Deterministic Cluster-based Skip List Protocol for Dynamic Distributed Systems

Ahmed A. A. Gad-EIRab  
Department of Mathematics  
Faculty of Science  
Al-Azhar University  
Cairo, Egypt

T. A. A. Alzohairy  
Department of Mathematics  
Faculty of Science  
Al-Azhar University  
Cairo, Egypt

Khaled A. A. Khalf-Allah  
Department of Mathematics  
Faculty of Science  
Al-Azhar University  
Cairo, Egypt

ABSTRACT

Dynamic distributed system (DDS) is a continually running system with a large number of entities as processes or nodes that are connected with each other and each of them has only a partial view of the system as Peer-to-peer (P2P) system which can be decentralized, centralized or hybrid of both to make the system operations faster as possible. In P2P, any number of entities can join and leave the system at any moment which makes the topology of the system continuously changed. So, the system must deal with these changes to be as stable as possible. So, data management algorithms are needed to build an overlay network which is a logical layer that used to store the information about these entities. Skip list structure is the most common and efficient overlay structure for data management in P2P systems. However, using this structure cannot minimize the time delay for query processes as searching, inserting, and deleting in case of there is a huge number of entities in the skip list. In addition, most of existing algorithms that use this structure have been developed based on a special structure of skip list and they did not be applicable for another structure of the skip list. In this paper, to overcome these drawbacks, a new skip list structure and query processing methods are proposed. The conducted simulation results show that the proposed structure and algorithm are much better than the existing algorithms in the time delay and the required number of steps to nish any query process.

Keywords
P2P, Distributed Systems, Dynamic Systems, Deterministic, Skip List

1. INTRODUCTION

The random nature of the dynamic distributed system is a continually running system with a huge number of entities connected with each other and each entity has a partial view of the system, and any moment any number of entities can be joining and leaving the system at any moment. So, the topology of the system continuously changed, and the system must deal with these changes to be stable as possible. One of the most popular dynamic systems is peer-to-peer (P2P) system which can be unstructured/decentralized, structured/centralized networks. Of course, each model has an advantage and disadvantage. To make the system operations faster as possible hybrid model can be used. So, data management algorithms are needed to build an overlay network which is a logical or virtual layer that is built on the top of an existing physical network which is used to store the information about these entities.

Many data management algorithms are proposed and implemented to be used in such systems, and to reduce the time complexity of searching, updating and deleting processes. One of the most used data management algorithms is distributed hash table (DHT) for peer-to-peer systems. DHTs are used to lookup data in an unstructured environment based on (key, value) pair, similar to hash tables. The big advantage of DHT is its scalability and fault tolerant architecture that makes it useful for several distributed search applications. The disadvantage of DHT is its hash based look up which destroys the semantic locality of the keys, and because of the non-linear nature of the hash function that used to hash the keys and maps each one to its hash value. So, DHT can only provide service for a single query and cannot be used for systems which require range of query.

Distributed Segment Tree (DST) introduced to support of range query and cover query over DHT. DST is built on top of generic DHT based on the concept of a segment tree in maintaining the structure of ranges; DST is shown to be very efficient for supporting both range query and cover query in a uniform way. But it performs poorly for a large number of nodes because of the limitation on the number of keys at each node. Skip List (SL) is a probabilistic alternative to balanced trees proposed by William Pugh. SL can be used instead of balanced trees in [15]. In [15], the implementation of insertion, deletion and search algorithms are much simpler and faster than balanced trees based algorithms. SL is much faster compared to linked list. To search any element in a sorted linked list, we might have to traverse all the nodes of the list. Here the search complexity is $O(n)$. In a skip list can operate in the order of $\log n$ in average cases where $n$ is the total number of elements. As shown in Fig. 1 and Fig. 2, the SL is structure is constructed as levels. Each level is a sorted list. The lowest level contains $n$ nodes, the next higher level is another linked list with subset number of nodes from previous level, and so on till the top higher level reached with minimum number of nodes see Fig. 2. However, using this structure cannot minimize the time delay for query processes as searching, inserting, and deleting in case of there is a huge number of entities in the skip list. In addition, most
of existing algorithms that use the skip list structure have been
developed based on a special structure of skip list and they did not
be applicable for another structure of the skip list. In this paper, to
overcome these drawbacks, a new cluster-based skip list structure
and query processing methods are proposed. The rest of this pa-
rer is organized as follows: Section 2 gives an overview of the
related works on distributed skip-list. Section 3 introduces EN-
TITY MANAGEMENT PROBLEM IN P2P and its formulation. Sections
4 introduces the proposed cluster-based skip list algorithm. Section
5 describes of the experimental and simulation results. Section 6
concludes this paper.

