CHAPTER 4

P2P RESOURCE DISCOVERY PROTOCOL FOR MOBILE GRID

A mobile grid consists of a number of autonomously managed local resources under different administrative domains called Virtual Organizations (VO). The volatile and dynamic environment of the mobile grid calls for sophisticated mechanisms for Resource Discovery (RD). As discussed in the previous chapter, the discovery process in mobile grid is required to address issues like node dynamism, node mobility, heterogeneity and frequent changes in attributes.

4.1 INTRODUCTION

RD basically identifies resources that match the requirements of a grid job / user. Resource requirements of a job submitted to a general-purpose parallel distributed system can be determined a priori using historic data or appropriate prediction techniques (Hotovy 1996). The same is true for a mobile grid and these requirements are generally specified in the form of attribute values in queries. The discovery process resolves these queries to identify resources that satisfy the attribute specifications. Early query-based approaches to RD such as Condor (Litzkow 1988), Globus MDS (Foster 1997), Legion (Andrew 1997), etc. select candidate nodes from a pool of resources to satisfy the resource requirements specified by grid users.

Grid being a large-scale resource sharing structure, query resolution requires up-to-date information on widely-distributed resources. The best possible job-resource pair can be chosen only with sufficient information on multiple attributes and subsequent match-making. With increasing number of attributes
describing a resource, the complexity of query resolution becomes high. Organizing nodes in a mobile grid to facilitate query resolution is not trivial because of the dynamism in the participating mobile nodes. Further, continuous and rapid change(s) in some of the attributes of the resources adds to the complexity of the organization process.

The proposed protocol is a query-based approach which employs a non-DHT based overlay for effectively organizing and maintaining up-to-date resource information.

4.2 MOTIVATION FOR NON-DHT OVERLAY

Structured P2P systems are popularly used for RD in large-scale grids. Organization of resource information using P2P nodes is either DHT (Distributed Hash Table)-based or non-DHT based. DHT-based P2P networks have several limitations when used for discovery especially in mobile grids. Various issues in existing DHT-based discovery protocols are as follows:

- DHTs are found to be inefficient in maintaining the indexes for dynamic data on resources especially in a large-scale grid. Few attributes of resources are extremely dynamic requiring frequent updates in DHTs. For example, CPU utilization tends to be a continuously changing attribute value and sometimes bursty. Also mobility in nodes keeps bandwidth attribute of the node changing continuously. The corresponding DHT needs to be updated to reflect the changes in these attributes. This contributes to overheads proportional to the structure of the DHT.

- Different types of attributes like, for example CPU processing capability, bandwidth, and memory capacity of a node may require
different indexing mechanisms. Therefore, systems especially those using multiple DHTs for multi-attribute management result in high maintenance overhead for DHT structures.

- A DHT peer acts only as an index to the appropriate resource and hence it needs to send the information on suitable resource to the requesting node. As the hash indices serve as only secondary index structures, additional mechanism is required for locating nodes in the presence of network delays.

- DHTs require every resource-value pair to have a unique key, limiting its scalability when used for Grid resource discovery (Anand 2010).

- DHTs enable exact match queries while they do not directly support complex queries like range queries, aggregate queries and nearest-neighbor queries. They fail to handle range queries particularly when the requested range is large. This results in lookup cost becoming linear in contrast to the original logarithmic lookup cost of DHTs.

- Systems employing multiple DHTs for multiple attributes scale poorly.

- The hashing mechanism used in DHTs works well with static object identifiers like file names, but is not suitable for dynamic resource information.

- A majority of DHT-based systems employ consistent hashing (Karger 1997) where removal or addition of a node changes only the set of keys owned by that node with adjacent nodes. This leaves all other nodes unaffected and is ideal for a dynamic system. However, in a heterogeneous mobile grid the adjacent node may have poor attributes. Such a node is forced to perform additional data management because its neighbor(s) departed. This calls for frequent load balancing. So
consistent hashing in DHTs may not hold good for a highly dynamic mobile grid.

- DHTs are basically designed as a function of churn rates. Their main consideration is to reduce lookup latency while overlooking the maintenance of the structure, control message overhead and bandwidth usage (Hong 2008).

All the above issues motivate the use of non-DHT overlay. The protocol proposed in this thesis uses a non-DHT overlay for organizing the grid resources, instead of a separate hash index structure on it. Existing approaches using non-DHT overlays as discussed in the Section 3.3.2 of the previous chapter are basically designed for dynamic grids and not mobile or wireless grids. Hence they are not appropriate to address a majority of the issues and challenges in mobile grids.

To handle multiple attributes in the presence of performance and maintenance issues, the protocol uses a non-DHT overlay where the nodes of the mobile grid follow a heap topology. The rationale behind using heap data structure is analyzed here. The topology of the overlay network dictates how the participating nodes can communicate with each other in resolving a query. The mobile grid topology should be robust enough to accommodate frequent joining and leaving of the nodes.

In comparison with ring-based topologies, a heap serves better. N being the number of nodes in a VO, a ring requires O (N) for insertion, detection, and deletion. Insertion and deletion are done in constant time once the link to be modified is identified. However to identify the link, on an average N/2 inspections are required. Similar to a ring other common topologies like non-heap tree, mesh,
butterfly networks, etc. are prone to operational complexities and have limited scalability.

For a heap the order of complexity for the above operations is \( O(\log(N)) \) and is better for large values of \( N \). Of course, for very small values of \( N \), the complexity of the procedure causes more time for heap compared with ring. This advantage of the ring when \( N \) is small can normally be ignored. Thus a heap topology serves better for large-scale dynamic RD.

Organization of resources as a heap is vital because of complex search queries on large number of grid resources characterized by multiple attribute values. Having justified the use of heap overlay for RD, in the next section a preamble for the protocol is presented. What are grid resources, what constitutes the queries, types of queries, how queries are resolved, different ways of handling multiple attributes and various assumptions made for the design and deployment of the protocol are discussed below.

4.3 QUERY-BASED RESOURCE DISCOVERY

4.3.1 Grid Resources and Attributes

A grid resource typically refers to a physical device, service, data item, or any other entity for which discovery procedure is required. A grid resource is a reusable entity that fulfills a job request. It could be a machine, network, or some service that is synthesized using a combination of machines, networks, and software (Klaus 2002).

Basically resources fall into four categories (Koen 2005): (i) fixed – resources with fixed location and properties, e.g. network printer with fixed color and resolution, (ii) replicable – fixed resources combined with replicated files, (iii) mobile – resources with fixed location / variable location, and (iv) dynamic –
resources with fixed location, but their identifiers vary, for e.g., current load and available CPU time, memory capacity.

