Scalable Computing: Practice and Experience
Volume 12, Number 2, pp. 239–255. http://www.scpe.org
ISSN 1895-1767
© 2011 SCPE

DEEPG: DUAL HEAP OVERLAY RESOURCE DISCOVERY PROTOCOL FOR MOBILE GRID

Y. MOHAMADI BEGUM[*]AND M. A. MALUK MOHAMED[†]

Abstract. Inherent characteristics of mobile devices make resource discovery in mobile grid challenging. Adding to the complexity is heterogeneity and hence multi-attribute management of these devices. Decentralized resource discovery using DHTs incur maintenance overheads when changes in attributes are continuous and rapid. The paper proposes a novel dual heap non-DHT overlay (DEEPG) for query resolution dealing with inherent mobile characteristics along with a mathematical model to estimate the problem size. Simulation results show that DEEPG reduces search bound on query resolution and results in limited maintenance overheads. The results also indicate that higher the query sum reduced is the number of lookups for exact match query resolution.

Key words: Mobile grid, Resource discovery, DHT, Multi-attributes, Overlay

1. Introduction.

A computational grid constitutes a number of clusters, each of which may be under different virtual organizations (VO) [16]. Integration of mobile devices into grid computing infrastructure has been widely increasing. Mobile grid models [27, 52, 34] have evolved contributing to computational requirements of grid users. Such mobile grids find applications in domains such as e-health, disaster management, multimedia transfer and processing, and other context-sensitive applications. These applications demand resources involving a variety of attributes including the location of the mobile node. Attributes of grid resources are either static or dynamic. Static attributes of a resource include architecture type, memory configuration, CPU clock frequency, operating system, etc. Dynamic attributes include CPU queue length, memory utilization, CPU utilization, etc.

The volatile and dynamic environment of the mobile Grid calls for sophisticated mechanisms for resource discovery and selection [30]. Discovery process in mobile grid needs to handle issues like node dynamism, node mobility and heterogeneity with frequent attribute changes. Resource requirements of a job submitted to a grid can be determined a priori using historic data or appropriate prediction techniques [46]. Query-based approaches such as Globus MDS [15], Legion [44], Condor [33], etc. select candidate nodes from a pool of resources to satisfy requirements specified by grid users in the form of queries. Query resolution requires up-to-date information on widely-distributed resources. Organizing nodes in a mobile grid to facilitate query resolution is not trivial because of the highly dynamic availability of the participating mobile nodes and continuous, rapid change(s) in some of the attributes of the resources.

Centralized and hierarchical approaches for resource discovery in grid [12] are not suitable for resource discovery with multiple dynamic-query attributes. Such approaches are prone to single point of failure, lead to network congestion and also are not scalable [39]. Alternatively, P2P content distribution technology [2] is widely used for large-scale resource sharing and discovery. Unstructured P2P networks [31] have a variety of applications including resource discovery in grid [22]. In contrast to unstructured systems, DHTs (Distributed Hash Table) are used to design structured P2P systems to avoid flooding of query messages [39] thus reducing network traffic substantially. Some structured DHT-based systems [9, 13, 1] extend existing structured P2P routing substrate [24, 5, 48] for large-scale resource discovery. Systems such as [28, 4] are non-DHT overlays.

The best possible job-resource pair can be chosen only with sufficient information on multiple attributes and subsequent match-making. Query resolution is complex as the number of attributes describing a resource increases. A mapping is required to map attributes in a multi-dimensional space into a single-dimensional one. Space filling curves [51] perform such mapping to facilitate resource discovery. However, as the number of attributes increases, they do not perform well because of locality problem. Further, if the data distribution is skewed, it results in non-uniform query processing load for peers [39].

Various issues in existing DHT-based discovery protocols are as follows: (i) Few attributes of resources are extremely dynamic requiring frequent updates in DHTs. For example, CPU utilization tends to be a continuously changing attribute and sometimes bursty. The corresponding DHT needs to be updated to reflect the changes in this attribute contributing to overheads proportional to the structure of the DHT; (ii) Different types of attributes may require different indexing mechanisms. Therefore, systems especially those using multiple DHTs

[*]Software Systems Group, M.A.M. College of Engineering, Tiruchirappalli, India, Email: ssg_mohamadi@mamce.org
[†]Software Systems Group, M.A.M. College of Engineering, Tiruchirappalli, India, Email: ssg_malukmd@mamce.org

for multi-attribute management result in high maintenance overhead for DHT structures; (iii) 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; iv) DHTs require every resource-value pair to have a unique key, limiting its scalability when used for Grid resource discovery [36]. All these issues motivate the use of a non-DHT overlay. The heap overlay proposed in this paper is a non-DHT overlay used to organize the grid resources, instead of a separate hash index structure on it.

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 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. For a heap the order of complexity is O (log (N)) and is better for large 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. Heap thus qualifies as the most appropriate overlay topology for large-scale resource discovery in comparison with other data structures. Such an organization of resources as a heap is vital because of complex search queries on large number of grid resources characterized by multiple attribute values.

Heap is a data structure with the ability to organize nodes based on arithmetic sum or individual values of its attributes. DEEPG 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 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 search for resources with higher sum and repeat the search process.

The paper proposes a novel protocol to determine suitable node(s) to host a process as a part of local scheduling of the cluster administered by a VO. DEEPG serves the problems associated with mobility and resolves exact match queries for multi-attribute resource discovery in a dynamic grid. In addition to the proposed protocol a mathematical model for determining the problem size is devised. The model calculates total number of unique processors with all possible combinations of attributes together contributing a given attribute sum. A single node serving as an index in the sum-heap, clusters resources whose attributes may differ in attributes individually, but collectively representing the same attribute sum. Simulation results presented show how DEEPG reduces the search bound by clustering all such resources with same attribute sum.

The rest of the paper is organized as follows. Section 2 presents the related work in this area of research. In section 3 the background and system design are discussed. Section 4 describes the DEEPG protocol. In section 5 an estimate of the problem size is presented. In section 6 various experiments and results obtained are evaluated for performance. Section 7 concludes and gives an insight into future enhancements.

2. Related Work. Various decentralized resource discovery techniques in grid, driven by P2P network model are investigated in [39]. Structured P2P resource discovery in grid is done by either using the existing DHTs or by augmenting DHTs to support additional functionality. DHT-based P2P resource discovery systems like Chord [24], CAN [47], Pastry [5] and Tapestry [8] are suitable for single-attribute queries and also fail to handle fast-changing resource attributes. Therefore, some systems augmented these DHT routing substrates to handle multiple-attribute queries. Such systems used either single DHT or multiple DHTs for resolving dynamic multi-attribute range queries. Some example systems are discussed below.

In AdeepGrid [42], a d-dimensional attribute space for both static and dynamic attributes is mapped to a single DHT network. Node dynamicity and changes in dynamic attribute(s) may sometimes map attribute space to a different node and therefore results in maintenance overheads. LORM [19] relies on a single DHT to distribute resource information among nodes in balance with its hierarchical structure and claims to incur low overheads. SWORD [1] also uses a single DHT to locate a set of machines matching user-specified constraints on both static and dynamic node characteristics, including both single-node and inter-node characteristics. SWORD uses the same principle as Mercury [4], although the latter is non-DHT based overlay. Mercury requires explicit load balancing mechanism as it does not apply hashing resulting in non-uniform data partitioning. Also it creates a hub for each attribute and replicates data items on each hub. This is not suitable for a network with high degree of dynamism in nodes.