2. RELATED WORK
Skip lists are balanced by consulting a random number generator.
Searching a linked list for a specific element may require that ev-
every element of the list be examined as shown in Fig. 3. If the list is
stored in sorted order and every other element of the list also has a
pointer to the element two ahead of it in the list, searching for an
element requires that no more than \[ \frac{N}{2} + 1 \] elements be examined
(where \( N \) is the length of the list). By giving every \((2^i)\)th element a
pointer to the elements ahead as shown in Fig. 4, the number of
elements that must be examined can be reduced to \( \lceil \log_2 N \rceil \), while
only doubling the number of pointers. Such that the levels of ele-
ments as shown in Fig. 4, are distributed in a simple pattern: 50% are
level 1, 25% are level 2, 12.5% are level 3, and so on [15].

3. ENTITY MANAGEMENT PROBLEM IN P2P
Entity management problem (EMP) is one of the key issues in the
P2P systems. This problem deals with how to minimize the time
cost for entity management in dynamic environments. In this sec-
ction, the assumptions and models will be described then the formu-
lation of EMP problem will be introduced.

3.1 Assumptions, and Models
We assume that there is a dynamic environment which contains
a set of devices as nodes \( N \), which share information and re-

![Fig. 1. Sorted Linked List](image1)

![Fig. 2. Skip List](image2)

![Fig. 3. Search inside Sorted Linked List to find entity number 14](image3)

![Fig. 4. Search inside Sorted Skip List to find entity number 14](image4)
courses. Each node can join or leave the system at any moment, and the topology of the system changes over the time and these changes may make the system fails. We assume that the P2P system model constructed from set of nodes $N = \{\text{node}_1, \text{node}_2, \text{node}_3, \ldots, \text{node}_i \mid i > 0\}$. Each node $\text{node}_i$ contains $(key_i, value_i)$, these nodes are connected together and communicating with each other by sending messages to share information and resources like CPU and Memory.

### 3.2 Problem Formulation

The total average delay insert operation time is denoted as $T_{I_{total}}$. The total average delay delete operation time is denoted as $T_{D_{total}}$. The total average delay search operation time is denoted as $T_{S_{total}}$. The big challenge is to deal with the changes in the topology of the system by applying these changes on the entity management of data structure such insertion and deletion, to make an efficient search operation.

So, based on the proposed system model, the total average delay time $T_{I_{total}}$ and $T_{D_{total}}$ are calculated as following:

$$T_{I_{total}} = T_I + T_T$$  \hspace{1cm} (1)

$$T_{D_{total}} = T_S + T_D$$  \hspace{1cm} (2)

where $T_I$ is the delay time of insert operation, $T_T$ is the delay time of delete operation and $T_S$ is the delay time of search operation.

**Objective function:** the objective function is to minimize the total average delay time cost for $T_{I_{total}}$, $T_{D_{total}}$ and $T_{S_{total}}$ operations by reducing $T_I$, $T_D$ and $T_S$. This objective function is defined as follows.