Resources can become unavailable at any time as the nodes can enter or leave the resource pool and if not taken care of, it affects the QoS levels of the application. This requires that the RD mechanism be dynamic to keep up the QoS levels. Dynamic RD should essentially have strategies for organizing and locating resources of any type. It requires that the resources have a naming scheme to organize and locate them. Naming schemes include unique identifiers, strings, directories and attributes (Koen 2005). Of these, the most powerful naming scheme is the use of attributes.

In order to select a job-resource pair, resource attributes are very useful. Resources are described by a number of predefined attributes that take a value. The attributes simply characterize a resource and help in specifying the heterogeneity of the participating nodes. Further, attributes of a resource are either static or dynamic. Static attributes like OS name, CPU speed of a computing resource refer to resource characteristics that do not change frequently. Dynamic attributes are associated with fast changing characteristics, such as CPU load, free memory and resource access cost. Queries are framed using a combination of any of these resource attributes and RD is the ability to locate resources that comply with a set of requirements given in the query. Various types of queries are discussed below.

4.3.2 Query Types

Grid RD system uses different types of queries and query interfaces. Queries are basically either 1-dimensional or d-dimensional. Since a grid resource is identified by more than one attribute, grid queries are always d-dimensional. Resource query taxonomy discussed in (Ranjan 2008) classifies d-dimensional queries further into four sub-categories. They are:
1. Exact match: specifies the desired values for all attributes sought; for example, `Architecture = ‘x86’ and CPU-Speed =’4GHz’ and RAM=’512MB’ and bandwidth = 1 GB/s’ and OS = ‘Linux’.

2. Range Queries: specifies range values for all or some of the attributes; for example, OS = ‘Linux’ and ‘1.2 GHz’ ≤ CPU-Speed ≤ ‘3 GHz’.

3. Partial Match: specifies only selected attribute values; for example, `Architecture = ‘SPARC’, OS = ‘Linux’.

4. Boolean Queries: specifies Boolean conditions for all or some of the attributes; for example, `RAM ≥ ‘256 MB’ and No. of processors ≤ ‘10’.

As an alternative to single-resource queries as mentioned in the above examples, queries can also look out for multiple resources through a combination of sub-queries, where each sub-query can again be a multi-attribute query. For locating both single as well as multiple resources, query resolution is conceptually the same as discussed below.

### 4.3.3 Query Resolution

The discovery protocol assumes a collection of VOs each with a coordinator node or a local super-peer. It forwards query messages generated by a Grid node to the local super-peer. The local super-peer acts as the coordinator for query resolution. The super-peer can be a powerful node capable of storing and or looking up resource information about the various peers. The local super-peer examines the information service to verify if the requested resources are present in the nodes under its purview. If the resources are available, it sends to the requesting node the IDs of the nodes that satisfy the query specifications. Otherwise, the super-
peer forwards a copy of the query to a selected number of neighbor super-peers, which in turn contact the respective information system, and so on.

Query resolution of multi-attribute range queries is implemented using two techniques, namely iteration and single attribute dominated routing (Paolo 2007). In the iterative approach, a query composed of a number of sub-queries is resolved by resolving each sub-query separately in the proper attribute space. The results are collected and intersected at the query originator node. This approach is simple and easy to implement. Nevertheless, it is inefficient as it involves a large number of comparisons particularly when the number of attributes is very large.

In the single attribute dominated routing approach, one of the attributes which is given more weightage is first tried for satisfying the query specifications. Only those resources that satisfy this attribute are tried next and so on. This approach is better than the first one because it reduces the search time bound considerably. Different systems use different query resolution techniques. A comparison of various query resolution techniques is presented in (Paolo 2007).

4.3.4 Handling Multiple Attributes

Multiple attribute queries involve more complexity when the resource space and the attribute space are particularly very large as in the case of a mobile grid. Iterative approach mentioned above is followed in (Andrzejak 2002) and Xenosearch (Spence 2003) where the sub-query results are intersected at the querying node.

MAAN (Min 2004) follows iterative resolution with single attribute dominated routing based on Chord. MAAN addresses the d-dimensional range query problem by mapping the attribute values to the Chord identifier space via a uniform locality preserving hashing. A separate Chord overlay is maintained for
each attribute dimension. For attributes with numerical values MAAN applies locality preserving hashing functions to assign an identifier in the m-bit identifier space.

Each sub-query is resolved separately and the results are intersected at the querying node following the same iterative approach. To resolve sub-queries single attribute dominated routing is employed. The attribute with the smallest range is selected and the appropriate hub is queried. The query uses the underlying DHT system to locate the resources with the smallest value in the range of the query. It then proceeds to the next values, until the largest value in the range of the query. The query responds with a list of all the resources in the traversed range whose other attributes are also matched as per the query specifications.

A range dividing tree (Ratnasamy 2003) is created for every attribute. All trees are mapped into a single DHT. Each sub-query is resolved separately and then the iterative approach is followed. In Mercury (Bharambe 2004) a routing hub is created for each attribute. The routing hubs are organized as a circular overlay and each node in this overlay is responsible for a range of values for a particular attribute. Query resolution is carried out by a lookup on the DHT of the attribute with the smallest range similar to MAAN.

To resolve multi-attribute range queries, SWORD server (Jeannie 2008) maintains indices over one or more attributes. Each index maps ranges of an attribute to the rows that currently record values of that attribute in the corresponding range. Answering a query then involves using the index to retrieve the rows for the value range of interest for one of the attributes of interest, and then filtering out the rows that do not meet the desired values of the other attributes. The final set of rows contains the measurement reports from the nodes that meet all
criteria in the query. As a variation of this design, the scalability issue of a single server is handled by a cluster of servers against a single central server.

In the protocol proposed in this thesis, query resolution is a combination of iterative and single-attribute dominated routing. Thus it takes the advantages of both the approaches. It differs from the existing approaches by utilizing the min-heap overlays through which the query is routed and resolved. The overlay design of the protocol is discussed in section 4.5. Prior to the design of the overlay, a discussion on the basic mobile grid model is presented next.

4.4 SYSTEM MODEL

4.4.1 Mobile Grid

The mobile grid considered in this thesis is an extension of a cellular network, where the mobile devices can communicate directly with each other via short-range links or using the serving Base Station. Thus it shares a hybrid approach, combining centralized and distributed architectures into a scalable cellular controlled P2P network. It resembles the architecture as proposed in (Ghosh 2007).

