An Efficient Resource Discovery Methodology for HPGRID Systems

D.Doreen Hephzibah Miriam
Department of Computer Science and Engineering,
Anna University
Chennai, India
Email: doreenhm@gmail.com

K.S.Easwarakumar
Department of Computer Science and Engineering,
Anna University
Chennai, India
easwara@cs.annauniv.edu

Abstract — **An efficient resource discovery mechanism is one of the fundamental requirements for grid computing systems, as it aids in resource management and scheduling of applications. Resource discovery activity involves searching for the appropriate resource types that match the user's application requirements. Classical approaches to Grid resource discovery are either centralized or hierarchical, and it becomes inefficient when the scale of Grid systems increases rapidly. On the other hand, the Peer-to-Peer (P2P) paradigm emerged as a successful model as it achieves scalability in distributed systems. Grid system using P2P technology can improve the central control of the traditional grid and restricts single point of failure. In this paper, we propose a new approach based on P2P techniques for resource discovery in grids using Hypercubic P2P Grid (HPGRID) topology connecting the grid nodes. A scalable, fault-tolerant, self-configuring search algorithm is proposed as Parameterized HPGRID algorithm, using isomorphic partitioning scheme. By design, the algorithm improves the probability of reaching all the working nodes in the system, even in the presence of non-alive nodes (inaccessible, crashed or nodes loaded by heavy traffic). The scheme can adapt to a complex, heterogeneous and dynamic resources of the grid environment, and has a better scalability**

***Keywords-** Peer-to-Peer; Grid; Hypercube; Isomorphic partitioning; Resource Discovery*

I. Introduction

Computational Grids and Peer-to-Peer (P2P) computing are the two popular distributed computing paradigms that have been converging in recent years. Computational Grid is an infrastructure that can integrate the computational resources of almost all kinds of computing devices to form a global problem-solving environment. On the other hand, P2P systems aim at resource sharing and collaboration through direct communication between computers without a centralized server as a medium. Computational Grids and P2P are both resource sharing systems having as their ultimate goal the harnessing of resources across multiple administrative domains. These two distributed systems have some commonalities as well as some conflicting goals as discussed in [4]. They have many common characteristics such as dynamic behavior and heterogeneity of the involved components. Apart from their similarities, Grid and P2P systems exhibit essential differences reflected mostly by the behavior of the involved users, the dynamic nature of Grid resources (i.e., CPU load, available memory, network bandwidth, software versions) as opposed to pure file sharing which is by far the most common service in P2P systems. Although Grid and P2P systems emerged from different communities in order to serve different needs and to provide different functionalities, they both constitute successful resource sharing paradigms. It has been argued in the literature that Grid and P2P systems will eventually converge [12, 17]. The techniques used in each of these two different types of systems will result to a mutual benefit.

Resource discovery is the key requirements in large heterogeneous grid environments, and an effective and efficient resource discovery mechanism is crucial. Traditionally, resource discovery in grids was mainly based on centralized or hierarchical models. Resource discovery could be the potential performance and security bottleneck and single point of failure. The Peer-to-peer systems for discovering resources in a dynamic grid discussed in [19]. Using P2P technology, the resource can be discovered quickly and effectively in grid environment, scalability and robustness can also be improved in P2P Grid.

In this paper, we propose a P2P based Grid resource discovery model, HPGRID system which uses Parameterized HPGRID algorithm, to optimize grid resource discovery and reaches all the grid nodes during searching process even in the presence of non-alive (crashed, inaccessible, experiencing heavy traffic, etc.). The model overcomes the defects of central resource discovery mechanism. The HPGRID model can adapt to the distributed and dynamic grid environments, and has a better scalability. The HPGRID nodes are partitioned isomorphically listing the available resources according to their zones which aids the user to select the needed resource to execute its job rather than traversing the whole grid nodes.

The rest of this paper is structured as follows. Section 2 surveys the related work. Section 3 describes the Hypercubic P2P grid topology. Section 4 describes the HPGRID resource discovery algorithm integrated with Isomorphic partitioning.

Section 5 presents the performance evaluation. Finally, section 6 concludes the paper and presents the future work

II. Related Work

The taxonomy of resource discovery discussed in [21] has identified four main classes of Resource Discovery systems namely centralized, distributed third party, multicast discovery and P2P resource discovery. P2P-based resource discovery systems allow nodes participating in the system to share both the storage load and the query load [18]. In addition, they provide a robust communication overlay. P2P-based Grid resource discovery mechanisms that appear in the literature can be divided into two categories: structured and unstructured [11]. Most proposed systems depend on a structured P2P underlying layer. A structured system however assumes that all pieces of information are stored in an orderly fashion according to their values in a DHT. This is the reason structured systems support efficient resource discovery. However, apart from static resources, Grids include dynamic resources whose values change over time. Whenever the value of a resource attribute stored in a structured system changes, it needs to be republished. If this occurs too often, the cost of republishing becomes prohibitively high.

