SkipMard: A Multi-attribute Peer-to-Peer Resource Discovery Approach

Tao He¹ ², Jun Ni¹ ², Alberto M Segre¹, Shaowen Wang², Boyd M Knosp²
¹Department of Computer Science, The University of Iowa, Iowa City, IA 52242
²Information Technology Services, The University of Iowa, Iowa City, IA 52242, USA
tao-he@uiowa.edu; jun-ni@uiowa.edu; segre@vinci.cs.uiowa.edu;
shaowen-wang@uiowa.edu; boyd-knosp@uiowa.edu

Abstract

Resource discovery technologies for Grids and Peer-to-Peer (P2P) systems share some characteristics. However, some P2P systems such as Skip Graph cannot simply be applied to Grid resource discovery because complex grid resources need to be searched by using multi-attribute queries. This paper proposes a new multi-attribute P2P resource discovery approach (SkipMard) that extends Skip Graph structure to support multi-attribute queries. SkipMard provides a prefix matching resource routing algorithm to resolve multi-attribute queries, and introduces the concepts of "layer" and "crossing layer nearest neighbor" into the data structure. To decrease message passing numbers, an approximate closest-point method is addressed that can help routing a searching key to a node with a key value that has the minimum distance between two keys. Each node has \(O(m*l)\) neighbors for total \(m\) layers and \(l\) levels in SkipMard. The expected time for a multi-attribute query is \(O(\log N)\) and the message passing number is \(O(\log N) + O(k)\).

Keywords: Resource Discovery, Network, Multi-Attribute, Peer-to-Peer, Grid, SkipMard, DHT.

1. Introduction

The current resource discovery systems fall into two categories according to the resource-sharing environment; Grids and Peer-to-Peer systems. Grids [1][2] provide a reliable, integral infrastructure to support the compute-intensive applications and collect computational resources from multiple administrative domains. P2P systems [5] are usually the large scale internet applications that contain tremendous dynamic resources. Those systems are known for its unreliability, heterogeneity, and diversity. While P2P systems are originally designed to share files among grass-roots in the Internet, Grids aim to provide the sophisticated services to the scientific communities that can deliver desired qualities of service (QoS). However, the explicit administration, strict secure policies and stronger community links also increase the cost of integrating computational resources into a Grid. As the size is growing, Grids will increasingly face the same problems with P2P systems [5]: unreliable resources and intermittent participation. Researchers [6] believe there will be an eventual convergence between Grids and P2P systems.

A P2P network is a decentralized, unreliable, and fast-changing system. Thus, a structured P2P resource discovery system cannot simply be applied to Grid resource discovery because of strict requirements of grid resources. One requirement is the support of multi-attribute queries. The computational resources that we want to discover in Grids have the multiple attributes. For example, locating a high performance computer on a network has to consider at least three attributes of this computer: CPU speed, memory size and disk capacity. The inherent characteristics show that it is a complex and challenging topic to researchers to develop an efficient solution to discover the computational resources with multi-attribute in a dynamic, decentralized P2P network and apply it into Grids. In our research work, we try to investigate a design of P2P overlay networks, which can support the multi-attribute computational resource discovery in Grids.

The major contribution of our work is that we introduce a new multiple-attribute overlay network design, called “SkipMard” (Multiple attributes resource discovery using Skip List-based approaches). This new design uses Skip List-based techniques and supports a multi-attribute P2P resource discovery on a
distributed Grid network. Current Skip Graph [3] and SkipNet [4] are two successful Skip List-based approaches towards P2P system. But their theory was only designed to support the one-attribute query. In our design of SkipMard, we extend the existing Skip List-based approaches such as Skip Graph from the one-attribute query to the multi-attribute query and all computational resources are registered in a decentralized model with a set of attribute-value pairs. These resources are guaranteed to be searched in a few hops by routing algorithms in SkipMard. In addition, an approximate closest-point method is addressed that can help routing a searching key to a node with a key value that has the minimum distance between the two keys. And a parallel structured flooding technique with Time-To-Live (TTL) mechanism is provided for the discovery of dynamic attribute resources. SkipMard can efficiently execute the multi-attribute query, increase the network expansibility and decrease the number of transmitted messages.