Minimize $T_{I_{total}} + T_{D_{total}} + T_{S_{total}}$ \hspace{1cm} (3)

such that

$$\exists \text{path} (\text{node}_i, \text{node}_{i+1}); \forall \text{node}_i, \text{node}_{i+1} \in N$$  \hspace{1cm} (4)

and $\text{node}_i \neq \text{node}_{i+1}$

$$\text{if value}(\text{node}_i) = \text{value}(\text{node}_{i+1})$$

$$\Rightarrow \text{key}(\text{node}_i) = \text{key}(\text{node}_{i+1})$$

$$\Rightarrow \text{node}_i = \text{node}_{i+1}$$  \hspace{1cm} (5)

$$\forall \text{node}_i, \text{node}_{i+1} \in N$$

$$\text{steps}(\text{search}) \neq \infty$$  \hspace{1cm} (6)

$$T_{stop}(\text{node}_i) \leq T_{stop}(\text{node}_{j}); \forall \text{node}_i, \text{node}_{j} \in N, j \leq i$$ \hspace{1cm} (7)

where constraint (4) means that there is no circle or path between any node and itself to avoid infinite loops i.e. if there is a path from $\text{node}_i$ to $\text{node}_{i+1}$ then $\text{node}_i$ and $\text{node}_{i+1}$ are not equal. Constraint (5) means that the value of any nodes is not the same i.e. if the values of two nodes $\text{node}_i$ and $\text{node}_j$ are equal then the nodes are the same node to avoid the duplication of nodes. Constraint (6) means that the required time to stop search operation of $\text{node}_i$ less than or equal the required time to stop search operation of $\text{node}_j$.

### 4. CLUSTER-BASED SKIP LIST ALGORITHM

To solve the EMP problem, a new skip list algorithm called Cluster-Based Skip List Algorithm (CBSL) is proposed. This proposed algorithm is an improvement for the Standard Skip List (SL) algorithm by convert the standard Skip List to Clustered-Based Skip List.

#### 4.1 Basic Idea

The proposed CBSL is based on (1) dividing each level in the skip list into a group of clusters which is considered as new structure. (2) proposing two different methods for join, delete, and search operations. The first method called Sequential method (SM) which searches for any entity in a sequential order cluster by cluster. While the second method called Prediction method (PM) which searches for any entity by predicting its nearest cluster rst, then jumps to this predicted cluster and searches for the required entity. In the rest of this section, formation of clusters is described, the proposed methods are introduced and finally the insert, delete, and search operations are described.

#### 4.2 Clustering Formation Process

All nodes in SL are distributed on levels $\{\text{level}_1, \text{level}_2, \text{level}_3, \ldots, \text{level}_j \mid j > 0\}$. Each node $\text{node}_i$ contains $(key_i, value_i)$, and has four links. Two for the connections $right_i$ and $left_i$ with its neighbors, and the other two links $up_i$ and $down_i$ to connect the node with itself at different levels. In CBSL, these nodes will be distributed on clusters $\{C_1, C_2, C_3, \ldots, C_k \mid k > 0\}$. Each cluster has a number of entities $C_{max}$ with the same levels in the SL as shown in Fig. 5. These clusters are another logical layer on SL to search about any entity by using two search methods to nd which cluster contains on the target entity, which makes the search inside the cluster faster than the search in the whole skip list as shown in Fig. 5. Each cluster has maximum number of nodes $C_{max}$, $C_{max}$ is xed for all clusters and is unchanged through running time. Each cluster has three main elements start_i, end_i, and top_i, elements, and has two connections to previous and next clusters next_i, prev_i, start_i, end_i, and top_i elements specied while building the skip list and changed after adding or deleting elements as shwon in Fig. 5.

#### 4.3 The proposed methods

Based on clustering design, CBSL uses two different searching methods: Sequential method (SM) and Prediction method (PM) These two methods are described as follows.

1. **Sequential method (SM):** which checks each cluster of the list in a sequential order cluster by cluster until it nds a cluster that has the target entity. If the algorithm reaches the end of the list, the search terminates with failure as shown in Fig 9 and and Algo 4.

