuration, without interrupting the system.

Currently, there are many database management systems implementing different approaches to data partitioning, for example: MySQL, MongoDB, Aerospike, etc. However, these systems do not differentiate the weights of data access nodes. These data access nodes must have similar hardware configuration in order to handle the same amount of traffic. Additionally, modification in one of the data access node’s hardware configuration can unbalance the system.

The weighted load balancing method referred in this disclosure will ensure the ability to horizontally scale data access system’s volume, linearly increasing system’s throughput when adding new data access nodes. This method also supports customization on the weight of each data access node allowing them to have different hardware configuration.

# **Technical fundamentals of the invention**

The goal of this weighted load balancing method is to address the core problem of large data access system, which is load balancing and rebalancing the data when adding, removing data access nodes or changing nodes’ hardware configuration, minimizing the amount of data to be shifted between nodes.

In order to achieve this goal, the weighted load balancing method on data access nodes includes the following steps:

Step 1:update the routing table; when adding, removing nodes or changing nodes’ weight, move virtual nodes from a node that has a decreased number of virtual nodes to a node that has an increased number of virtual nodes; for example, consider a system which has *n* virtual nodes and *m* nodes whose identifiers and weights in turn are ID1, ID2, ID3, …, IDm and W1,W2, W3, …, Wm.

When adding a node with identifier IDm+1, the proportion of data needs to be shifted is .

When removing a node with identifier IDi having I ∈ [1, *m*], the proportion of data needs to be shifted is .

When replacing nodes’ weight with new values Q1,Q2,Q3, …, Qm, the proportion of data needs to be shifted is .

In which *m* is the total number of nodes before adding or removing; Wm+1 is the weight of added node; Wi is the weight of removed node; Q is new total weight; W is old total weight; Qk is new weight of node IDk; Wk is old weight of node IDk.

Step 2: store old routing table (before being updated) on array A1 and new routing table (after being updated on step 1) on array A2.

Step 3: block access to records that need to be moved to other node:

For read access with key K, perform hash function F(x) to calculate value I = F(K); read record corresponding to key K from node having identifier A1[I] then return result.

For write access with key K, perform hash function F(x) to calculate value I = F(K); if A1[I] = A2[I], record corresponding to key K is not being moved, write record to node having identifier A1[I] then return success code; if A1[I] ≠ A2[I], record corresponding to key K is being moved, deny write access and return error code.

Step 4: copy records from old node to new node: for each key K, perform hash function F(x) to calculate value I = F(K); if A1[I] ≠ A2[I], copy record corresponding to key K from node having identifier A1[I] to node having identifier A2[I]; if A1[I] = A2[I], record corresponding to key K does not need to be moved.

Step 5: after copying all records that need to be moved, all read/write access is performed using new routing table A2: for each read/write access with key K, perform hash function F(x) to calculate value I = F(K);record corresponding to key K is accessed on node having identifier A2[I].

Step 6: clean duplicated records: for each key K, perform hash function F(x) to calculate value I = F(K); if A1[I] ≠ A2[I], remove record corresponding to key K from node having identifier A1[I]; if A1[I] = A2[I], record corresponding to key K is not duplicated.

# **Brief description of figures**

[Figure 1](#pic_1): illustrates the data partitioning method.

Figure 2: illustrates the process of updating the routing table when adding a new node.

Figure 3: illustrates the process of updating the routing table when removing a node.

Figure 4 illustrates the process of updating the routing table when changing nodes’ weight.

Figure 5: illustrates the process of write access after the routing table is updated.

Figure 6: illustrates the process of copying records to other node after the routing table is updated.

Figure 7: illustrates the process of cleaning duplicated records after successfully copying records to other node.

Figure 8: illustrates an overview of the steps to implement the proposed method.

# **Detailed description of the invention**

The weighted load balancing method on data access nodes is composed of two methods: data partitioning method and load balancing method when adding, removing data access nodes or changing nodes’ weight.

Some terms used in the following detailed description are defined as follows:

|  |  |
| --- | --- |
| Term | Definition |
| Node | A data access node in the data access system. |
| Virtual node | Each node is corresponding to multiple virtual nodes. Data is accessed to node via corresponding virtual nodes. |
| Weight | A non-negative real number corresponding to each node. Node that has bigger weight will have more virtual nodes. |
| Routing table | A fixed size array containing the list of virtual nodes. |
| Record | A data structure which is stored in data access node. |
| Key | Each record is corresponding to one key. Record is accessed via corresponding key. |
| Hash function | A function with input is key’s value and output is a non-negative integer belonging to interval [0, *n*] with *n* is a defined positive integer. |

The data partitioning method is used to determine the data access node of the record corresponding to a specific key. In the case of data being rebalanced when adding, removing nodes or changing nodes’ weight, data is accessed using the load balancing method as will be mentioned later.