The grid constitutes a collection of various service areas called cells occupied by mobile nodes (MN), each governed by a Base Station (BS). Each cell is termed a Basic Service Set (BSS) as per the IEEE 802.11 based wireless LAN nomenclature. All BS are connected by a wired network enabling them to communicate to each other. A dedicated server of the grid called High-level Scheduler (HS) forwards job requests to a central coordinating fixed node (CCN) close to the BS after determining one among the multiple BS.
The mobile grid here adopts a super-peer network model, in which the coordinating node acts as super-peer and all MNs in the respective BSS are peers. The HS submits a typical job-resource request pair to the coordinator to determine the suitable host. The coordinating node in turn, does local scheduling by locating an appropriate node from the cell using the protocol to host the process or a migrated process, if any. It allocates resources to various grid nodes / users as per their requirements.

4.4.2 Assumptions

With the system model defined for the overlay design, assumptions made prior to the design are as follows.

- Resource in this thesis means a computational node or a device although the protocol can be used for any type of resources.

- Dynamic attributes are more challenging and hence while generating queries, static attributes are intentionally omitted. Further, the dynamic attributes are represented in numeric form denoting continuous changes in their values.

- The scheduling paradigm and hence the Resource Discovery (RD) process of a grid fall into two categories namely, global and local ones. By local scheduling issues such as scalability, site autonomy and heterogeneity can be easily addressed. The protocol focuses on local scheduling in one of the administrative domains of the grid. Hence the discovery problem has been localized within a single cluster constituting N nodes, which can be extended for all other clusters in the mobile grid.

- The protocol follows hybrid decentralized architecture. The BS acts as the central server facilitating interaction between peers by
maintaining attribute information of the nodes and passes the same to CCN periodically. However, end-to-end interactions between the peers may take place directly without the intervention of BS through short range wireless communication.

- Optimal resource selection is typically an NP-hard problem (Liu 2004a). Therefore finding an optimal solution in grids with millions of resources is impractical. Instead the protocol attempts at an opportunistic solution obviously a best-effort one, in search of resources depending on their availability.

4.4.3 Data Structure

Binary Heap is the primary data structure used for building the overlay for the protocol. A priority queue based on this structure was first described in (Williams 1964).

Conceptually, a min (max)-heap (Knuth 1973) is a binary tree having the following properties: a) heap-shape: all leaves lie on at most two adjacent levels, and the leaves on the last level occupy the leftmost positions; all other levels are complete and, b) min(max)-ordering: the value stored at a node is less (greater) than or equal to the values stored at its children.

A min-heap or a max-heap of size n can be constructed in linear time and can be stored in an n-element array. When a min-heap implements a priority queue, Find-Min can be performed in constant time, while both Delete-Min and Insert(x) have logarithmic time. Similarly, when a max-heap implements a priority queue, Find-Max can be performed in constant time, while both Delete-Max and Insert(x) have logarithmic time (Atkinson 1986). The protocol takes advantage of these complexities to build an overlay for RD.
Heap facilitates organizing mobile grid nodes based on arithmetic sum of individual values of its attributes. The protocol uses this principle of heaps and creates a two-level heap. In sum-heap, resources are arranged based on their attribute sum and nodes with same attribute sum are clustered together. Resolving a query involves examining resources for individual attributes. All resources whose attributes make a lesser sum are not qualified for assignment to the job and hence not examined at all. This quickens the query resolution process.

At the second level is the attribute heap organizing resources based on one of the attributes. Resources qualified from the first-level sum-heap are examined in attribute heap to determine their fitness for the job. If these resources do not satisfy rest of the attributes, the system searches for resources with higher sum and repeats the search process.

In the subsequent sections of this chapter, a complete discussion on the protocol including the overlay design and protocol is presented. In addition to the proposed protocol, a mathematical model for determining the problem size of RD is devised. It gives the core complexity of the problem and also helps analyze how the protocol is efficient.

4.5 OVERLAY DESIGN

4.5.1 Overlays

Overlay network is a virtual network created on top of the physical network in a networked distributed system. It is done by deploying a set of overlay nodes above the existing IP routing infrastructure. The nodes in the overlay use each other as routers to send the data. Being peers, they cooperate with each other to route packets on behalf of any pair of communicating nodes. Thus an important feature of overlay architecture is the existence of multiple alternative overlay paths.
between a pair of source and destination. This feature is very important especially when overlays are formed for RD in a dynamic grid.

Overlays offer a mix of various features such as robust wide-area routing architecture, efficient search of data items, selection of nearby peers, redundant storage, permanence, hierarchical naming, trust and authentication, anonymity, massive scalability and fault tolerance (Stephanos 2004). Of particular interest to the application of overlays in mobile grid RD is their ability to self-configure in the event of frequent arrivals and departures of mobile nodes in the grid.

An overlay is generally represented as a directed graph where the vertices represent network nodes and an edge represents the knowledge of a node about another node. There exist two alternatives for constructing such an overlay network, either they can be manually configured or they can grow by means of self-organization (Koen 2005). Manually configured overlays are suitable for small-scale networks while self-organizing systems scale better and operate at WAN level thus qualifying for consideration in realizing mobile grids. In the following section overlay design for such a self-organizing system is described in detail.

4.5.2 Overlay Design

In this protocol a non-DHT overlay organizes various peers in a structure, so as to facilitate logical connection among the peers. Two fundamental issues to be addressed in the design of overlays for RD are efficient query routing and balanced load distribution. It is the topology of the overlay that determines the compliance of these two issues. The protocol uses heap topology for its overlay. Given the overlay topology and the substrate network status, the assignment process selects the substrate node and path for each individual overlay node and link respectively.
The overlay structure of the protocol is derived using the attribute information. The peers periodically inform and update their attribute information to their super peer, the BS. The BS is kept aware of the node joins and departures. *This information on peers gathered at the BS is shared with CCN and also broadcast to all peers so that every peer updates its overlay information.* Such distributed overlay information can be exploited for multi-person games as well as for a number of real life critical applications.

By knowing the nearby mobile nodes, invitations could be given for a joint meeting, even if the communication has to be through the BS. Another noteworthy application may be in directing an emergency response team. For example, say a request for ambulance service is received by an occupied ambulance. Using the overlay information, this ambulance can redirect the request to another nearby ambulance. Further, in many encounters of the police with criminals, sometimes police cars converge on the point of confrontation. Here again the overlay information is very useful.