2. **Prediction method (PM):** in this method, the proposed algorithm predicates the position of the cluster or the closest cluster position for the required entity as shown in Fig 10 and Algo 5 by using the following equation:

$$C_{ex} = \lceil \frac{X}{C_{max}} \rceil$$ \hspace{1cm} (8)

Such that $C_{ex}$ the expected position of the cluster, $X$ is Node.ID and $C_{max}$ maximum number of elements can be added for each cluster.
By using one of these two methods, CBSL checks if the target cluster was reached or not. If not, go to the next cluster or the previous one depending on the value of nodeID and the starti and endi of the current cluster. If the cluster does not exist then a new one will be created on the right position and the connections added.

4.4 Search, Insert, and Delete Operations

4.4.1 Search operation. As shown in Fig 6 and Algo 1, CBSL starts to find the position of cluster first by using SM or PM methods. Then starts to search inside the cluster to find the node position. The search inside the cluster from the top element of this cluster. Then the search move to left or right or down based on the key of the node.

Algorithm 1: CBSL Search Algorithm

```
Input : NodeID
Output: Node position and data
1. cluster ← findClusterByPM(NodeID) or findClusterBySM(NodeID)
2. Node ← findNode(NodeID, cluster)
```

Algorithm 2: CBSL insert cluster algorithm insertCluster function

```
function insertCluster (cluster, newCluster)
Input : cluster, newCluster
Output: New cluster position
1. newClusternext ← clusternext
2. newClusterprev ← clusternextprev
3. clusternextprev ← newCluster
4. clusternext ← newCluster
5. return newCluster
```
4.4.2 Insert Operation. To insert or join a new node in the list, CBSL follows the following steps as shown in Algorithm 9:

1. Determine the level of the new node by using SM or PM methods as shown in Algorithm 1, 3, 4.
2. Insert a new node (NodeId) in its level.
3. Add new edge between the left and a new node (NodeId).
4. Add new edge between the right and a new node (NodeId).
5. Delete the old edge between the left and the right nodes. (When describing a link we always state the smaller identifier first).

4.4.3 Delete Operation. To delete an existing node from the list, CBSL follows the following steps as shown in Algorithm 10:

1. Search to find cluster which contains the node that will be deleted by using SM or PM methods.
2. If the cluster exists then search inside it to find the node position, else do nothing.
3. If the node exists go to next step, else do nothing.
4. Add a new edge between the left and right nodes at all levels.
5. Delete the old edge between the left and the right nodes. (When describing a link we always state the smaller identifier first).

(6) Update the other clusters if it is required.

**Algorithm 3: Find Cluster using SM** findClusterBySM function

```plaintext
1 function findClusterBySM (NodeId) ;
2 Input : NodeID
3 Output: Cluster position
4 5
6 cluster ←− cluster
7 start ←− cluster
8 end ←− cluster
9 if NodeID ≥ start and NodeID ≤ end then
10 return cluster
11 else if clusternext ≠ NULL and NodeID ≥ clusternext
12 return clusternext
13 else
14 return failure ⊿ or call insertCluster function for adding a new Node
```

**Algorithm 4: Find Cluster using PM** findClusterByPM function

```plaintext
1 function findClusterByPM (NodeId) ;
2 Input : NodeID
3 Output: Cluster
4 5
6 i ←− ┐\[\text{NodeId} \bmod \text{Cmax}\] ┐
7 cluster ←− cluster
8 start ←− cluster
9 end ←− cluster
10 if NodeID ≥ start and NodeID ≤ end then
11 return cluster
12 else if NodeID ≤ start and clusterprev ≠ NULL and
13 return searchPrevClusters(NodeID, clusterprev)
14 else if NodeID ≥ end and clusternext ≠ NULL and
15 return searchNextClusters(NodeID, clusternext)
16 else
17 return failure ⊿ or call insertCluster function for adding a new Node
```
Algorithm 5: Search next clusters \texttt{searchNextClusters} function