The data partitioning method uses a routing table which is an array A of *n* items, the value of each item is the identifier of a node. Item with value X is considered a virtual node of node with identifier X. The number of virtual nodes of a node is determined as:

* Where C is number of virtual nodes of the node; *n* is total number of virtual nodes; W1 is the weight of the node; W2 is the total weight of all nodes.

When accessing a record corresponding to a specific key K, perform hash function F(x) to calculate value I = F(K) ∈ [0, *n* - 1].

The value of item A[I] is the identifier of the node which has the record corresponding to key K.

Refer to figure 8, the weighted load balancing method on data access nodes according to the proposed invention is performed sequentially as follows:

Step 1: update the routing table (move virtual nodes from a node to other nodes) when adding, removing nodes or changing nodes’ weight.

Consider the system which consists of *n* virtual nodes and *m* nodes having identifiers and weights in turn are ID1, ID2, ID3, …, IDm and W1,W2, W3, …, Wm.

Refer to figure 2, when adding a new node with identifier IDm+1, the proportion of data that needs to be shifted is:

* Where *m* is total number of nodes before adding new node; Wm+1 is the weight of added node; Wk is the weight of node IDk.

For each node that has identifier IDi with I ∈ [1, *m*], move virtual nodes from node IDi to node IDm+1. The number of virtual nodes to be moved is:

* Where *n* is total number of virtual nodes; *m* is total number of nodes before adding new node; Wi is the weight of added node; Wk is the weight of node IDk.

Refer to figure 3, when removing a node with identifier IDi with I ∈ [1, *m*], the proportion of data that needs to be shifted is:

* Where Wi is the weight of removed node; Wk is the weight of node IDk.

For each node having identifier IDj with J ∈ [1, *m*] and J ≠ I, move virtual nodes from node IDi to node IDj. The number of virtual nodes to be moved is:

* Where *n* is total number of virtual nodes; *m* is total number of nodes before removing node; Wi the weight of removed node; Wj is the weight of node IDj; Wk is the weight of node IDk.

Refer to figure 4, when changing the weight list of data access nodes to Q1,Q2,Q3, …, Qm, the proportion of data that needs to be shifted is:

* Where Q is new total weight; W is old total weight; Qk is new weight of node IDk; Wk is old weight of node IDk.

For each node having identifier IDi with I ∈ [1, *m*], the number of virtual nodes to be moved is:

* Where Q is new total weight; W is old total weight; Qk is new weight of node IDk; Wk old weight of node IDk; C is the number of virtual nodes to be moved.
  + C = 0: The number of virtual nodes does not change.
  + C > 0: The number of virtual nodes increases.
  + C < 0: The number of virtual nodes decreases.

Move virtual nodes from nodes having number of virtual nodes decreased to nodes having number of virtual nodes increased.

Step 2: Store old routing table (before being updated) on array A1 and new routing table (after being updated on step 1) on array A2.

Step 3:block access to records that needs to be moved to other node:

Refer to figure 5, the process of read/write access after updating the routing table is performed as follows:

For read access with key K:

* Perform hash function F(x) to calculate value I = F(K).
* Read record corresponding to key K on node A1[I] then return the result.

For write access with key K:

* Perform hash function F(x) to calculate value I = F(K).
* If A1[I] = A2[I], record corresponding to key K is not being moved, write record to node having identifier A1[I] then return success code.
* If A1[I] ≠ A2[I], record corresponding to key K is being moved, deny write access and return error code.

Step 4: copy records from old node to new node:

Refer to figure 6, the process of copying records from old node to new node is performed as follows:

For each key K:

* Perform hash function F(x) to calculate value I = F(K).
* If A1[I] ≠ A2[I], copy the record corresponding to key K from node A1[I] to node A2[I].
* If A1[I] = A2[I], the record corresponding to key K does not need to be moved.

Step 5: after copying all records that need to be moved, all read/write access is performed using new routing table A2.

Refer to figure 1, each read/write access with key K is performed as follows:

* Perform hash function F(x) to calculate value I = F(K).
* Record corresponding to key K is accessed on node A2[I].