2. Relative Works

Many recent structured P2P systems and Grid systems are related to our research work. Traditional Grid and P2P approaches can be classified by the network topology: centralized and decentralized. Centralized resource discovery approaches are usually based on central servers hosting a directory of resources. Napster [7] is a typical example of centralized P2P system to use a super-node working as a central indexed directory for all resources. Globus MDS [8] uses a suite of web services such as index service to collect and register resources on Grids. Condor system [9] is originally used to manage a cluster of dedicated computers as in a Beowulf Cluster. But Condor-G [10] is used to build Grid-style computing environments that cross administrative boundaries and fully interoperate Grid resources with Globus MDS system. Centralized models can have some obvious advantages in resource discovery systems: easy operation, reliable resources and fast query. However, the disadvantages are also obvious: scalability limited, bottleneck, overload and vulnerability. If only there is a single point of failure, centralized approaches are more vulnerable to be attacked.

Instead of centralized models, decentralized approaches can broadcast resource queries to all connected nodes or design a structured network topology to route resource queries along routing tables. The typical example of decentralized but unstructured P2P system is Gnutella [7][11], whose network topology is an unstructured mesh and a query is executed hop-by-hop through this mesh until the success or failure or timeout. In Grid environment, [13] proposed a P2P approach to organize and locate grid resources by using a Time-To-Live (TTL) mechanism. TTL-based query forwarding is similar to Gnutella. Although this mechanism is effective and relatively simple, it does not ensure the successful execution of a query and usually has the inefficient message flooding problem.

Decentralized and structured topologies usually look like a ring, a d-dimension torus, structured mesh and butterfly. These structures are usually constructed by distributed hash table- DHT techniques and Skip List-based approaches [12]. They all guarantee a query to be completed just in a few hops.

DHT-based approaches use hashing functions such as SHA1, MD5 simply to guarantee load balance, efficiency and scalability. However, random-looking hash values of keys also destroy the natural locality property in a distributed system. Traditional DHT-based approaches such as Chord [16], and Pastry [17] only support one-attribute exact match queries with expected time O(log n) for routing. Recently, researchers also propose many new DHT-based projects into Grid environment. MAAN [12] extends Chord to support the range query and multi-attribute query. It designs a locality preserving hash function to uniformly distribute resources on all nodes, and adopts one separate DHT for each attribute. Two kinds of routing algorithms respectively use iterative or single attribute dominated query to resolve multi-attribute queries among multiple DHTs. SWORD [14] uses another method to wave all attributes into one DHT. In SWORD, users can describe their desired configuration by using a native SWORD XML syntax and the system will return an ordered list with the lowest-penalty mapping of available nodes to groups, each of which is defined by a required number of nodes or a range of acceptable node characteristics.

Skip List-based approaches are naturally locality preserving. Skip List [15] is a simple structure to link each node with ordered multiple linked lists and then to build multiple levels. Skip List has the O(log n) expected time for searching, inserting and removing. It is efficient and scalable. But if researchers want to apply Skip List into the structured overlay P2P network, a problem will be found that there is no redundant property in Skip List. One failure node will destroy the whole network under some situations. To overcome this problem, some new
Skip List-based data structures have been addressed in Skip Graph and SkipNet. Unlike Skip List, this data structure has 2^i linked lists in one level where i is the level number. Besides the key value, each node has a vector that generated the 0 or 1 by a random function. The vector length is the total level number of Skip Graph. One node links to the other nodes at level i with matching the prefix of length i in the member vector. The expected time for searching and inserting in Skip Graph is also O(log n). However, its design just mainly supports single query and range query on one-attribute resource lookup. In Grid environment, the resources need to be searched by using the multi-attribute queries. Thus, in our research, we propose a new data structure SkipMard, which extends the Skip Graph to support multi-attribute queries.

3. Design

In our research, we focus on Skip List-based tools to design a structured P2P network without DHT. For this reason, our primary consideration is locality property. The advantage of DHT's approaches is that it can easily guarantee the load balance, efficiency and scalability on P2P systems. But one of the primary drawbacks is that the use of random-looking hash values of the key generated by hashing functions destroys the P2P natural locality property. The locality property ensures that the query messages are only routed within two location-near nodes. Another important drawback is that regular DHT technique only supports the “exact matching”. Therefore, multiple “exact matching” operations have to be repeatedly executed to satisfy a complex range query, and a number of nodes are visited repeatedly in the regular DHT-based P2P overlay network.