\begin{verbatim}
function searchNextClusters (NodeID, cluster) : 
  Input : NodeID, cluster 
  Output: Cluster position 
  start ← cluster\_start 
  end ← cluster\_end 
  if NodeID ≥ start and NodeID ≤ end then 
    return cluster 
  while NodeID ≥ end and cluster\_next \neq NULL and NodeID ≥ cluster\_next\_start do 
    cluster ← cluster\_next 
  start ← cluster\_start 
  end ← cluster\_end 
  if NodeID ≥ start and NodeID ≤ end then 
    return cluster 
  return failure 
\end{verbatim}

Algorithm 6: Search previous clusters \texttt{searchPrevClusters} function

\begin{verbatim}
function searchPrevClusters (NodeID, cluster) : 
  Input : NodeID, cluster 
  Output: Cluster position 
  start ← cluster\_start 
  end ← cluster\_end 
  if NodeID ≥ start and NodeID ≤ end then 
    return cluster 
  while NodeID ≤ start and cluster\_prev \neq NULL and NodeID ≤ cluster\_prev\_end do 
    cluster ← cluster\_prev 
  start ← cluster\_start 
  end ← cluster\_end 
  if NodeID ≥ start and NodeID ≤ end then 
    return cluster 
  return failure 
\end{verbatim}
**Algorithm 7:** Find Node function

```plaintext
function findNode (NodeID, cluster);
Input : NodeID, cluster
Output: Node Data
2 start ← cluster.start
3 end ← cluster.end
4 top ← cluster.top
5 if NodeID = start|end|top then
6  return start|end|top
7 else
8   return findNodeInCluster(NodeID, top, start, end)
9 return failure ▷ or call insertCluster function for adding a new Node
```

**Algorithm 8:** Find Node inside cluster function

```plaintext
function findNodeInCluster (NodeID, top, start, end);
Input: NodeID, top, start, end
Output: Node and Node Data
2 node = top
3 while TRUE do
4   if NodeID < node.key then
5     while node.down ≠ NULL and node.left.key ≤ node.key
6       node = node.down;
7     while node.left ≠ NULL and node.left.key ≥ start.key
8       and node.left.key ≤ end.key do
9       if node.key ≤ NodeID then
10      break
11     node = node.left;
12   else
13     node = node.down;
14     break
15 else
16     while node.right ≠ NULL and node.right.key ≤ end.key
17       and node.right.key < NodeID do
18       node = node.right;
19     else
20      node = node.down;
21     break
22 return node;
```

**Algorithm 9:** CBSL insert algorithm function

```plaintext
function InsertNewNode (node);
Input : node
Output: Insert success or failure
2 NodeID ← node.id
3 cluster ← insertNode(NodeID) ▷ or findClusterBySM(NodeID)
4 if cluster not found then
5   cluster ← insertCluster(node, cluster)
6   The node will be added at start of the cluster
7 else
8   node ← findNode(NodeID, cluster) ▷ Find the Node or smallest node before it if not found
9   if node.down ≠ NULL then
10  The node will be added to the first level of the cluster
11  The links left and right will be created between the node and its neighbors
12  up and down links will created between the node and itself at each level
13 return node
```
Algorithm 10: CBSL Delete Algorithm \( \text{deleteNode}(\text{NodeID}) \)

1. \textbf{function deleteNode} (\text{NodeID})
2. \textbf{Input}: NodeID
3. \textbf{Output}: Delete success or failure
4. \text{cluster} \leftarrow \text{findClusterByPM}(\text{NodeID}) \quad \triangleright \text{ or findClusterBySM(\text{NodeID})}
5. \textbf{if} \text{cluster not found} \quad \textbf{then}
6. \quad \text{return} \quad \text{failure}
7. \textbf{else}
8. \quad \text{node} \leftarrow \text{findNode}(\text{NodeID}, \text{cluster}) \quad \triangleright \text{Find the Node or the smallest node before it if not found}
9. \quad \text{The links between the left and right neighbors will be created at each level first before the node's links are removed}
10. \quad \text{up and down links will created between the node and itself at each level}
11. \quad \text{The node will be deleted to the first level of the cluster}
12. \quad \text{return} \quad \text{success}

4.5 Complexity

The number of steps to find cluster by PM and SM can be represented as following:

\( - \text{SMcost} : O(M_{\text{clusters}}) \).