Xenosearch [13] uses one DHT per attribute and resolves multi-attribute query by aggregating results from every DHT. So it serves as a poor choice when number of attributes of resources is very high. Systems proposed in [3, 43] are based on space filling curves [51] and have drawbacks as discussed earlier. Also they are DHT-based in contrast to SOG [36] which does not use any DHT. However, SOG organizes grid nodes into groups based on their statistical characteristics and cannot be used effectively in a mobile grid where resources vary in their attributes quite often.

Adaptive [25] also supports updates and queries for both static and dynamic resource attributes, again using multiple DHTs. When the attribute value changes to the extent that it is mapped to a new node, the previous mapping is erased. Systems with multiple DHTs are affected by maintenance overheads and [53, 19, 45] attempt at reducing these overheads. Multi-attribute addressable network (MAAN) [9] extends Chord [24] to support multi-attribute and range queries. It maintains multiple DHTs, one per attribute. Nodewiz [41] maintains a single distributed index and hence the update and query traffic is independent of the number of attributes. DIndex [18] has a distributed indexing component in support of range queries.

CONE [6] is a distributed heap-based data structure layered on Chord [24]. CONE uses the DHT only for node joins and departures, and not for querying. A tree based scheduling with heap sort is described in [29]. In majority of these systems, the cost of maintaining the structure of DHT(s) in the presence of potentially frequent node joins and departures is a challenge. Further, in DHTs there is no direct support for complex queries including range queries, aggregate queries and nearest-neighbor queries. Also a majority of these systems employ consistent hashing [26] 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 may not hold good for a highly dynamic mobile grid.

Some non-DHT structured overlays attempt to handle the above issues by organizing overlays based on the attributes of the resources. SkipNet [35] enables systems to preserve useful content and path locality, while Skip tree graphs [17] support aggregation queries and broadcast/multicast operations. Mercury [4] is non-DHT-based and creates a routing hub for each attribute in the application schema while RCT [49] organizes resources on the basis of selected primary attributes. Other non-DHT overlays include ACOM [10], BATON [21] and multi-way tree [20] structures. In [50] an attribute-based overlay is explored where each peer is characterized by a single set of attributes and the peers satisfying a given range or k-nearest-neighbor query are looked up.

Challenges in resource management in mobile grid including resource discovery are addressed in [30, 32, 11]. The M-Grid approach [37] handles disconnected operation service in mobile grid, but fails to address the heterogeneity issue. A proxy-based approach [23] handles mobility by grouping mobile devices located on the same subnet and presenting the group as a single virtual resource. In [27] there is a central entity close to the Base Station (BS) or on the BS that handles instability in mobile grid. To the best of our knowledge we find no approaches that specify a structured topology for mobile nodes to facilitate discovery. DEEPG uses an overlay topology that is simpler and incorporates strategy to handle mobility of devices while performing resource discovery.

3. System Architecture.

3.1. System Model.

The mobile grid for this work resembles the architecture as proposed in [38]. The grid as shown in Fig. 3.1, constitutes a collection of various service areas called cells occupied by mobile nodes (MN), each governed by a 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 BS after determining one among the multiple BS. The BS in turn, does local scheduling by locating an appropriate node using DEEPG to host the process or a migrated process, if any.

To realize a mobile grid, P2P system is an attractive architectural alternative to the traditional client-server computing. A number of critical, real-time, computationally high-end applications can be successfully implemented on a mobile P2P grid. Further, P2P network is self-organizing, which is a key advantage for a dynamic network. They offer efficient search/location of nodes. The mobile grid here adopts a super-peer network model, in which the BS acts as super-peer and all MNs in its BSS are peers. The HS submits a typical

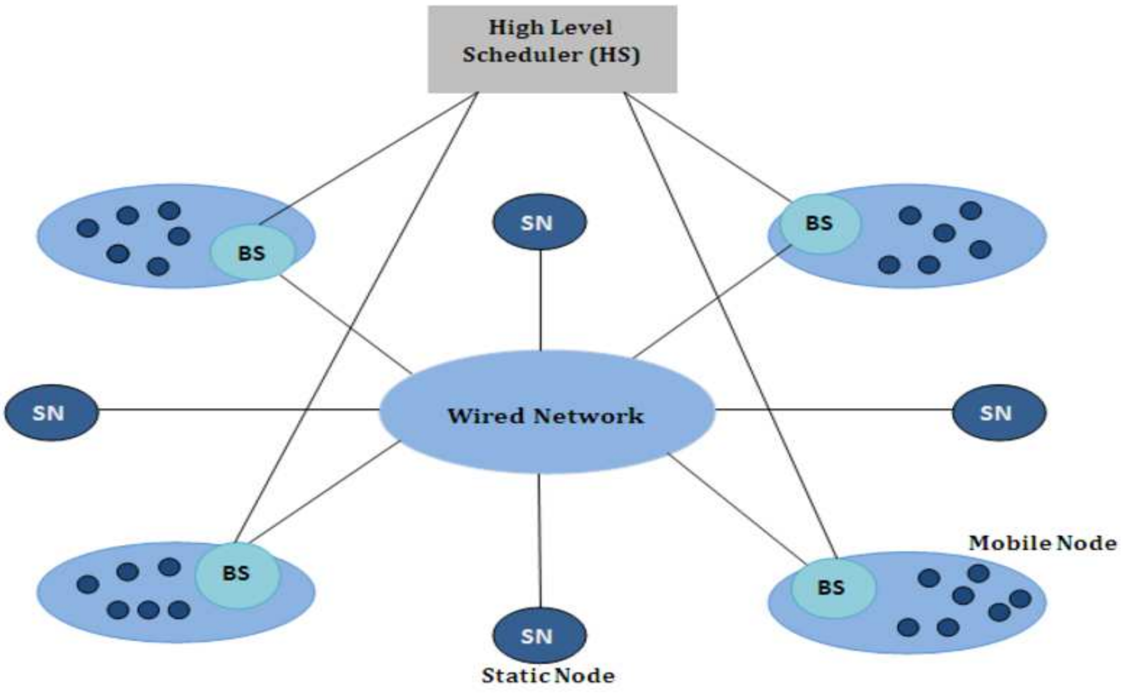

Fig. 3.1: A Mobile Grid connecting HS and BS

job-resource request pair to the BS to determine the suitable host. The BS uses DEEPG to allocate the process peer(s) that matches the specified resource requirements.

3.2. Overlay Design. DEEPG is built on the notion of organizing the participating nodes of a VO to perform local scheduling in it, in the form of a non-DHT overlay. A non-DHT overlay organizes various peers in a structure, so as to facilitate logical connection among the peers. The overlay structure 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 broadcast to all peers so that every peer updates its overlay information.

DEEPG helps in solving the exact match, dynamic multi-attribute resource discovery problem. The attributes considered are dynamic and assumed to be in numeric form. The structure of the overlay is dictated by the principle of heaps. A min-heap is a complete binary tree in which at every node the data stored at the node is no more than the data at either child. 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 BS 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.

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 the 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 [14] approach helps and it can be supplemented with optimization. Using the weighted sum approach, the sum-heap is constructed 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 Sum_a is obtained. A *centralized* min-heap is constructed at the BS using Sum_a as the value and individual nodes such that any node has a sum less than at either child. Table 3.1 shows some sample nodes with three attributes each.

Let the second heap called the attrib-heap, be based on 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

Table 3.1: Nodes with 3 attributes

Node No.	CPU(GHz)	Memory(GB)	Bandwidth (MBps)	Sum_a
1	1	3	1	5
2	7	4	3	14
3	3	2	2	7
4	4	1	1	6
5	6	3	4	13
6	4	0	0	4
7	4	2	2	8
8	3	1	0	4
9	5	3	3	11
10	0	4	0	4
11	5	4	4	13
12	2	3	2	7
13	7	4	6	17

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. Figs. 3.2 and 3.3 represent corresponding sum and attribute heaps for Table 3.1. 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.