### 4.5.2.1 Sum Heap and Attribute Heap

The structure of the overlay is dictated by the principle of heaps. There are two min-heaps here, one is the sum-heap and the other is the attrib-heap. The sum-heap acts as centralized index server at the CCN and the attrib-heap acts as distributed heap in which nodes are arranged based on any one of the attributes, called primary attribute. Each node is responsible for storing its own attributes and change its neighboring peers as and when required. The change in attributes of neighboring nodes actually redefines the query routing path dynamically from time to time.

In DHT-based P2P networks, it is easy to keep multiple single attribute DHTs and select that parameter with the least records to start checking on other
attributes. If there are k attributes to be checked, the time bound is k x n where n is the total number of nodes while searching for each attribute. The protocol attempts to reduce this bound with this non-DHT overlay design. If attributes are treated with equality, then the following heuristic reduces the search space. When multiple objectives are to be satisfied, a weighted sum (Fatos 2010) approach helps.

Table 4.1 Nodes with 3 attributes

<table>
<thead>
<tr>
<th>Node No.</th>
<th>CPU (GHz)</th>
<th>Memory (GB)</th>
<th>Bandwidth (MBps)</th>
<th>Sum_a</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>7</td>
<td>4</td>
<td>3</td>
<td>14</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>7</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>1</td>
<td>1</td>
<td>6</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>3</td>
<td>4</td>
<td>13</td>
</tr>
<tr>
<td>6</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>7</td>
<td>4</td>
<td>2</td>
<td>2</td>
<td>8</td>
</tr>
<tr>
<td>8</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>9</td>
<td>5</td>
<td>3</td>
<td>3</td>
<td>11</td>
</tr>
<tr>
<td>10</td>
<td>0</td>
<td>4</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>11</td>
<td>5</td>
<td>4</td>
<td>4</td>
<td>13</td>
</tr>
<tr>
<td>12</td>
<td>2</td>
<td>3</td>
<td>2</td>
<td>7</td>
</tr>
<tr>
<td>13</td>
<td>7</td>
<td>4</td>
<td>6</td>
<td>17</td>
</tr>
</tbody>
</table>

The sum-heap is constructed using the weighted sum approach as follows. For example, a unit of one is assigned for each of the attributes of the node. For every participating node, the sum of attributes $\text{Sum}_a$ is obtained. $\text{Sum}_q$ represents the sum of attributes specified in the query. In Table 4.1, some sample nodes with their attributes and attribute sum are shown. A centralized min-heap is constructed at the BS using $\text{Sum}_a$ as the value and individual nodes such that any node has a sum less than at either child.
Let the second heap called the attrib-heap be based on any one of the attributes, say CPU frequency and nodes be arranged in the form of a min-heap. This heap is distributed so that each node of this heap except the root node and leaf nodes are responsible for maintaining information on at most three neighbors namely the parent and children if any. Each of the nodes obtained from comparison of sum in the first heap, now point to attrib-heap. This results in a two-level heap.

![Image of heap diagram]

**Fig. 4.1 Sum Heap for Table 4.1**

![Image of attribute heap diagrams]

**Fig.4.2 Attribute Heaps based on ‘CPU’ value for Table 4.1**
Figures 4.1 and 4.2 represent corresponding sum and attribute heaps for the sample nodes given in Table 4.1. In the sum-heap, multiple nodes with same sum are shown as a linked list and other nodes simply as nodes of the heap. In attrib-heap, the nodes are arranged to preserve the ordering of a min-heap based on their CPU capacity.

If there is a requirement for a node with attributes, 5 GHz clock, 2 GB memory, and 1 USB2 port, the sum of the attributes for the query $\text{Sum}_q$ is $5 + 2 + 1 = 8$ (for simplicity, fraction values in attributes are rounded). Now it is easy to show that all nodes with a sum of 7 or less are not suitable. However, a node with sum greater than or equal to 8 need not satisfy all the requirements. For example, a node with 10 GHz clock but with 1 GB memory and no USB2 port has a sum of 11 but fails to satisfy the memory and USB2 port requirements. This problem is taken care by the attrib-heap. Section 4.6 describes how the protocol utilizes these two heaps for resolving queries.

4.5.2.2 Evaluation of Overlay

The two issues of overlay design namely, query routing and load distribution efficiently are revisited to evaluate the proposed overlay. Considering query routing, upon receiving a query the CCN performs a search in its sum heap first. This is done by computing $\text{Sum}_q$ for the query and searching the heap for nodes satisfying $\text{Sum}_q$. The worst case occurs when the search fails or when the requested $\text{Sum}_q$ is found at the last leaf of the heap. These conditions can only occur after all $n$ nodes in the heap have been considered. Thus lookup operation for $\text{Sum}_q$ has $O(n)$ worst case running time.

A hash table or a binary tree of $\text{Sum}_a$ may be alternatives to sum heap in reducing the search time. However, any solution that speeds up the lookup operation slows down the entire system. This is because the frequent changes in the
Sum of various peers, the dynamics of peers or churn has an adverse effect on the maintenance of hash table or a binary search tree. Further, the advantage of search in sum heap is that it reduces the search bound by filtering candidate nodes for further comparison of individual attributes. Considering this advantage, the worst time complexity may be ignored.

The protocol uses proactive routing scheme where the routing path of a request or query is already determined through the heap structure. The routing state used by an overlay routing algorithm is referred to as the routing table. Being a heap overlay, routing table entries for each node are now no more than three nodes. That is, for every node except the root node and leaf nodes there can be a maximum of three neighbors namely, its parent and at most two children. Thus the overlay involves a smaller routing table which results in easy maintenance.

Considering load distribution, sum heap is basically a centralized index serving as a pointer to a cluster of nodes sharing the same Sum. Only when overlays are used for content distribution or file exchange, load distribution is a significant issue. For RD neither the sum heap nor the attribute heap causes any load imbalance in the system. Therefore, the issue of implicit or explicit load balancing does not arise at all. Further, these heaps are non-DHT-based and hence no collision issue is involved. Other than these fundamental issues, node arrivals and departures in heaps are easy to implement and maintain in both the heaps. They have a logarithmic complexity as discussed below.

When a new device joins or enters the cell its attributes need to be included in both the heaps. There are two possibilities now. If the new device has same Sum as some other existing node(s), its identifier simply gets appended to the existing index on performing a search in sum heap. If Sum is different, it requires a new node to be inserted into the heap to serve as an index for this device. As a heap
by definition is a complete tree, the new node gets added to the bottom level to preserve the structural property of the heap.