SkipMard (Multi-attribute resource discovery using Skip List-based tools) is our design of a preliminary P2P data structure, which naturally supports the multi-attribute resource discovery. In this part, we will introduce the data structure of SkipMard, the routing algorithms and the join/leave algorithms. And then we will analyze the runtime of SkipMard.

3.1 Architecture

The design of SkipMard is primarily based on the existing Skip List-based algorithm -- Skip Graph. It has been further developed to extend Skip Graph to naturally support multi-attribute queries from one-attribute queries.

In SkipMard, we employ a location address space and multiple pre-defined resource-attribute tables. A node in this proposal usually refers to a computer that holds an independent IP address or URL, and hosts multiple resources and attributes. Each node corresponds to an ID in the location address space by the key number. Each attribute belongs to one pre-defined resource-attribute table. All attributes in this node can be classified into different resource categories according to these pre-defined tables, and values of attributes are enumerated to represent a group. For example, a node has 3 resources: CPU, memory and disk. Each resource defines one attribute, speed, size and capacity respectively. The values of each attribute can be defined in Table 1.

SkipMard is a Skip Graph-like multi-level data structure defined as a generalization of a Skip List. As in Skip Graph, each node is a member of multiple doubly linked lists. The level 0 contains all nodes in order. However, compared with the data structure of Skip Graph, SkipMard has 3 obvious different properties. (1) First, instead of the membership vector m(x) in Skip Graph, SkipMard uses a resource vector rv(x). An element in the membership vector is generated a value 0 or 1 by a binary random function and this value itself is meaningless, while each element in a resource vector concretely stands for one resource and the value of an element represents an attribute value. The length of a resource vector is equal to the number of maxlevel, or the high of a SkipMard. (2) Secondly, since Skip Graph uses a binary element in its membership vector, this approach has only 2^i linked lists at i level, where i is 0... maxlevel. But in SkipMard, we have p^i linked lists at one level, where p >=2 (such as p=3, 4, etc.), one of which is called layer. If p is equal to 2, then this SkipMard is an instance of Skip Graph. (3) Finally, each node in Skip Graph only has left and right neighbor on one level. But in SkipMard, each node on one level has not only the left and right neighbor in the same layer, but also contains the pointers to store the closest neighbor with different layers. We call those neighbors Crossing Nearest Layer Neighbor. Because each node in SkipMard contains more neighbor information than in Skip Graph, the expected space of routing table of SkipMard is O(m*n^2* log n), where n is total size of nodes and m is total number of layers. Fig. 1 shows one example of SkipMard with 14 nodes, 3 resources, and a length of resource vector 3.
SkipMard data structure provides an easy way to implement multi-attribute queries. A resource vector of each node can contain multiple attributes of computational resources that we are interested in. Given an application of SkipMard on the internet, we can assume the deployment of computing resources is in a randomized distribution. Thus SkipMard has $O(\log n)$ expected routing time.

3.2 Algorithms

SkipMard provides two routing algorithms and both have $O(\log n)$ expected time. One is to implement one-attribute query by routing a key. The other is for multi-attribute queries by routing a vector. A prefix matching technique and an approximate closest-point method are used in the routing algorithms. In the algorithm of routing by a resource vector, we briefly introduce a structure parallel flooding strategy for dynamic multi-attribute query.

3.2.1 Approximate Closest-Point Method

In routing algorithms for one-attribute, SkipMard employs an approximate closest-point method for a specified searching key that enables each node to always try to route this key to the node with the closest key value. Unlike Skip Graph, which just uses the comparison operations to determine the next node where the requested message is sent, SkipMard uses a minimal distance function to determine the next node which is closer to requested node than the previous one. Skip Graph adopts one side forwarding approach, while SkipMard always goes around the requested node and tries to converge. Fig.2 shows an example of finding a position to insert node J. We can compare the processing of Skip Graph with that of SkipMard. Search starts on the node W. In Skip Graph, node W finds J is bigger than its neighbor node G on the level 2. Thus, it looks for the node R on the level 1. R is bigger than J, so routing message is sent to R. Finally Skip Graph finds the right position for J on the left side of node M at level 0. But in SkipMard, node G is the closest node to meet the query for node J among all neighbors of node W because the distance of node G to node J is minimal. So the routing message is sent to G from W, not to node R. Finally SkipMard finds the right position for J on the right side of node G at level 0. The expected time of routing by a key is still $O(\log n)$ as in Skip Graph. The proof is Lemma 1. The pseudo-code of approximate closest-point method is algorithm 1 (See Appendix).