—

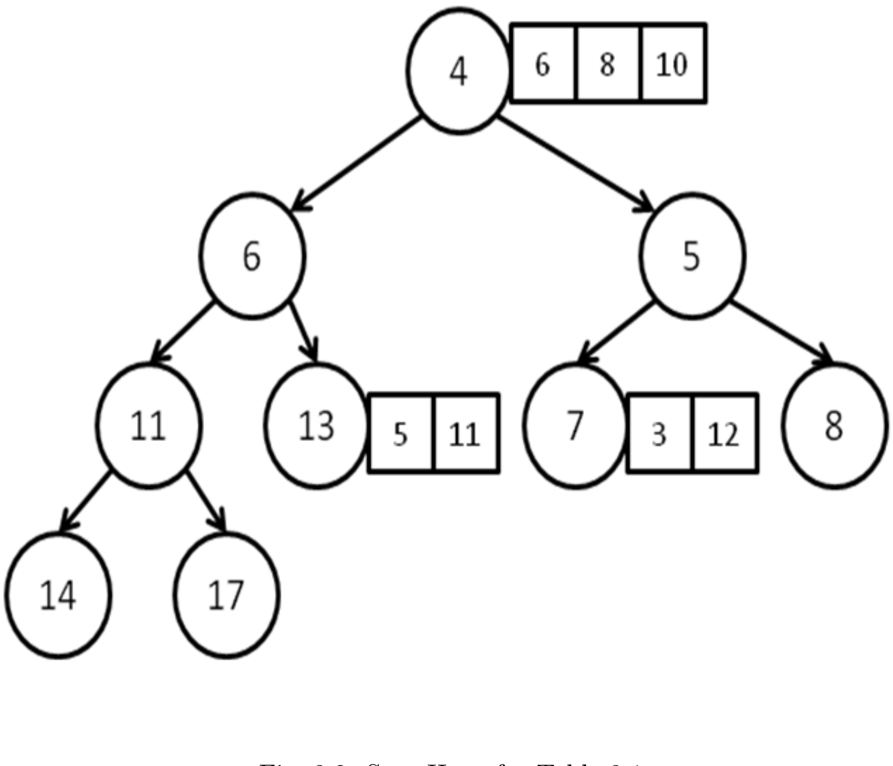

Fig. 3.2: Sum Heap for Table 3.1

If there is a requirement for a node with attributes, 5 GHz clock, 2 GB memory, and 1 USB2 port, the sum 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

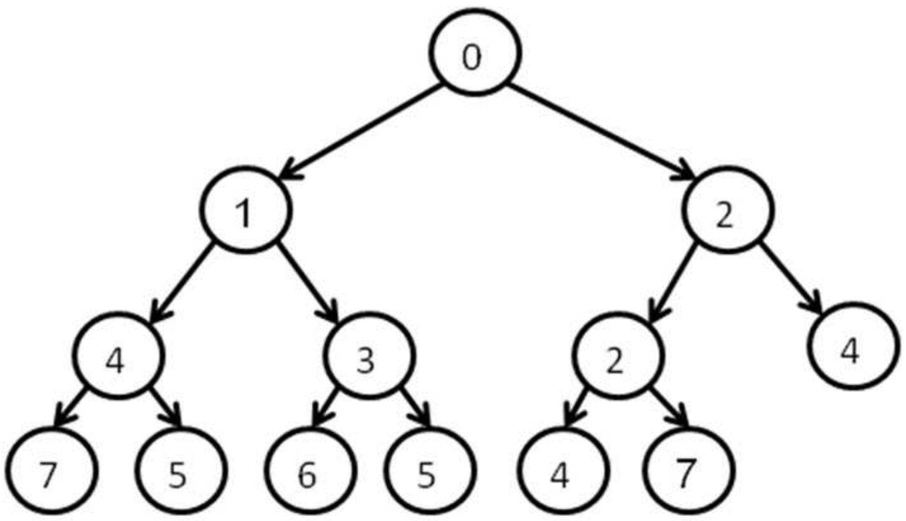

Fig. 3.3: Attribute Heap for Table 3.1

sum of 11 but fails on the memory and USB2 port. This justifies the need for attrib-heap to be constructed. The next section describes how DEEPG utilizes these two heaps for resolving queries.

4. DEEPG Protocol. A mobile grid is basically a finite set of clusters and DEEPG accomplishes resource discovery in one such cluster of the grid. A node entering a BSS and which intends to participate in grid indicates its willingness to the concerned 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. Being organized as a heap, the join and leave operations consume O (log N) time. Similarly any change in attributes of the nodes is periodically reflected in the heaps and is possible with the same time complexity. In sequel, the resource discovery process of DEEPG is described. Let N be the total number of nodes in a cluster, participating in the grid. The nodes are denoted as S_i and each node is characterized by k attribute values namely v_1, v_2... v_k. DEEPG protocol comprises two phases for resolving an exact match query and employs one heap per phase. Fig. 4.1 shows the query getting forwarded to the two heaps. Upon receipt of exact-match-query from HS, the BS computes Sum_q, the sum of the attributes specified in the query. Then it examines the sum-heap in Phase-1. Those nodes whose Sum_a equals Sum_q, that is S $=\{s_1, s_2...s_{N'}\}$ are located, discarding nodes whose Sum_a is less than the Sum_q. Thus the nodes to be examined is reduced from N to $N^{'}$, where $N^{'}$ < N. Thus DEEPG filters the candidate resources to a manageable number using the first heap.

In Phase-2, the second min-heap is considered where nodes are arranged in their increasing order of attribute values. Nodes in S are now visited in the order as suggested in this attrib-heap (S reorganized as $S^{'}$) and begin the search. The first node in $S^{'}$ is located in the heap to make further comparisons so as to satisfy other attributes. If this node is unable to satisfy the query criteria, 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 $S^{'}$, the next higher value in sum is considered by visiting the subsequent nodes (neighbors) in the sum-heap and repeating the search. The difference in query sum and next sum examined is termed as *slack*. Thus slack represents the degree by which query sum varies.

Table 4.1 depicts a sample of 10 nodes each with an attribute pair. Suppose the query involves search for a processor with attribute pair (1, 4). Sum_q is 5 and 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 Sum_a equals 5. Let the attrib-heap be organized based on v_1 and the set S is reordered now as $S^{'}$ = (2, 3), (3, 2) and (4, 1). Considering (2, 3) first, it is found to satisfy processor speed, but fails on memory. Then the next faster processor (3, 2) also fails