Step 6: clean duplicated records:

Refer to figure 7, the process of cleaning duplicated records is performed as follows:

For each key K:

* Perform hash function F(x) to calculate value I = F(K).
* If A1[I] ≠ A2[I], remove record corresponding to key K from node A1[I].
* If A1[I] = A2[I], the record corresponding to key K is not duplicated.

# **Benefits of the invention.**

Benefits of the invention include:

The data partitioning method: ensure the ability to horizontally scale the data access system, the load handling capacity is increased linearly according to the number of data access nodes.

The load balancing method when adding, removing data access nodes or changing nodes’ weight: minimize the amount of data to be shifted, minimize service interruption and ensure data integrity during the migration.

Differentiate the weights of data access nodes, allowing data access nodes to have different hardware configuration.

# **Claims**

What is claimed is:

1. The weighted load balancing method on data access nodes comprising the following steps:

Step 1:create a new routing table by updating an old routing table; when adding, removing nodes or changing a weight of nodes, move virtual nodes from a first node that has a decreased number of virtual nodes to a second node that has an increased number of virtual nodes; wherein a system which has *n* virtual nodes and *m* nodes whose identifiers and weights in turn are ID1, ID2, ID3, …, IDm and W1,W2, W3, …, Wm.

When adding a node with identifier IDm+1, a proportion of data that needs to be shifted is .

When removing a node with an identifier IDi having I ∈ [1, *m*], the proportion of data needs to be shifted is .

When replacing the weight of nodes with new values Q1,Q2,Q3, …, Qm, the proportion of data needs to be shifted is .

In which *m* is the total number of nodes before adding or removing; Wm+1 is the weight of the added node; Wi is the weight of the removed node; Q is the new total weight; W is the old total weight; Qk is the new weight of node IDk; Wk is the old weight of node IDk.

Step 2: store the old routing table (before being updated) on array an A1 and the new routing table (after being updated on step 1) on an array A2.

Step 3: block access to records that need to be moved to other node:

Wherein the block access to records comprises, For read access with key K, perform a hash function F(x) to calculate a value I = F(K); read a record corresponding to key K from the node having identifier A1[I] then return result.

For write access with key K, perform the hash function F(x) to calculate the value I = F(K); if A1[I] = A2[I], the record corresponding to key K is not being moved, write record to the node having identifier A1[I] then return success code; if A1[I] ≠ A2[I], the record corresponding to key K is being moved, deny write access and return an error code.

Step 4: copy records from the old node to the new node: for each key K, perform the hash function F(x) to calculate value I = F(K); if A1[I] ≠ A2[I], copy the record corresponding to key K from the node having identifier A1[I] to the node having identifier A2[I]; if A1[I] = A2[I], the record corresponding to key K does not need to be moved.

Step 5: after copying all records that need to be moved, all read/write access is performed using the new routing table A2: for each read/write access with key K, perform the hash function F(x) to calculate value I = F(K); therecord corresponding to key K is accessed on node having identifier A2[I].

Step 6: clean duplicated records: for each key K, perform the hash function F(x) to calculate value I = F(K); if A1[I] ≠ A2[I], remove the record corresponding to key K from node having identifier A1[I]; if A1[I] = A2[I], the record corresponding to key K is not duplicated.

1. The weighted load balancing method on data access nodes according to claim 1, wherein the number of virtual nodes needing to be moved in the case of adding a new node having identifier IDm+1 is where *n* is total number of virtual nodes; *m* is total number of nodes before adding new node; Wi is the weight of added node; Wk is the weight of node IDk.
2. The weighted load balancing method on data access nodes according to claim 1, wherein the number of virtual nodes needing to be moved in case of removing a node having identifier IDi with I ∈ [1, *m*] is where *n* is total number of virtual nodes; *m* is total number of nodes before removing a node; Wi is the weight of removed node; Wj is the weight of node IDj; Wk is the weight of node IDk.
3. The weighted load balancing method on data access nodes according to claim 1, wherein when replacing nodes’ weight with new values Q1, Q2, Q3, …, Qm, for each node having identifier IDi with I ∈ [1, *m*], the number of virtual nodes to be moved is where Q is new total weight; W is old total weight; Qk is new weight of node IDk; Wk is old weight of node IDk; C is number of virtual node to be moved.

# **Summary**

The weighted load balancing method on data access nodes ensures the ability to horizontally scale the data access system, the load handling capacity of the system is increased linearly according to the number of data access nodes. The proposed method includes the following steps: step 1:update the routing table, when adding, removing nodes or changing nodes’ weight, move virtual nodes from node having decreased number of virtual nodes to node having increased number of virtual nodes; step 2: store old routing table on array A1 and new routing table on array A2; step 3:block access to records that need to be moved; step 4: copy records from old node to node;step 5: perform read/write access using data partitioning method with new routing table A2;step 6: clean duplicated records.