\( - \text{PMcost} : 1. \)

Such that \( SMcost \) is the complexity of SM to find the cluster, and \( PMcost \) is the complexity of SM to find the cluster.

The number of elements that must be examined can be reduced to:

\( - \text{Search Operation Complexity by using SM is} \quad O(\log N_{\text{MAX}}) + O(M_{\text{clusters}}) \).

\( - \text{Search Operation Complexity by using PM is} \quad O(\log N_{\text{MAX}}) + 1. \)

Clusters layer can be extended as skip list of clusters, so the complexity of finding the cluster can be computed as following:

\( - SLMcost : O(\log M_{\text{clusters}}) \).

Then the number of elements that must be examined can be reduced to: Search Operation Complexity by using SLM is \( O(\log N_{\text{MAX}}) + O(\log M_{\text{clusters}}) \).

5. SIMULATION RESULTS AND ANALYSIS

In this section, to evaluate the proposed CBSL algorithm, CBSL is compared with the standard skip list protocol for query delay time. The simulation parameters are shown in Table 1.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total Number of Nodes</td>
<td>5000, 10000</td>
</tr>
<tr>
<td>maximum Number of Nodes Per Cluster</td>
<td>5, 10, ..., 500</td>
</tr>
</tbody>
</table>

Fig 12 shows the average delay time against the the target NodeID for searching, inserting, and deleting when the number of nodes was 5000 and the maximum number of nodes per cluster was 5 nodes. As shown in Fig 6, the average delay time increases as the target NodeID increases, this is because, the existence of more nodes in the list needs more time to manage the list for inserting, searching, and deleting operations. Also, the average delay time of CBSL-SM and CBSL-PM is much better than the standard SL method. This is because, CBSL uses clustering design for management the skiplist while SL does not. In addition, PM achieves lower delay time than SM method, this is because, PM can reach to the required cluster in less number of steps by skipping some clusters.

Fig 13 and Fig 14 show the average delay time against the target NodeID for searching, inserting, and deleting when the number of nodes was 10000 and the maximum number of nodes per cluster was 5 and 10 nodes.

Of course the algorithms are tested with different maximum number of nodes for the clusters as shown in Fig 15 and Fig 16 as shown in Fig 15 and Fig 16 when the number of nodes for each cluster increased to be \( C_{\text{max}} = 100. \) The search inside the cluster by using the expected method becomes similar to the slandered SL search. Thus the expected method gives us a good result with cluster of small number of nodes. And the sequential method average
delay time results become faster than the expected method average delay time, because the number of clusters was reduced. Fig 17 show the relation between the maximum number of nodes of cluster and the total average delay time. As shown in Fig 17 the CBSL methods give us a good result for clusters of maximum length from 5 to 250. Thus when the maximum length of cluster increased after 250 the standard SL becomes better, because the search inside cluster of bigger length take more time than the clusters of small length. And this is make sense, because we loss the advantage of clustering.

6. CONCLUSION

In this paper, the most relevant data structure protocols types in dynamic distributed environments were introduced. In addition, a new cluster based skip list (CBSL) algorithm in dynamic distributed environment was proposed. CBSL algorithm divides the standard skip list into a group of clusters and uses two methods to

7. REFERENCES


[2] Roberto ”Baldoni, Marin Bertier, Michel Raynal, and Sara” Tucci-Piergiovanni. "looking for a definition of dynamic dis-
Fig. 5. CBSL Structure