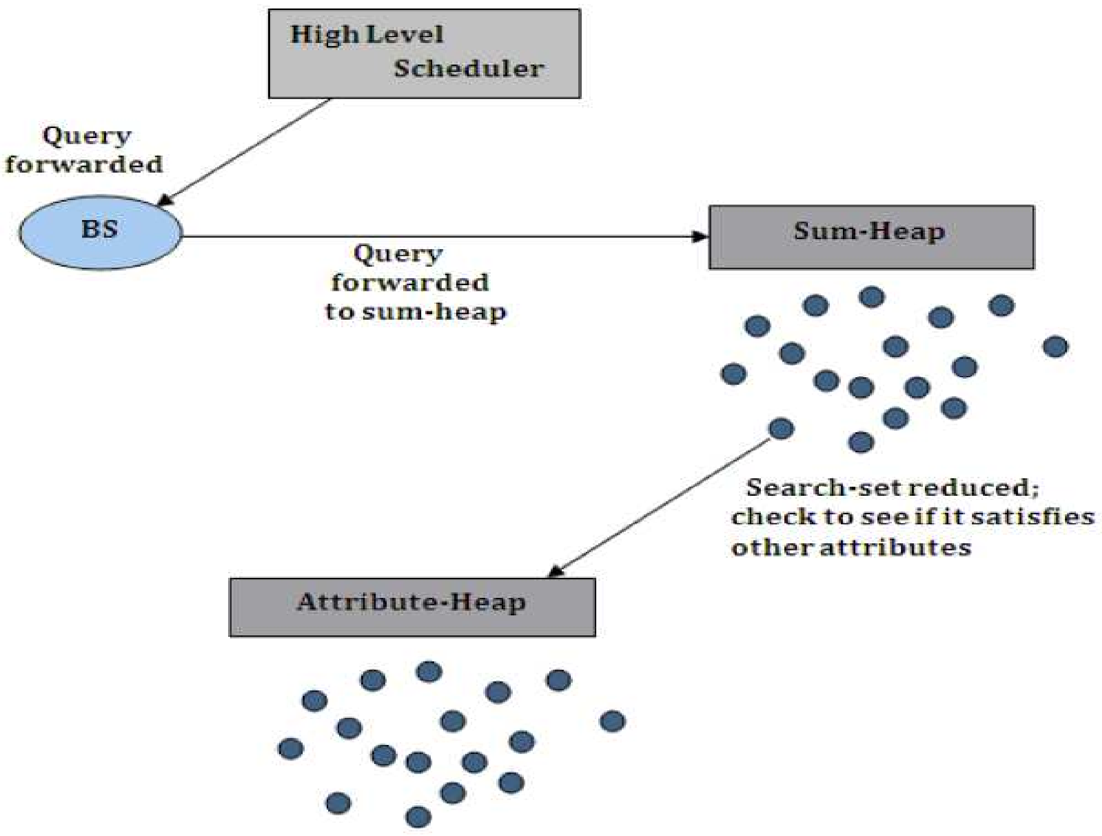
High Level Scheduler
Query forwarded
BS
Query forwarded to sum-heap
Sum-Heap
Search-set reduced; check to see if it satisfies other attributes
Attribute-Heap

Fig. 4.1: Query forwarded to heap overlays

Table 4.1: Ten nodes with two attributes and their sum

Node No.	CPU(GHz) (v_1)	Memory(GB) (v_2)	Sum_a
1	1	3	4
2	7	4	11
3	3	2	5
4	4	1	5
5	6	3	9
6	4	2	6
7	5	3	8
8	5	4	9
9	2	3	5
10	7	4	11

at memory. Similarly processor with attribute pair (4, 1) fails on memory. So sum-heap is revisited to go one level higher to examine 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 the processors with the sum of 7 are tried and again do not qualify the search. Considering the next higher sum 8 there is just one processor (5, 3) which also fails. The next higher sum 9 has 2 processors (6, 3) and (5, 4). To optimize resource use, pair (5, 4) is tried first as per the reorganized sum set $S^{'}$ and this reports success.

There are two typical cases in these searches. First, there may be multiple nodes satisfying either Sum_q or Sum_q+ slack with same attribute values. In that case, only the first node that satisfies the requirements is selected. Second, a suitable node may not be found even after exhaustively searching all nodes reaching the leaves of the heap. In this case, a failure is reported. Alternatively, the HS may redirect the query to another

BS. Thus there can be one of three outputs: success without slack (exact match), success with slack and failure.

The DEEPG protocol works well for a grid as discussed above and takes care of node dynamism and heterogeneity. If DEEPG is used in mobile grid, it should also handle node mobility and disconnections prevalent in a mobile grid. MN could be self location-aware like a cell phone with GPS capability. Such a MN has the facility to handle location specific activities built-in. The GPS devices have the map of the area of use built-in. Any location specific query is answered by interrogating the built-in map. Therefore, it is sufficient to consider the location specific activity of non-GPS MN alone. The issue of mobility especially for location specific activities of non-GPS MN can be handled by having the location of the device linked to the BS. The location of the non-GPS MN is best known to be within the signal range of the BS only and a change in location can be an attribute updated by the BS whenever the device moves away from its range. Similarly, the BS of the non-GPS MN into whose range the MN moves would update the location of the MN and handle location specific activities. Making location specific activity thus anchored to the BS would solve the issue of mobility. Further, mobility may result in unstable network leading to intermittent connections and poor bandwidth. Bandwidth being an attribute, its changes is reflected periodically in the heaps. Intermittent connections and forced disconnections are out of the scope of this work.

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 [7]. 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 which in turn updates the two heaps accordingly.

5. Estimating Problem Size. DEEPG 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.

In the sum-heap all nodes whose Sum_a is less than Sum_q are rejected. The efficiency of the protocol can thus be estimated by knowing how many nodes share same Sum_a. The number of nodes with the same Sum_a can be recursively calculated. Thus the theoretical maximum of number of resources with k attributes contributing same Sum_a can be estimated. The recursive algorithm is devised as follows. Let NOP be the number of processors. Consider for example, nodes whose Sum_a is 4 with k attributes, where k is equal to 3. Table 5.1 lists all resources with these combinations of attributes.

Table 5.1: Resource List with Sum_a = 4 and k = 3

Node No.	CPU(GHz)	Memory(GB)	USB2Ports
1	4	0	0
2	3	1	0
3	3	0	1
4	2	2	0
5	2	1	1
6	2	0	2
7	1	3	0
8	1	2	1
9	1	1	2
10	1	0	3
11	0	4	0
12	0	3	1
13	0	2	2
14	0	1	3
15	0	0	4

From Table 5.1, it can be observed that for any Sum_a, there can be a deterministic number of unique combinations of a set of attributes. Further recursion is terminated when any of these two conditions are met: (1) NOP with 1 attribute is the number of attributes, which is 1 (as in Eq.5.2); (2) NOP with 0 attribute is the number of attributes, which is 0 (as in Eq.5.3); NOP with Sum_a on k attributes is given by

$$NOP(Sum_a, k) = \sum_{i=Sum_a}^{0} NOP(Sum_a - i, k-1) \tag{5.1}$$

$$NOP(Sum_a, 1) = 1 \tag{5.2}$$

$$NOP(Sum_a, 0) = 0 \tag{5.3}$$

Using the recursive equation Eq.5.1 the function for k attributes and any Sum_a can be obtained. NOP can thus be calculated for any number of attributes and sum. Table 5.2 lists different NOP values for various values of Sum_a and k. As an example, it can be observed that there are 15 unique processors with $Sum_a = 4$ and k = 3 while 5 processors with $Sum_a = 4$ and k = 2; 66 unique processors with $Sum_a = 10$ and k = 3. That is, if the query sum is 15 there can be 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.*

Table 5.2: NOP for various Sum_a and k=2, 3, and 4

Sum_a	NOP		
	k=1	k=2	k=3
0	1	1	1
1	2	3	4
2	3	6	7
3	4	10	16
4	5	15	30
5	6	21	50
6	7	28	77
7	8	36	112
8	9	45	156
9	10	55	210
10	11	66	275
11	12	78	352
12	13	92	442
13	14	105	546
14	15	120	665
15	16	136	800

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 $Sum_a$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.

6. Experiments and Results. A node in a P2P network that cannot satisfy the query criteria in general, 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 DEEPG, we are interested to find answers for the following questions:

1. DEEPG 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 that will fall under these categories? Is there any relation between slack value and number of resources examined?
2. What is the relation between 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 as 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?

The proposed resource location protocol DEEPG was experimented using GridSim [40], a Java-based grid simulation toolkit. Using GridSim, a mobile computational grid was realized. The resulting environment consists of multiple users and resources with multiple attributes. Each user has different requirements of resources which are sent as a query to one of the nodes designated as a BS. Depending on the experiment, these queries are either specified or generated randomly. 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.