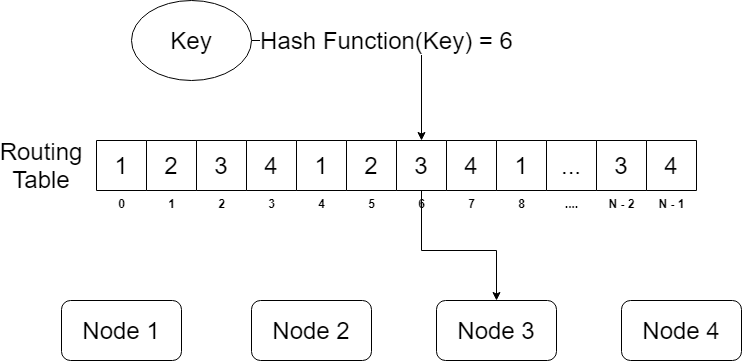


Figure 1

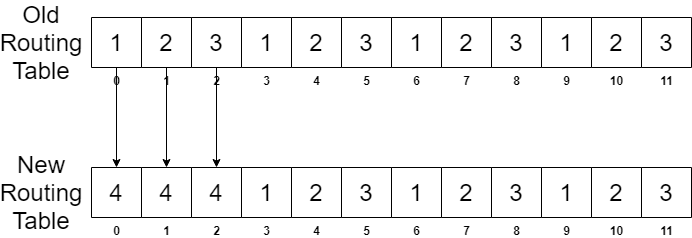


Figure 2

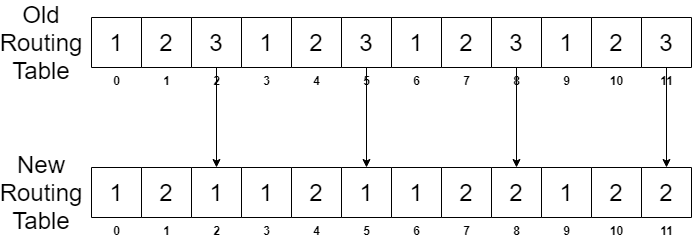


Figure 3

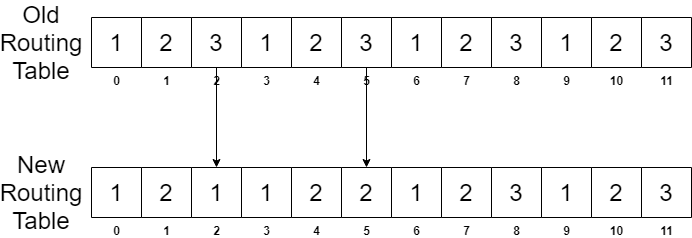


Figure 4