3.2.2 Routing by a Key

Routing algorithm by a key in SkipMard belongs to one-attribute query. It applies an approximate closest-point method to calculate a minimal distance from all neighbor of the current node to this requested node. This minimal distance determines the next accessing node where a searching message is sent. Like Skip List and Skip Graph, the search starts at the top-most level of any node and then goes down from top-level to the bottom level. The expected time of routing by a key is $O(\log n)$.

Fig. 3 shows an example to search node 56 from the start node 101. First, node 101 finds its neighbor node 78, which has a minimal distance to requested node 56. Secondly, a query message is sent to node 78. Likewise, node 78 finds its crossing layer neighbor node 61 that is closest to 56. This query message is transferred to node 61. Finally, node 56 is a direct neighbor to node 61. The pseudo-code of searching for a key is algorithm 2 (See Appendix).

3.2.3 Routing by a Resource Vector

Prefix matching techniques are widely used in DHT approaches such as Pastry [17], PHT [18],
etc. SkipMard combines the techniques of the prefix matching and the crossing nearest layer neighbor into a routing algorithm for multi-attribute queries.

To better understand multi-attribute queries in SkipMard, an example is given in Fig.4. If we want to search a node with these attributes: \(\text{cpu-speed} \geq 2.5 \text{ GHz}\) and \(\text{memory-size in [512MB,1GB]}\) and \(\text{disk-capacity} \leq 100\text{GB}\). The requested resource vector will be “321” by mapping Table 2. This query is executed by routing algorithm for a resource vector from the start node 101. In the example, this algorithm first looks at the level 1 of the start node 101. Node 101 lies in the layer 2 at the level 1. The first element of the requested resource vector is 3. Thus, node 101 sends a query message to node 123, which is the node 101’s crossing nearest layer neighbor on the layer 3 at level 1. After node 123 receives this message, it looks up its all neighbors at its level 2. The neighbor node 62 lies in the layer 32 and satisfies the requested. So a query message will be sent to node 62. Likewise, node 49 is obtained to finally satisfy this multi-attribute query.

Multi-attribute resources usually include static attributes and dynamic attributes. To discover dynamic attributes is more difficult than static attributes because of their intermittence and randomization. Most current resource discovery projects use the flooding technique to solve dynamic resource discovery. In SkipMard, we also propose a parallel structured flooding technique in routing algorithm of searching for a vector to discover dynamic attributes. A Time-To-Live (TTL) mechanism is used in flooding to guarantee the efficiency and termination of queries. In this routing algorithm, a user-defined dynamic resource requirement is saved to a data pointer, and a TTL_num is specified as a maximum number of resources that satisfy the requirements or the step length. An internal function Dynamic provides the comparison between user-defined dynamic requirements and a local dynamic table, which is maintained by a node itself. If a node can satisfy the dynamic requirement, then TTL_num will decrease by one, which represents a successful finding. If TTL_num is still greater than zero, then a parallel structured flooding approach will be applied. The query message will be sent to the neighbors of the two sides of the current node in parallel until TTL_num decreases to 0. A dynamic multi-attributes query will return multiple satisfied results and the maximum number will be less than the \(2^{*}\text{TTL}\_\text{num}\). The expected time of searching for a vector is \(O(\log n)\) and the message is \(O(\log n)+O(k)\), where \(k\) is a step length of TTL_num in the parallel structured flooding. The proof is Lemma 2 and pseudo-code is algorithm 3 (See Appendix).

3.3 Analysis

Lemma 1. The search algorithm for a search key in SkipMard with \(n\) nodes takes the expected time \(O(\log n)\) and \(O(\log n)\) messages.