6.1. Successful and Failed Attempts. Given N nodes in a cell, a query may be resolved by comparing a minimum of 1 and a maximum of N number of nodes. 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, framing sample queries that will fall under any of the above three categories is discussed below.

Say for example, when generating nodes with attribute 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. While generating queries, let the queries specify a minimum of 1 for each attribute. These are queries satisfied *without slack (QWS)*. 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 Sum_a from 2 attributes, whereas all processors have a maximum of (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 (Sum_a+1) sum. These are queries that are satisfied *with a slack value (QS)*.

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 needs some GHz. 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. 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 was 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 (QF)*.

Using the above procedure sample queries were generated for searching from a random set of resources. Figs. 6.1 and 6.2 plot various query sum values against the number of resources discarded for all the three cases of queries QWS, QS, and QF. From these graphs 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 reduction in query resolution time is found in such cases.

Using the above procedure sample queries were generated for searching from a random set of resources. Figs. 6.1 and 6.2 plot various query sum values against the number of resources discarded for all the three cases of queries QWS, QS, and QF. From these graphs 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 reduction in query resolution time is found in such cases.

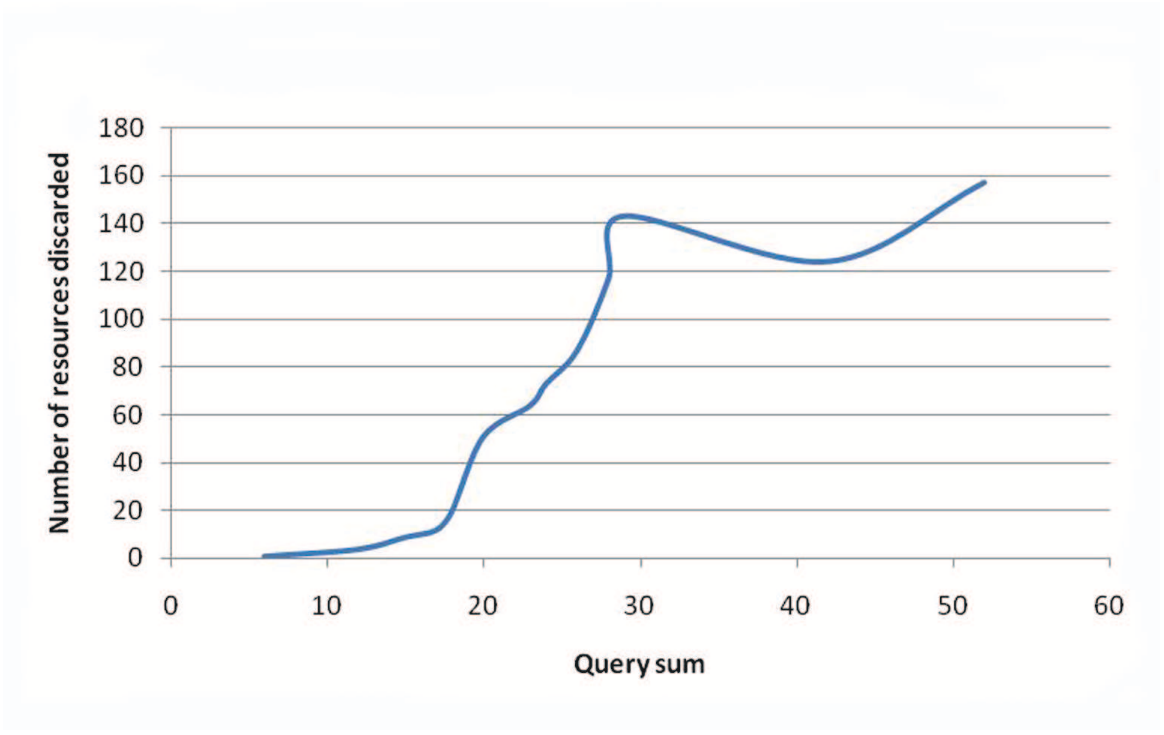

Fig. 6.1: Success without slack: query sum vs number of resources discarded

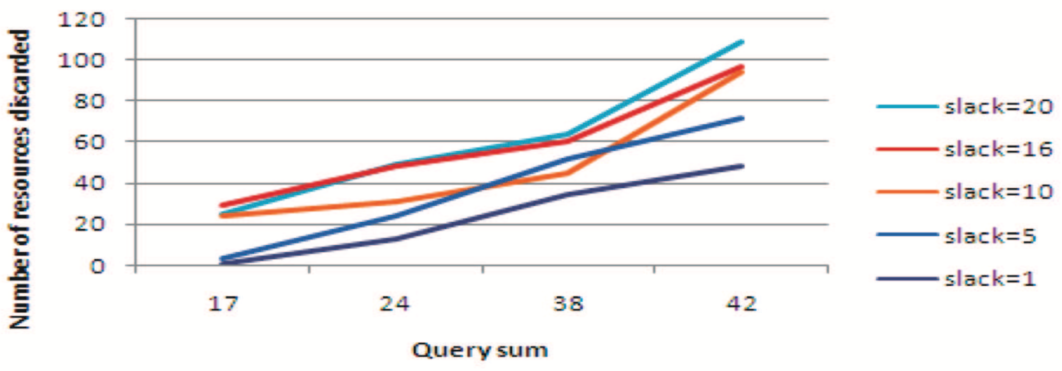

Fig. 6.2: Success with slack:query sum vs number of resources discarded

In the case of QS type of queries, the slope we find in Fig. 6.2 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. Further, higher the slack value, more is the number of resources discarded. In Fig. 6.3 for QS type of queries, number of resources is plotted against number of resources examined for different slack values. Interestingly, we find that there is again a linear relationship between these two values. This indicates that slack value has an effect on both resources examined as well as resources discarded.