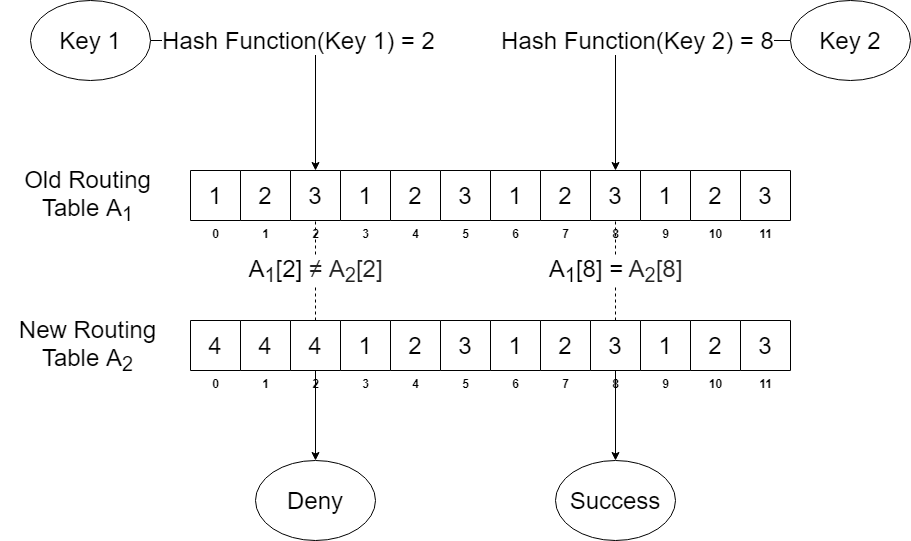


Figure 5

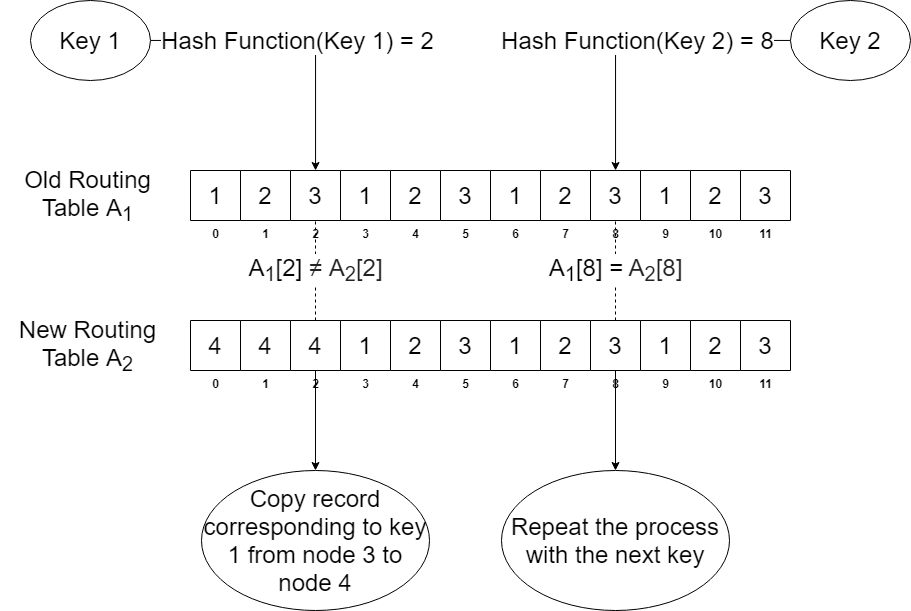


Figure 6

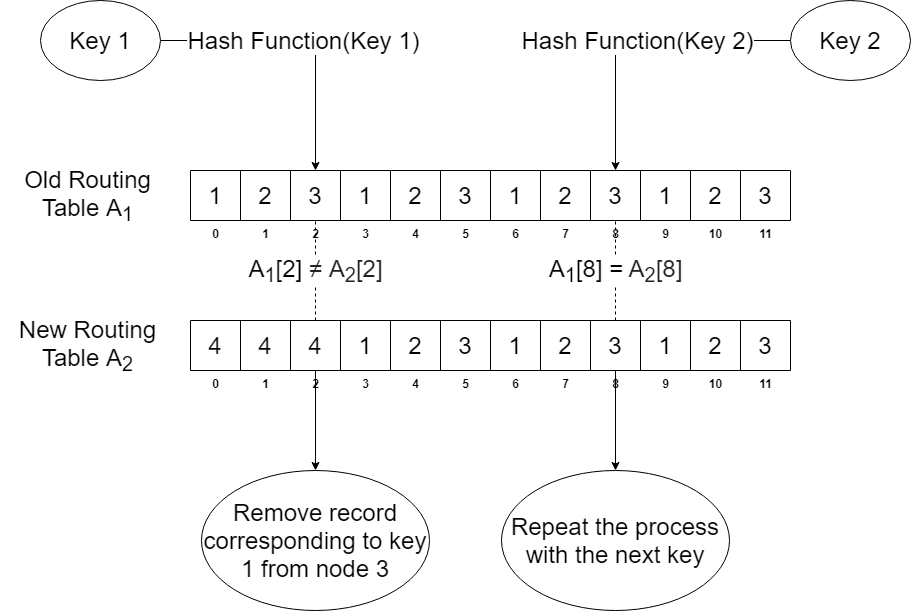


Figure 7

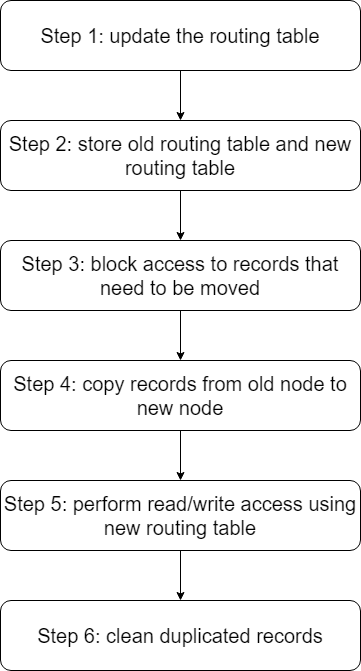


Figure 8