Proof: We first consider that in the SkipMard structure, there exist more than two layers in one level. Let \(p\) represent the maximum range value of any attribute. It is also the total number of layers at \(i\) level. Thus, \(1/p\) represents a fixed probability, which means the chance that one node is a member in one layer at level \(i\). The total nodes \(n\) can be divided into \(p\) groups. The search algorithm for a search key in SkipMard starts from any node on the top level and scans along each linked list for the target. If we determine the target node does not exist in a given level, then it will go down the next lower level until we reach the target node or get the report that this node does not exist but a suitable position is found for this node. The SkipMard’s search algorithm for a search key has the \(O(\log_p n / p)\) expected query time. By the logarithm operation laws, we have

\[
O(\log_p n / p) = O\left(\frac{\log n - \log p}{\log p}\right) = O(\log n)
\]

Thus, the total cost of a search expected time \(O(\log_p n / p)\) is equal to the \(O(\log n)\). During each step of searching for a key, node will send a message only to its neighbor, whose key value is closest to the search key, thus the complexity of the total message number is also equal to the searching expected time \(O(\log n)\).

Lemma 2. The search algorithm for a resource vector in SkipMard with \(n\) nodes takes the expected time \(O(\log n)\) and \(O(\log n)+O(k)\) messages.

Proof: The routing by a resource vector uses prefix matching data structure. This algorithm starts the search from the level 1 of start node and goes up to find the target along the resource vector. Following the proof of Lemma 1, SkipMard search algorithm for a vector also has the \(O(\log_p n / p)\) expected query time, where \(p\) represents the average range length of all resource categories. However, for the complexity of the total message number, this algorithm has \(O(\log n)+O(k)\) messages. \(k\) is a step length of TTL_num in parallel structured
flooding. The reason why we add the additional O(k) message to this algorithm is that it is still possible to exist for the multiple nodes with exactly the same resource vector. Thus this algorithm will have O(log n) + O(k) messages to guarantee the search result.

4. Experimental Results

The experimental results were given to verify SkipMard data structure and algorithms by our simulation tool GOOSE system, which implemented SkipMard in C++ on Windows. Our ideal analytic model is: There are n distinct nodes, which have been configured and installed from different hardware, software and locations. A daemon of SkipMard service is running on all nodes. Each of the nodes is independent and self-managed. All computing resources are registered according to the SkipMard coding method. Based on these conditions, GOOSE simulated this heterogeneous environment to support our analysis.

We measure the neighbor size per node against the number of layers at one level. In Skip List-based approaches, the neighbor size is only related to the level number. But in SkipMard, we create the concept of Crossing Layer Neighbor. So each node will add a little space to save these neighbors. In Fig. 5, there is a linear relationship between neighbor size and number of layers at one level. The expected neighbor size per node is O(m*l), where m is layer number and l is level number.

Another measure is the relationship between the number of routing hops and the node size. We used our tool GOOSE to create a virtual distributed simulation environment with different node size, from 100 to 2000. Every virtual node can receive, handle, transfer and reply a query message. A start node can create this query message and send it to one of its neighbors. We compared the results with two environment settings. One is that each node has 3 attributes and values of each attribute are respectively put into 3 layers. The other is 6 attributes and 6 layers. Fig.6 shows the routing hops against the node size. With the size growing, the number of routing hops increases by one unit. The line of 6-attribute and 6-layer is lower than the line of 3-attribute and 3-layer. It seems the routing for a node with 6-attribute is faster than a node with 3-attribute. In fact, the reason is that a node with 6-layer creates a bigger routing table than that of a node with 3-layer. We use the space consume instead of the message consume. The number of attributes is only related to the number of levels. But the number of layers is closely related to routing table size and message hops. The increase of hops is in logarithm.

5. Conclusion

In this paper, we review the current research work in resource discovery, and present our research work of a natural multi-attribute Peer-to-Peer resource discovery approach (SkipMard). In SkipMard, all resources can be registered as a pair of <attribute, value> and this pair can be mapped into a resource vector. This approach extends Skip Graph algorithm to support multi-attribute queries. The routing algorithms employ the prefix matching technique and approximate closest-point method, which means that for a specified searching key, each node always tries to route this key to the node with the closest key value. The expected time is O(log n) for one-attribute queries and multi-attribute queries.

SkipMard can support well multi-attribute queries. But it does have some limitations. These limitations will be investigated in our future work.

First, all resource-attribute relationship is based on a pre-defined internal system table, which specifies the number of attributes and their range values. These values have to be fixed and known when SkipMard network initializes.
Thus, a dynamic and flexible attribute schema will be considered in our research.

Secondly, we assume SkipMard works in a high degree heterogeneous environment such as global internet. However, the actual research environment is usually a smaller space such as a lab, a few offices, or a local network. In a small space, we might have a homogeneous environment, which means most of the nodes have the same computing resources. Therefore, how to deal with the homogeneous environment will be a problem that we should solve in the next step.