If the resulting tree violates min-heap ordering, it has to be reestablished by the ‘heapify’ operation call. This operation starts with a value in leaf node. It moves the value up the path towards the root by successfully exchanging the value with the value in the node above. The operation continues until the value reaches a position where it is greater than its parent, or failing that, until it reaches the root node. Complexity for this operation is $O(\text{height})$ which is $O(\log n)$.

When a device departs from the grid the overlay heaps need to reflect the same. Compared to arrival, departure of a device is very easy to incorporate. A search for the $S_{\text{sum}}$ of the device in the sum heap followed by deleting its identifier from the corresponding index accomplishes the job. Accordingly the attribute heap is also modified. When a departing node is the only node with $S_{\text{sum}}$ in the sum-heap, the corresponding sum node is deleted and the heap is maintained by ‘heapify’ operation. Thus heap overlays proposed here are found to be efficient for meeting the RD objective compared to any other existing mechanism. The next section presents the protocol for RD using the heap overlays.

4.6 THE PROTOCOL

4.6.1 Protocol Description

A mobile grid is basically a finite set of clusters and the proposed protocol accomplishes RD in one such cluster of the grid. A node entering a BSS and which intends to participate in grid indicates its willingness to its respective BS. The BS in turn fixes the new node in the existing heaps (sum and attribute) through heap sort. Similarly, any node that leaves the BSS again causes an update in the heaps. The join and leave operations in a heap consume $O(\log N)$ time. Further, any change in attributes of the nodes is periodically reflected in the heaps and is
incorporated with the same time complexity. With the overlays established and maintained as discussed in previous section, the resource discovery process of the protocol is explained next.

Let \( N \) be the total number of nodes in a cell participating in the grid. The nodes are denoted as \( M_i \) and each node is characterized by \( k \) attribute values namely \( v_1, v_2, \ldots v_k \). The protocol comprises two phases for resolving an exact or partial match query and employs one heap per phase.

Figure 4.3 shows the query getting forwarded to the two heaps. Upon receipt of the query from HS, the CCN computes \( \text{Sum}_q \), the sum of the attributes specified in the query. Then it examines the sum-heap in Phase-1. Those nodes whose \( \text{Sum}_a \) equals \( \text{Sum}_q \), that is \( S = \{s_1, s_2, \ldots s_N\} \) are located, discarding nodes whose \( \text{Sum}_a \) is less than the \( \text{Sum}_q \). Thus the nodes to be examined are reduced from
N to N’, where N’ < N. Thus the protocol filters the candidate resources to a manageable number using the first heap.

Phase-2 involves visiting nodes of S in the order, starting with nodes whose sum of attributes equals $\text{Sum}_q$. The nodes with the same attribute sum are also organized as a heap, $S'$. $S'$ is organized on a predetermined attribute as a heap. The first node in $S'$ is examined to determine whether it satisfies all attributes one by one. If this node is unable to satisfy even a single attribute, the next node in $S'$ is examined. The query thus gets propagated until the desired node is located or until the sum set $S'$ is exhausted.

If the resource needs are not satisfied by any node examined in $S'$, we start looking at the next higher value in sum by visiting the subsequent nodes (neighbors) in the sum-heap and repeating the search. We call the difference in query sum and next sum examined as ‘slack’. Thus slack represents the degree by which query sum varies. An example query resolution for sample nodes in table 4.2 is given below.

Table 4.2 depicts a sample of 10 nodes each with an attribute pair. Suppose we need a processor with attribute pair (1, 4). $\text{Sum}_q$ is 5 and a search in the sum heap reduces the search set S to 3 nodes. S has 3 nodes with attribute pairs: (3, 2), (4, 1), and (2, 3) whose $\text{Sum}_a$ equals 5. Let the attrib-heap be organized based on $v1$ and the set S is reordered now as $S' = (2, 3), (3, 2)$ and (4, 1).

Considering (2, 3) first, we find that it satisfies processor speed, but fails on memory. Then we look at the next faster processor (3, 2) which also fails at memory. We find (4, 1) also failing on memory. So we go back to sum-heap, go one level higher and look at processors with a sum of 6. Here the slack value is 1. This
leads to a single processor (4, 2) which also fails to satisfy the query. Then we try
the processors with the sum of 7, again none. So we try the next higher sum, 8
which again has just one processor (5, 3) which also fails. Next sum 9 has 2
processors (6, 3) and (5, 4). To minimize waste, we try (5, 4) first as per the
reorganized sum set \(S'\) and we succeed.

Table 4.2. Ten nodes with two attributes and their sum

<table>
<thead>
<tr>
<th>Node No. ((M_i))</th>
<th>CPU(GHz) ((v_1))</th>
<th>Memory(GB) ((v_2))</th>
<th>Sum&lt;sub&gt;a&lt;/sub&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>7</td>
<td>4</td>
<td>11</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>3</td>
<td>9</td>
</tr>
<tr>
<td>6</td>
<td>4</td>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>7</td>
<td>5</td>
<td>3</td>
<td>8</td>
</tr>
<tr>
<td>8</td>
<td>5</td>
<td>4</td>
<td>9</td>
</tr>
<tr>
<td>9</td>
<td>2</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>10</td>
<td>7</td>
<td>4</td>
<td>11</td>
</tr>
</tbody>
</table>

There are two typical cases in these searches. First, there may be
multiple nodes satisfying either \(\text{Sum}_q\) or \(\text{Sum}_q + \text{slack}\) with same attribute values. In
that case, we select only the first node that satisfies the requirements. Second, we
may fail to find a suitable node when exhaustively searching all nodes reaching the
leaves of the heap. In this case, we can report failure. Thus always in query-based
RD using this protocol, there can be one of three outputs: success without slack
(exact match), success with slack and failure.
The protocol looks out for resources starting from the lower bound on each of these attributes and when found unable to satisfy them, extends search for higher levels of attributes. This causes the search to continue until a) we find a node or b) we reach all leaves of the heap showing that none satisfies the requirement. So failure to find a node takes full search of the heap. This could be avoided if we store the maximum value of the attributes in the sub heap also in every node.

If the requested attribute value is greater than the maximum, search can be terminated without undergoing a full search of the heap. This will reduce the number of comparisons further.

4.6.2 Handling Multiple Attributes

To handle multiple attributes the protocol is a novel approach in that it is non-DHT based; it does not have multiple indices and hence reduced maintenance overheads; it follows a simple and efficient structure for the overlay; in literature no other work attempts to reduce the search set as attempted in the protocol. With respect to the sum heap this protocol is scalable; its search time is stable as the number of attributes increase. A single node serving as an index in the sum-heap clusters resources whose attributes may differ in attributes individually, but share the same attribute sum.