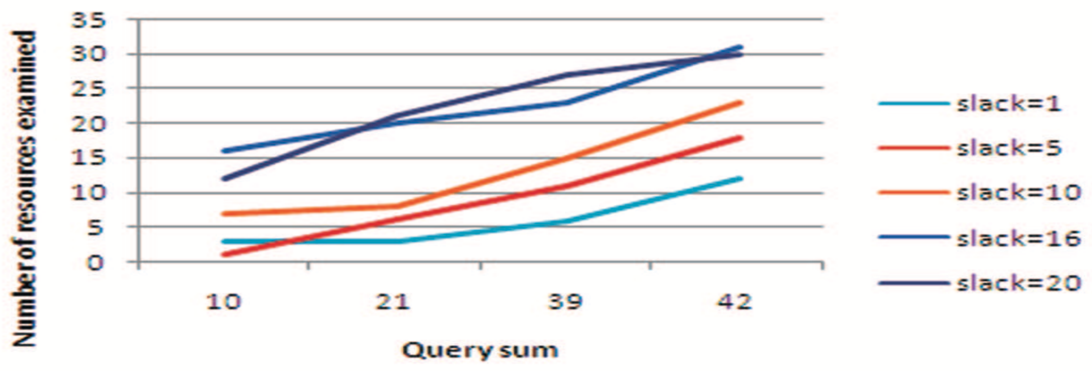

Fig. 6.3: Success with slack:query sum vs number of resources examined

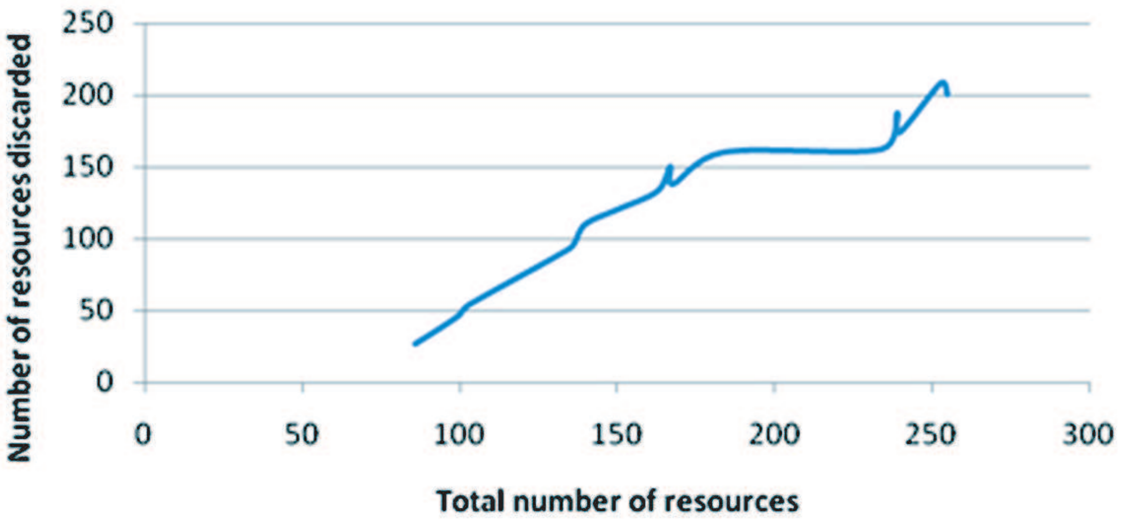

Fig. 6.4: Failure to satisfy a query: number of resources vs number of resources discarded

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 Figs. 6.4 and 6.5 that both number of resources discarded as well as examined is proportional to total number of resources. This is again justified because irrespective of the position of the node representing query sum in sum-heap, search continues and failure is reported only after exhaustive checking.

6.2. Number of Attributes and Query Resolution. The number of attributes that describe a resource affects query resolution time. This is evident from Fig. 6.6 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. As shown in Fig. 6.6 there are some values that do not satisfy

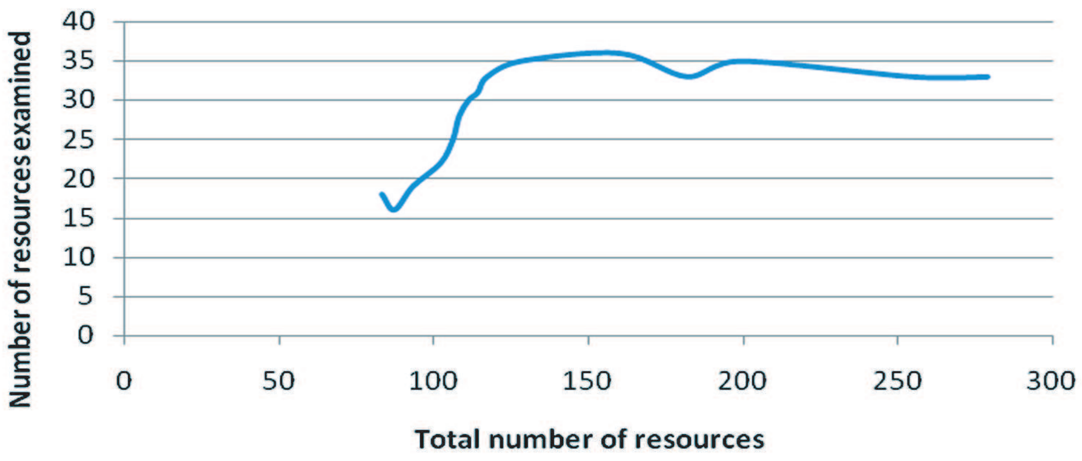

Fig. 6.5: Failure to satisfy a query: number of resources vs number of resources examined

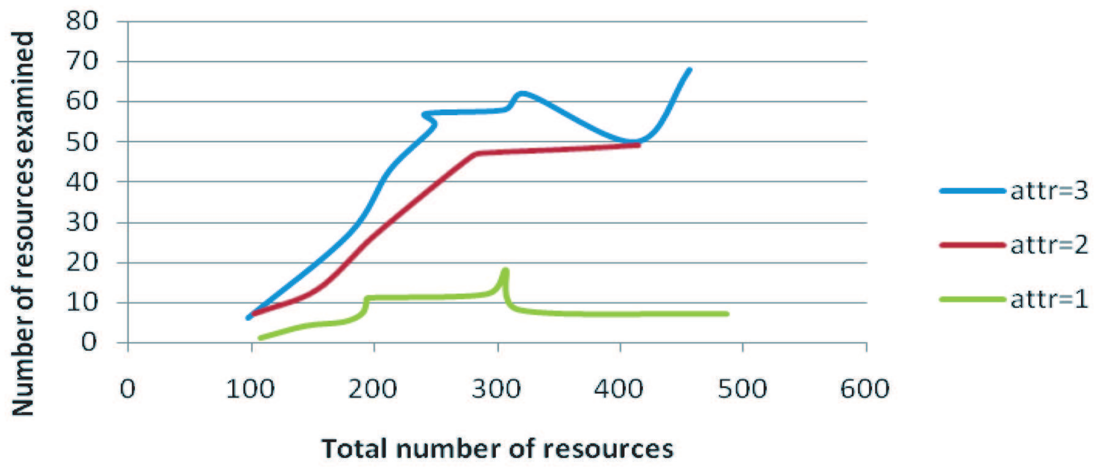

Fig. 6.6: Effect of number of attributes

this linearity. 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.

6.3. 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. Fig. 6.7 shows 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.