Thirdly, in this proposal, we roughly propose a parallel structured flooding technique to solve the discovery of dynamic multi-attribute resources. In the future work, we need to analyze and optimize it in detail. Some new tools and models such as event-driven model in P/S systems, SFC query and Bloom Filter functions will be further studied.

Finally, SkipMard just provides a straightforward architecture on P2P networks. It still leaves many spaces for the research of potential problems and extensions. Load-balance, security and fault-tolerance are a few key problems. In the next research, we will try to study them.

Reference


Table 2. Pre-define resource-attribute table

<table>
<thead>
<tr>
<th>Resource-Attribute</th>
<th>Value</th>
<th>Pre-defined meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU Speed</td>
<td>1</td>
<td>Below 1.5GHz</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>1.5GHz --- 2.5GHz</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>Above 2.5GHz</td>
</tr>
<tr>
<td>Memory Size</td>
<td>1</td>
<td>Below 512MB</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>512MB --- 1GB</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>Above 1GB</td>
</tr>
<tr>
<td>Disk Capacity</td>
<td>1</td>
<td>Below 100GB</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>100GB --- 250GB</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>Above 250GB</td>
</tr>
</tbody>
</table>

Fig. 1 A SkipMard example with 14 nodes and 3 levels

Fig. 3 An example: searching algorithm for a key 56
Fig 4. An example: searching algorithm for a resource vector 321

Appendix

**SkipMard Algorithm Pseudo Codes**

1. **Comparison**

**Algorithm 1**: approximate closest-point method for searching

```plaintext
min_distance_1(n, search_key, level)
    for(i=3 to n.layers[level])
        Obj[i]=n.Layer[level][i];
    min_val=MAX_INT;
    return_i=0;
    for(i=0 to n.layers[level])
```

274
if (Obj[i]<>NULL)
    tmp=abs(Obj[i].key – search_key)
    if (tmp<min_val)
        min_val=tmp;
    return_i=i;
end;
return Obj[return_i];

2. Routing by a Key

Algorithm 2: routing by a key

searching_by_key(searchOp_by_n, start_node, search_key, level)
    if (n.key=search_key)
        send <searchOp_by_n_reply, n> to start_node;
    while (level>=0)
        near_neighbor=min_distance_1(n, search_key, level);
        if (near_neighbor.key=n.key)
            level--;
        else
            send <searchOp_by_n, start_node, search_key, level> to near_neighbor;
            break;
    if (level <0 )
        send <searchOp_by_n_reply, n> to start_node;

3. Routing by a Resource Vector

Algorithm 3: routing by a vector

searching_by_rv(searchOp_by_rv, start_node, resource_key, dynamic_needs, ttl_num, level)
    while (level<=max_level)
        if (n.rv[level]<> resource_key[level])
            send <searchOp_by_rv, start_node, resource_key, dynamic_needs,
                ttl_num, level> to n.Layer[level][resource_key[level]];  
            break;
        else
            level++;
    if (level>max_level)
        level=max_level;
    isRight=Dynamic(n, dynamic_needs);
    if (isRight=TRUE)
        ttl_num--;
        send <searchOp_by_rv_reply, n> to start_node;
    if (ttl_num>0)
        if (n.Left[level]<>NULL)
            DynamicResourceCounter(searchOp_by_rv, start_node, LEFT, dynamic_needs,
                                     ttl_num, level);
        if (n.Right[level]<>NULL)
            DynamicResourceCounter(searchOp_by_rv, start_node, RIGHT, dynamic_needs,
DynamicResourceCounter(searchOp_by_rv, start_node, side, dynamic_needs, ttl_num, level)

isRight=Dynamic(n, dynamic_needs);
if (isRight=TRUE)
    ttl_num--;
    send <searchOp_by_rv_reply, n> to start_node;
if (ttl_num>0)
    if (side=LEFT)
        if (n.Left[level]<>NULL)
            DynamicResourceCounter(searchOp_by_rv, start_node, LEFT, dynamic_needs, ttl_num, level);
        else
            if (n.Right[level]<>NULL)
                DynamicResourceCounter(searchOp_by_rv, start_node, RIGHT, dynamic_needs, ttl_num, level);