4.6.3 Handling Mobile Device Characteristics

The protocol works well for a grid as discussed above and takes care of node dynamism and heterogeneity. If the protocol is used in mobile grid, it should also handle node mobility and disconnections prevalent in a mobile grid. Wireless and mobile devices have unique challenges to be overcome to realize grid application for them. The devices are characterized depending on their limited processing power, storage capacity, energy capacity and the transmission range (Ahuja 2006).
Mobility of a node may lead to dynamism in network connectivity. The nodes moving from one cellular region to the other can make the network unstable resulting in intermittent connections and poor bandwidth. Bandwidth being an attribute, its changes is reflected periodically in the heaps.

Mobile nodes may frequently operate in a doze mode or disconnect entirely from the network. In doze mode, a mobile host is reachable from the rest of the system and thus when required, can be induced by the system to resume its normal operating mode (Badrinath 1993). Therefore BS simply needs to inform the MN to change from its doze mode to active mode, when it finds that a particular query request can be satisfied by that node.

When voluntarily disconnected from the network, the node prior to its departure from grid informs BS about it. BS in turn updates the two heaps accordingly and locates a suitable node for hosting the migrated process. Such voluntary disconnections are handled by process migration and the protocol helps in identifying a suitable destination node for the migrated process. Intermittent connections and forced disconnections are out of the scope of this work. In the following section we estimate the complexity of the problem in order to establish how the protocol is efficient in performing resource discovery.

4.7 ESTIMATING PROBLEM SIZE

The protocol has been proposed with a view to support resource discovery on resources with multiple attributes. As the number of attributes increases, it is expected that total number of resources to be compared for resolving a query increases. Here the relationship between these two factors is examined so as to understand how the proposed protocol minimizes the number of comparisons.
Table 4.3 Resource List with $\text{Sum}_a = 4$ and $k = 3$

<table>
<thead>
<tr>
<th>No</th>
<th>CPU GHz</th>
<th>Memory GB</th>
<th>USB2 ports</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>9</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>11</td>
<td>0</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>12</td>
<td>0</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>13</td>
<td>0</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>14</td>
<td>0</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>15</td>
<td>0</td>
<td>0</td>
<td>4</td>
</tr>
</tbody>
</table>

In the sum-heap all nodes whose $\text{Sum}_a$ is less than $\text{Sum}_q$ are rejected. The efficiency of the protocol can thus be estimated by knowing how many nodes share same $\text{Sum}_a$. The number of nodes with the same $\text{Sum}_a$ can be recursively calculated. Thus the theoretical maximum of number of resources with $k$ attributes contributing same $\text{Sum}_a$ can be estimated. The recursive algorithm is devised as follows. Let NOP represent in short, the number of processors. As an example, consider nodes whose $\text{Sum}_a$ is 4 with $k$ attributes, where $k$ is equal to 3. Table 4.3 lists all resources with all possible combinations of attributes.

From Table 4.3 it can be observed that for any $\text{Sum}_a$, there can be a deterministic number of unique combinations of a set of attributes. The two termination conditions for recursion are: (1) NOP with 1 attribute is the number of attributes, which is 1 as in Equation (4.2); (2) NOP with 0 attribute is the number of attributes, which is 0 as in Equation (4.3); then the NOP with $\text{Sum}_a$ on $k$ attributes is
NOP (Sumₐ, k) = \sum_{i=\text{Sum}ₐ}^{0} NOP (\text{Sum}ₐ-i, k-1) \quad (4.1)

NOP (Sumₐ, 1) = 1 \quad (4.2)

NOP (Sumₐ, 0) = 0 \quad (4.3)

Table 4.4 NOP for various Sumₐ and k=2, 3, and 4

<table>
<thead>
<tr>
<th>Sumₐ</th>
<th>NOP</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>k=2</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>8</td>
<td>9</td>
</tr>
<tr>
<td>9</td>
<td>10</td>
</tr>
<tr>
<td>10</td>
<td>11</td>
</tr>
<tr>
<td>11</td>
<td>12</td>
</tr>
<tr>
<td>12</td>
<td>13</td>
</tr>
<tr>
<td>13</td>
<td>14</td>
</tr>
<tr>
<td>14</td>
<td>15</td>
</tr>
<tr>
<td>15</td>
<td>16</td>
</tr>
</tbody>
</table>

Using Equation (4.1) the function for k attributes and any Sumₐ can be obtained. NOP can thus be calculated for any number of attributes and sum. Table 4.4 lists different NOP values for various values of Sumₐ and k. It can be observed
that there are 15 unique processors with $\text{Sum}_a = 4$ and $k = 3$ while 5 processors with $\text{Sum}_a = 4$ and $k = 2$; 66 unique processors with $\text{Sum}_a = 10$ and $k = 3$.

If the query sum is 15 there can be a maximum of 800 nodes each with 4 attributes which need to be examined as the worst case. This implies that the complexity of query resolution increases with the increasing number of attributes as well as the query sum. As an example, for an attribute sum 15, there can be a maximum of 800 unique nodes each with 4 attributes. Consider the case where there are no nodes whose $\text{Sum}_a$ is less than 14. When $\text{Sum}_q = 16$, assuming the theoretical maximum number of unique nodes, the protocol rejects all 800 nodes with attribute sum 15 at one instant. Efficiency of the protocol is very high as it reduces the number of comparisons considerably and thus the search time.

The recursive equation thus helps in estimating the problem size and therefore its complexity. By selecting at the sum-heap as many processors as NOP are considered in case nodes exist with all possibilities. In a real case, it is very rare that number of nodes (with same $\text{Sum}_a$ and $k$) equal NOP. Assuming the rare case, the actual benefit would be less than the theoretical maximum. Also it should be noted that replicas of resources are not considered while calculating NOP.

**4.8 SIMULATION**

This section presents the simulation model for evaluating the performance of the protocol. The proposed model is applicable to mobile grid based on cellular networks where the mobile nodes communicate with each other via the BS. The proposed resource location protocol was experimented using GridSim (Buyya 2002), the Java-based grid simulation toolkit. The Gridsim toolkit facilitates simulating different classes of heterogeneous resources, users, applications, resource brokers and schedulers. It is an efficient tool for simulating application schedulers for single or multiple administrative domains in clusters and grids.
Using GridSim, initially a computational grid was realized to perform experiments on the protocol. The mobile grid was emulated in the same by introducing dynamism of nodes, designating one of the nodes as BS and forming a communication path between various nodes via the BS. The resulting environment consists of multiple users and resources with multiple attributes other than the BS. Each user has different requirements of resources which are sent as a query to the BS. Queries are either generated randomly or formulated manually as explained in Section 4.9.1 depending on the experiment.