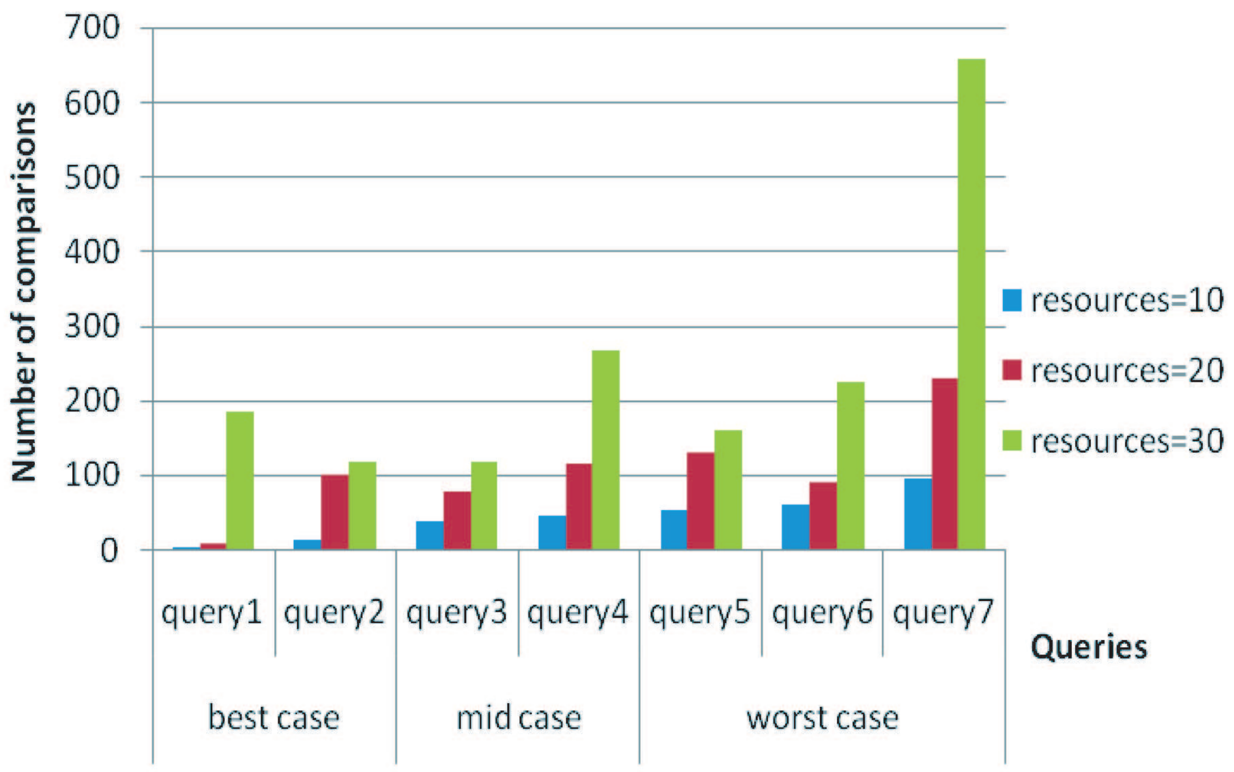

Fig. 6.7: Effect of number of attributes

6.4. 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, it could be treated as a constant. Thus the time complexity is log (N).

As regards a topology such as a ring, it requires O (N) for insertion, detection, and deletion. The insertion and deletion are constant time once the link to be broken is identified. However to identify the link, on an average N/2 inspections are required. For a heap the order is O (log (N)) and is better for large N. For very small N, the complexity of the procedure causes more time for heap compared with ring. This advantage of the ring when N is small is normally ignored.

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

7. Conclusion. There is an interesting property in our approach to resource discovery. In the sum-heap we select all nodes satisfying Sum_q. On matching attributes of these selected nodes with individual attributes in Sum_q, we may find that at least one attribute may not match the query requirement. So we may fail to find a match. So we move to the node in sum-heap, with sum more than the sum of attributes. The difference is called slack. If we need x bandwidth, in the sub tree we check only mobile nodes with bandwidth values x, x+1 ...x+slack. If we take a mobile node with (x + slack + 1) or more bandwidth, at least one other attribute may be smaller. DEEPG can be modified to check for this condition and thus we need not search the sub tree until we reach the leaves. We may reach some leaves and very rarely all the leaves of the sub tree. Undoubtedly this will improve the efficiency of the protocol.

In this paper, DEEPG has been proposed to locate resources with exact and approximate attribute value (when no resources satisfy the requirement exactly). DEEPG can be modified to locate all resources with attribute values larger (or smaller) than a certain number depending on the QoS demands of the grid user. Resource discovery systems built on Chord [24] for N number of nodes, incur a worst-case time complexity of O (N) to locate a node and also O ((log N) 2) to rearrange to accommodate joins and leaves. In contrast, the proposed heap-based overlay has a time complexity of O (log N). Further, when peers in DHTs such as Chord [24], CAN [47], Pastry [5] and Tapestry [8] maintain an index of O (log N) peers, the proposed heap is simpler in that it has a maximum of only three neighbors including its parent and possibly two children. Another factor that makes DEEPG superior is that there is no need for load balancing as done in other DHTs using consistent hashing or any other explicit mechanisms. This is because, we do not create a distributed index like other DHTs, but we simply organize the overlay based on either the sum of attributes or one of the attributes. DEEPG can be extended to handle range queries also. Further, it requires only a small modification to consider static attributes for query resolution.

Acknowledgments. Special thanks to Prof. M.Ibramsha for many substantive discussions on this material and comments on the text.

REFERENCES

[1] J. Albrecht, D. Oppenheimer, A. Vahdat, and D. A. Patterson, *Design and implementation tradeoffs for wide-area resource discovery*, ACM Transactions on Internet Technology, 8 (2008).

[2] S. Androutsellis-Theotokis and D. Spinellis, *A survey of peer-to-peer content distribution technologies*, ACM Computing Surveys, 36 (2004).

[3] A. Andrzejak and Z. Xu, *Scalable, efficient range queries for grid information services*, in Proceedings of the Second IEEE International Conference on Peer to- Peer Computing (P2P 02), 2002, p. 33.

[4] A.R.Bharambe, M.Agarwal, and S.Seshan, *Mercury: supporting scalable multi-attribute range queries*, in In SIGCOMM04: Proceedings of the 2004 conference on Applications, technologies, architectures, and protocols for computer communications, 2004, pp. 353–366.

[5] A.Rowstron and P.Druschel, *Pastry: Scalable, decentralized object location and routing for large-scale peer-to peer systems*, in Proceedings of IFIP/ACM International Conference on Distributed Systems Platforms, 2001, pp. 329–359.

[6] R. Bhagwan, P. Mahadevan, V. G, and G. M. Voelker, *Cone: A distributed heap-based approach to resource selection*, Tech. Report CS2004-0784, UCSD, 2004.

[7] A. B.R.Badrinath and T.Imielinski, *Impact of mobility on distributed computations*, ACM SIGOPS Operating Systems Review, 27 (1993), pp. 15–20.

[8] B.Y.Zhao, J.D.Kubiatowicz, and A.D.Joseph, *Tapestry: An infrastructure for fault-tolerant wide-area location and routing*, Tech. Report UCB/CSD-01-1141, UC Berkeley, Apr 2001.

[9] M. Cai, M. Frank, J. Chen, and P. Szekely, *Maan: A multi-attribute addressable network for grid information services*, J. of Grid Computing, (2004).

[10] S. Chen, B. Shi, S. Chen, and Y. Xia, *Acom: Any-source capacity-constrained overlay multicast in non-dht p2p networks*, IEEE Transactions on Parallel and Distributed Systems, 18 (2007).

[11] D. Chu and M. Humphrey, *Mobile ogsi.net: Grid computing on mobile devices*, in Proceedings of the 5th IEEE/ACM International Workshop on Grid Computing (Grid2004), Nov 2004.

[12] D. Cokuslu, A. Hameurlain, and K. Erciyes, *Grid resource discovery based on centralized and hierarchical architectures*, International Journal for Infonomics, 3 (2010).

[13] D.Spence and T.Harris, *Xenosearch: Distributed resource discovery in the xenoserver open platform*, in Proceedings of the 12th IEEE International Symposium on High Performance Distributed Computing (HPDC03), 2003, p. 216.

[14] A. A. Fatos Xhafa., *Computational models and heuristic methods for grid scheduling problems*, Future Generation Computer Systems, 26 (2010), pp. 608–621.

[15] I. Foster and C. Kesselman, *Globus: A metacomputing infrastructure toolkit*, International Journal of Supercomputer Applications, 11 (1997), pp. 115–128.

[16] I. Foster and C. Kesselman, *The grid: Blueprint for a new computing infrastructure*, Morgan Kaufmann Publishers, 1998.

[17] GonzaLez-BeltraN, P.Milligan, and P.Sage, *Range queries over skip tree graphs*, Computer Communications, 31 (2008), pp. 358–374.

[18] M. Hentschel, M. Li, M. Ponraj, and M. Qi, *Distributed indexing for resource discovery in p2p networks*, in 9th IEEE/ACM International Symposium on Cluster Computing and the Grid, 2009.

[19] H.Shen, A.Apon, and C.Xu, *Lorm: Supporting low-overhead p2p-based range-query and multi- attribute resource management in grids*, in Proceedings of ICPADS, 2007.

[20] H.V.Jagadish, B.C.Ooi, K.L.Tan, Q.H.Vu, and R.Zhang, *Speeding up search in peer-to-peer networks with a multi-way tree structure*, in Proceedings of SIGMOD2006, 2006, pp. 1–12.