The broker entity in GridSim is emulated as BS and it is delegated the responsibility of super-peer. Although a grid machine can have more than one CPU, for simplicity it is assumed as one Processing Element per machine per resource. Henceforth, the terms resource, processor, device and node are interchangeably used. In the following section we show simulation results to indicate how the protocol reduces the search bound for RD by clustering all resources with same attribute sum.

4.9 EVALUATION / PERFORMANCE ANALYSIS

In general a node in a structured P2P network that cannot satisfy the query criteria, forwards the query to other nodes by unicast, multicast, flooding, etc. The node receiving the request forwards the query to other nodes in case it is unable to answer the query. Thus irrespective of whether the search is in structured or unstructured P2P, efficiency of resource location policy is closely dependent on the request forwarding strategy. To evaluate the proposed request forwarding strategy using the protocol, we are interested to find answers for the following questions:

1. The protocol filters the candidate resources to a manageable number using the sum-heap depending on the query sum. What is the relation between query sum and number of resources ignored from examination for
successful attempts (with or without slack) and failed attempts? How to generate queries manually that will fall under these categories? Is there any relation between slack value and number of resources examined?

2. What is the relation between the number of attributes and the query resolution time?

3. How does the query resolution time vary depending on whether the resource sought is located in the sum-heap at higher levels, mid-levels, or at the leaves?

4. Can the proposed dual heap withstand/support frequent node joins and departures?

5. What is the impact of frequency of attribute changes over query resolution time?

Each of these evaluation criteria is presented in detail in the following sections.

4.9.1 Successful and Failed Attempts

Given N nodes in the cell, a query may be resolved by comparing a minimum of 1 and a maximum of N number of nodes for satisfying either a single attribute or multiple attributes. The result of query resolution can be either a successful attempt with or without slack or it can be a failure. For experiment and analysis purpose, generating sample nodes and formulating queries that will fall under any of these three categories is discussed below.

1. Generating nodes: Say for example, the nodes are characterized by three attributes with units GHz, GB, and USB2; for every sum greater than 2, a minimum value of 1 is assigned for each of the attributes. Therefore, a sum of 4
has combinations namely, (2, 1, 1), (1, 2, 1) and (1, 1, 2). Every other sum vector would have at least one zero. These combinations thus get dropped.

2. Generating queries:

   a. While generating queries, let the queries specify a minimum of 1 for each attribute. These are queries satisfied without slack \((QWS)\).

   b. Queries with at least one zero in their attributes are generated in order to get queries that do not get satisfied of a given sum. This query demands \(\text{Sum}_a\) from 2 attributes, whereas all processors have a maximum of \((\text{Sum}_a - 1)\) from 2 attributes. Thus these queries would cause failure at the initial sub tree and hence we have to look for nodes with at least \((\text{Sum}_a + 1)\) sum. These are queries that are satisfied with a slack value \((QS)\).

   c. All queries satisfying the restriction that each attribute is at least 1 are generated to yield success as discussed above. But in practice, it is not necessary that an application poses restrictions on all attributes. Say for example, a simple store and read application does not require processing speed as it is controlled by the speed of the communication link. An application that does not require storage of results when the MN gets switched off does not require USB2 port. Similarly, an application that copies a file from the BS to a USB2 attached drive does not need GB memory. Encouraged by these examples, another set of queries is generated including a zero in one of the attributes. These queries would demonstrate the performance of the protocol under failure mode from the immediate sub-tree.
To generate a query that would eventually fail ($QF$) after looking at all sums in the complete heap, first the maximum GHz, GB, and USB2 attribute values are recorded. A query that demands one more than the maximum recorded can be generated. Such query requirements therefore cannot be satisfied resulting in failure. For example, if all the processors have a maximum of 2 USB2 ports, looking for a processor with 3 USB2 ports would result in global failure.

Using the above procedure a set of resources and sample queries were generated. Figure 4.4 plots various query sum values against the number of resources discarded for the query type QWS. From the figure we observe that the number of resources discarded increases with the increase in query sum. In the sum-heap, with the increase in the query sum, the distance between the node satisfying the sum and the root node increases. This leads to an increased number of nodes getting discarded from examination.
If the query sum is the highest, the node satisfying the same is located in one of the leaves of the sum-heap. A drastic increase in number of nodes ignored from examination and hence a considerable reduction in query resolution time is found in such cases. In the case of QS type of queries, the slope we find in figure 4.5 is a function of slack. Experiments were conducted with different slack values. As the slack value increases, the number of resources discarded increases with increasing query sum. This shows that higher the slack value, greater is the number of resources discarded.

In figure 4.6 for queries satisfied with slack, query sum and number of resources are plotted on X-axis and Y-axis respectively. There are two curves, one showing the resources examined to resolve the query and the other one showing the NOP values for the given $\text{Sum}_a$ and $k=3$ (obtained from table 4.4). NOP values, i.e., the theoretical maximum number of nodes for varying query sum on x-axis and
number of attributes \( k = 3 \) are plotted. For example, NOP being 136 for \( \text{Sum}_a = 15 \) and \( k = 3 \), we find that the number of resources examined is less than 10. This shows the efficiency of the algorithm in reducing the search set to a manageable number.

![Graph](image)

**Fig. 4.6 Success with slack – Comparison with NOP**

In the case of QF type of queries query sum has no effect on query resolution time. This is because failure is reported only after traversing the entire sum-heap. However, we infer from figure 4.7 that number of resources discarded is proportional to total number of resources. As new nodes enter the cellular region, number of resources examined / discarded increase with the increase in number of resources. This is again justified because search continues and failure is reported only after exhaustive checking up to the last leaf.
Fig. 4.7 Query with Failure – number of resources discarded

Fig. 4.8 Query with Failure – average number of resources discarded
In order to get a better view of the results, there were 10 runs made each varying total number of resources as 100, 200, 300, 400 and 500. For each of these resources, attributes were assigned randomly and queries were generated randomly as shown in Appendix 1. Figure 4.8 plots minimum, maximum and average of the 10 values of resources discarded against the total number of resources. As shown in the figure, although the curve plotting minimum values shows linearity as the minimum value depends on the position of the satisfying node controlled by the random process. The average of the values plotted gives a near linear interpretation.