[21] H.V.Jagadish, B.C.Ooi, and Q.H.Vu, *Baton: A balanced tree structure for peer-to-peer networks*, in Proceedings of the 31st International Conference on Very Large Data Bases (VLDB), 2005, pp. 661–672.

[22] A. Iamnitchi, I. Foster, and D. C. Nurmi, *A peer-to-peer approach to resource discovery in grid environments*, in Proceedings of the 11th Symposium on High Performance Distributed Computing, 2002, p. 419.

[23] S. Isaiadis and V. Getov, *Integrating mobile devices into the grid: Design considerations and evaluation*, in Proceedings of the International Euro-Par Conference (Euro-Par 2005), 2005.

[24] I.Stoica, R.Morris, D.Karger, M.F.Kaashoek, and H.Balakrishnan, *Chord: A scalable peer-to-peer lookup service for internet applications*, in SIGCOMM 01: Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications, 2001, pp. 149–160.

[25] J.Gao and P.Steenkiste, *An adaptive protocol for efficient support of range queries in dht-based systems*, in Proceedings of the 12th IEEE International Conference on Network Protocols, 2004, pp. 239–250.

[26] D. Karger, E. Lehman, T. Leighton, R. Panigrahy, M. Levine, , and D. Lewin, *Consistent hashing and random trees: distributed caching protocols for relieving hot spots on the world wide web*, in Proceedings of the twenty-ninth annual ACM symposium on Theory of computing, 1997, pp. 654–663.

[27] S. Kurkovsky, Bhagyavati, A. Ray, and M. Yang, *Modeling a grid-based problem-solving environment for mobile devices*, in Proceedings of the IEEE International Conference on Information Technology: Coding and Computing (ITCC-04), 2004.

[28] L.Gong, *Jxta: a network programming environment*, IEEE Internet Computing, 5 (2001), pp. 88–95.

[29] F. Li, D. Qi, L. Zhang, X. Zhang, and Z. Zhang, *Research on novel dynamic resource management and job scheduling in grid computing*, in Proceedings of the First International Multi-Symposiums on Computer and Computational Sciences (IMSCCS 2006), 2006.

[30] A. Litke, D. Skoutas, and T. Varvarigou, *Mobile grid computing: Changes and challenges of resource management in a mobile grid environment*, in Proceedings of the 5th International Conference on Practical Aspects of Knowledge Management (PAKM 2004), Dec 2004.

[31] Q. Lv, P. Cao, and E. Cohen, *Search and replication in unstructured peer to peer networks*, in Proceedings of the 16th international conference on Supercomputing (ICS 02), June 2002, pp. 84–95.

[32] D. Millard, A. Woukeu, F. Tao, and H. Davis, *Experiences with writing grid clients for mobile devices*, in Proceedings of the 1st International ELeGI Conference, 2005.

[33] M.Litzkow, M.Livny, and M. Mutka, *Condor a hunter of idle workstations*, in Proceedings of the 8th Int. Conf. on Distributed Computing Systems (ICDCS 88), June 1988, pp. 104–111.

[34] M. Mohamed, *An object based paradigm for integration of mobile hosts into grid*, International Journal of Next-Generation Computing, 2 (2011).

[35] N.J.A.Harvey, M.B.Jones, S.Saroiu, M.Theimer, and A.Wolman, *Skipnet: A scalable overlay network with practical locality properties*, in Proceedings of Fourth USENIX Symposium on Internet Technologies and Systems (USITS03), 2003, pp. 113–126.

[36] A. Padmanabhan, S. Ghosh, and S. Wang, *A self-organized grouping (sog) framework for efficient grid resource discovery*, Journal of Grid Computing, 8 (2010), pp. 365–389.

[37] S.-M. Park, Y.-B. Ko, and J.-H. Kim., *Disconnected operation service in mobile grid computing*, in Proceedings of the International Conference on Service Oriented Computing (ICSOC 2003), 2003.

[38] P.Ghosh, N.Roy, and S.K.Das, *Mobility-aware efficient job scheduling in mobile grids*, in First IEEE International Workshop on Context-Awareness and Mobility in Grid Computing (held in conjunction with CCGrid 2007), 2007, pp. 701–706.

[39] R. Ranjan, A. Harwood, and R. Buyya, *Peer-to-peer-based resource discovery in global grids: a tutorial*, IEEE Communications Surveys & Tutorials, 10 (2008), pp. 6–33.

[40] R.Buyya. and M.Murshed, *Gridsim: a toolkit for the modeling and simulation of distributed resource management and scheduling for grid computing*, Concurrency and Computation: Practice and Experience, 14 (2002), pp. 1175–1220.

[41] S.Basu, S.Banerjee, P.Sharma, and S.Lee, *Nodewiz: peer-to-peer resource discovery for grids*, in Proceedings of the Fifth IEEE international Symposium on Cluster Computing and the Grid (CCGrid'05), 2005.

[42] S.Cheema, M.Muhammad, and I.Gupta, *Peer-to-peer discovery of computational resources for grid applications*, in Proceedings of the 6th IEEE/ACM International Workshop on Grid Computing (Grid05), 2005, pp. 179–185.

[43] C. Schmidt and M. Parashar, *Flexible information discovery in decentralized distributed systems*, in Proceedings of the 12th IEEE International Symposium on High Performance Distributed Computing (HPDC03), 2003.

[44] A. S.Grimshaw and W. A.Wulf, *The legion vision of a worldwide computer*, Communications of the ACM, 40 (1997), pp. 39–45.

[45] H. Shen and Z. Li, *Spps: A scalable p2p-based proximity-aware multi-resource discovery scheme for grids*, in Proceedings of IEEE Military Communications Conference (MILCOM 08), 2008, pp. 1–7.

[46] S.Hotovy, *Workload evolution on the cornell theory center ibm sp2*, in Job Scheduling Strategies for Parallel Proc. Workshop (IPPS 96), 1996, pp. 27–40.

[47] S.Ratnasamy, P.Francis, M.Handley, R.Karp, and S.Schenker, *A scalable content-addressable network*, in In SIGCOMM01: Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications, 2001, pp. 161–172.

[48] S.Rhea, D.Geels, T.Roscoe, and J.Kubiatowicz, *Handling churn in a dht*, Tech. Report UCB//CSD-03-1299, UC Berkeley, Dec 2003.

[49] H. Sun, J. Huai, Y. Liu, and R. Buyya, *Rct: A distributed tree for supporting efficient range and multi-attribute queries in grid computing*, Future Generation Computer Systems, 24 (2008), pp. 631–643.

[50] M.-T. Sun, C.-T. King, W.-H. Sun, and C.-P. Chang, *Attribute-based overlay network for non-dht structured peer-to-peer lookup*, in International Conference on Parallel Processing (ICPP 2007), 2007.

[51] T.Asano, D.Ranjan, T.Roos, E.Welzl, and P.Widmayer, *Space-filling curves and their use in the design of geometric data structures*, Theoretical Computer Science, 181 (1997), pp. 3–15.

[52] T.Phan, L.Huang, and C.Dulan, *Challenge: Integrating mobile wireless devices into the computational grid*, in Proceedings of the 8th ACM Int. Conf. on Mobile Computing and Networking, (MobiCom 02), 2002.

[53] Z. Xu, R. Min, and Y. Hu, *Reducing maintenance overhead in dht based peer-to-peer algorithms*, in Proceedings of the Third International Conference on Peer-to-Peer Computing (P2P03), 2003.

Edited by: Dana Petcu and Marcin Paprzycki
Received: May 1, 2011
Accepted: May 31, 2011