4.9.2 Number of Attributes and Query Resolution

![Graph showing the effect of number of attributes](image.png)

**Fig. 4.9 Effect of number of attributes**

Mobile grids are basically multi-attribute resources. The number of attributes that describe a resource affects query resolution time. This is evident from the figure 4.9 that for k=1 number of resources examined are less and this quantity increases as the k value increases. A resource that satisfies one attribute specified in
a query need not satisfy rest of the attributes. Therefore, other resources are examined so as to evaluate their fitness with respect to all other attributes. Thus number of resources examined increases with the increase in value of $k$ as we increase the total number of resources.

The attribute values for different resources with which figure 4.9 was obtained can be seen in Appendix 2. In this figure there are some values that do not satisfy linearity, say for example the last two sets of data for $k = 1$. This is due to the fact that there are instances where a resource examined may satisfy more than one attribute at the first instance itself without calling for further resources to be examined.

4.9.3 Position of the Resource in a Heap

In the attrib-heap where the nodes are arranged on one parameter, say effective processing speed, query resolution time is saved by rejecting all nodes with processing speed less than the required speed. After these close-to-root processors are rejected, we are forced to check every node until success or we exhaust all nodes and declare failure. This exhaustive search has a time bound of $N$, the number of nodes with the same sum. Therefore, the time required is $\log(N)$ at the sum-heap and $(n)$ at the attrib-heap, where $n < N$ as in the attrib-heap we do not consider nodes disqualified at the level of the sum-heap.

We show in the figures 4.10 and 4.11, a comparison of query resolution time with resources available in the different levels of the sum-heap. Values are plotted by varying number of resources as 10, 20 and 30. We find that number of comparisons required increases with the position of resource in the sum-heap. That is, if the resource is in the higher levels close to the root node in sum-heap, number of comparisons is less and vice versa.
4.9.4 Dynamism of nodes

To speed up the process, every node retains the attributes used earlier when the attributes change calling for a new arrangement. Let the attrib-heap be
built based on processing speed of nodes. For example, if a node had effective free processing speed of 5.0 GHz and it becomes 5.5 GHz as it finished one task, it uses the earlier attributes to locate the current nodes in both the sum-heap and the attrib-heap. Then we find the new position using the new attributes. Now the links are adjusted.

Link adjustments consume constant time in a heap. The search complexity in sum-heap is log (N) and that of attrib-heap is (n). Since we keep the nodes in each attrib-heap small, ‘n’ could be treated as a constant. Thus the time is log (N).

4.9.5 Frequency of attribute changes

The analysis given above holds good for attribute changes also. Hence the order of complexity again is log (N).

4.10 COMPARISON WITH EXISTING NON-DHT SYSTEM

In this section the performance of the protocol and various factors that distinguish the protocol with a similar system (Ranjan 2008) is presented. For clarity and simplification, a representative system using non-DHT overlays similar to the protocol is considered for comparison. There are a number of similarities between the protocol and Mercury that makes this choice of comparison a reasonable one. Mercury supports multi-attribute based search for resources similar to the protocol. Table 4.5 shows a comparison of the protocol with such a system by name Mercury (Bharambe 2004) for resource discovery.

As Mercury uses no hash functions, it leads to non-uniform data partitioning among the participating nodes. This necessitates the use of an explicit load-balancing scheme. In the proposed protocol since we use nodes only to serve as an index and for forming an overlay for routing, the issue of load balancing itself
does not arise. Further, Mercury relies on random sampling techniques for query routing resulting in message overhead required to maintain a separate routing space for each attribute dimension. In our protocol routing is determined by the heap arrangement of nodes. Another limitation in Mercury is that search in a d-dimensional space requires querying every dimension separately and finding an intersection of the same which again incurs high message communication (Ranjan 2008). All these limitations pose a threat to scalability especially when the number of dimensions grows.

Table 4.5 Comparison of the protocol

<table>
<thead>
<tr>
<th>System</th>
<th>Overlay structure</th>
<th>Look-up complexity</th>
<th>No. of routing overlays</th>
<th>Support for Mobility</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dual heap protocol</td>
<td>Min-heap</td>
<td>(O(\log N); N - \text{total number of peers})</td>
<td>Two</td>
<td>Yes</td>
</tr>
<tr>
<td>Mercury</td>
<td>Circular</td>
<td>(O((\log N)/k); k - \text{long distance links}; N - \text{total number of peers})</td>
<td>Multiple</td>
<td>No</td>
</tr>
</tbody>
</table>

Join-leave overhead is another important metric to evaluate systems for multi-attribute-based search. As Mercury involves an overlay for each attribute, node joins and departures lead to high structure maintenance overheads (Shen 2008). In our protocol node joins and departures involve logarithmic complexity only.
4.11 SUMMARY

In this chapter the problem of resource discovery in mobile grid was presented. Resources are described by a number of predefined attributes that take a value. In order to select a job-resource pair, resource attributes are very useful. Depending on the grid user’s job, queries are composed represented as either values or range of values for various attributes. Queries were classified based on whether they are static or dynamic. In general, any discovery protocol forwards such query messages generated by a grid node to the local super-peer.

The local super-peer examines the local information service to verify if the requested resources are present in the nodes under its purview. If the resources are not available, the query is forwarded to other nodes using any of the two techniques: iterative and single-attribute dominated routing. The protocol proposed in this thesis, resolves queries using a combination of iterative and single-attribute dominated routing.

The data structure heap is used for building the overlay for this protocol. It differs from the existing approaches by utilizing the heap overlays through which the query is routed and resolved. It adopts a better strategy using the overlay for filtering nodes that do not satisfy the query criteria. Using the sum heap and attribute heap multiple attribute queries are resolved efficiently. Being a mobile grid the protocol attempts to handle the basic characteristics like heterogeneity, node dynamism and node mobility.

The protocol is described with typical examples and the use of two heaps for query resolution is justified. The simulation results show how the protocol reduces the search bound for resource discovery. We find that number of comparisons required increases with the position of resource in the sum-heap. Also
the frequent changes in attributes are handled with lower time complexities. The protocol takes care of RD in a mobile grid environment as discussed in this chapter and a comparison with Mercury an existing non-DHT system was presented.

The proposed protocol discovers resources dynamically using a systematic and organized strategy. Using the same organization of nodes, range queries can also be resolved and how the protocol handles them is discussed in the next chapter.