Iamnitchi et al. proposes resource discovery approach in [7] based on an unstructured network similar to Gnutella combined with more sophisticated query of forwarding strategies which is taken from the Freenet overlay network. Requests are forwarded to one neighbor which are only based on experiences obtained from previous requests, thus trying to reduce network traffic and the number of requests per peer compared to simple query flooding as used by Gnutella. Iamnitchi improves the central control of the traditional grid and adapts fully the decentralized resource discovery in grid environments. However, the limitations are still there in this approach.

Felix Heine et al. propose grid resource discovery approach based ontology and structured P2P technologies in [6]. The approach tackles the semantic problem, but the maintenance of Peer is too high cost because Peer joins and leaves dynamically in structured P2P grid environments. Moreover, the approach focuses on the inherited relationship among grid resource classes and have not discussed the unstructured P2P technologies.

Several P2P schemes, e.g. MAAN [2], NodeWiz [1] and SWORD [8], [16] have been proposed to index and discover Grid resources in a structured P2P network. By using appropriate routing schemes, search queries are routed to the nodes that are responsible for indexing the corresponding resources. Therefore, these schemes scale well to large number of participating nodes. On the other hand, their flat indexing structures pose a major challenge to the global resource monitoring in Grids due to its large-scale and decentralized nature

The HyperCuP system used ontology to organize peers into groups of similar interests using a hypercube topology network [9]. Search queries were forwarded to interest groups to produce a better hit rate and reduce redundant query messages. This approach required complex construction of the structured hypercube topology network. When joining the network, a peer declared its interest so that the network could put the peer into the cluster of its interest. As P2P is a dynamic environment, a peer might change its interest over time. Constantly updating the network would result in high cost. Furthermore, it would be more complicated if peers had more than one interest. A super-peer model for resource discovery services in large-scale grids discussed in [20]. Zheng [22] describes a model for resource discovery among Grids based on the community categorized by application domain. Rozlina [23] discussed the issues related to matrix for measuring the cost and benefit for choosing the right resource discovery mechanism for a P2P systems. The main purpose of the resource discovery Strategy [24] is to improve the efficiency of the implementation of grid system. Abdelkader Hameurlain [25] provides a survey and a qualitative comparison of the most promising approaches (P2P techniques and agent systems) for RD. Viability of Grid systems relies mainly on efficient integration of P2P techniques and mobile agent (MA) systems to bring scaling and decentralized control properties to Grids.

III. *HYPERCUBIC* P2P Grid

A. *Hypercubic P2P Grid Topology*

The Hypercubic P2P Grid Topology is the hypercube structure with additional neighborhood links. In short, we refer Hypercubic P2P Grid as HPGRID.

The Hypercubic P2P Grid nodes have $(k-1)\times(n+1)$) neighbors. Let l , $1 \le l \le k^{n-2}$, be the layer of the HPGRID. Let d be the set of nodes at each layer of the HPGRID, then d = 0, 1, 2, 3. Also, the number of nodes in HPGRID is k^n, and the number of edges are $n2^{n-1}+k^{n-1}$. The HPGRID Topology for n = 3 is depicted in Figure 1. There in, the dashed lines are the additional neighborhood links.

The HPGRID system can be represented by an undirected graph $G=(V,E)$ where $V=\{v_{l.d},......,v_{0.0}\}$.

$$E=\bigcup_{l=1}^{2^{n-2}}\{(v_{l.0},v_{(l+1).2}),(v_{l.1},v_{(l+1).0}),(v_{l.2},v_{(l+1).3}),(v_{l.3},v_{(l+1).1})\}$$

$$\bigcup\{(p,q)\ni:|p\oplus q|_1=1\}$$

where p and q are the binary values of the nodes of HPGRID for a 3D HPGRID (000) denotes node 0and (001) denotes node 1 and so on., and $|p|_1$ denotes the number of ones in p. Here

$$V=\begin{Bmatrix}v_{1.0}(=000),v_{1.1}(=001),v_{1.2}(=010),v_{1.3}(=011)\\ v_{2.0}(=100),v_{2.1}(=101),v_{2.2}(=110),v_{2.3}(=111)\end{Bmatrix}$$

and

$$E = \begin{Bmatrix} (v_{1.0}, v_{2.2}), (v_{1.1}, v_{2.0}), (v_{1.2}, v_{2.3}), (v_{1.3}, v_{2.1}) \\ (v_{1.0}, v_{1.1}), (v_{1.1}, v_{1.3}), (v_{1.3}, v_{1.2}), (v_{1.2}, v_{1.0}) \\ (v_{2.0}, v_{2.1}), (v_{2.1}, v_{2.3}), (v_{2.3}, v_{2.2}), (v_{2.2}, v_{2.0}) \\ (v_{1.0}, v_{2.0}), (v_{1.1}, v_{2.1}), (v_{1.2}, v_{2.2}), (v_{1.3}, v_{2.3}) \end{Bmatrix}$$

In *E*, the first four edges are the additional neighborhood links, and the remaining edges are the hypercubic edges.

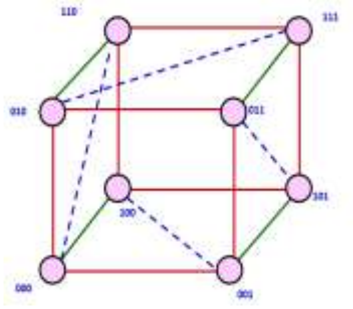


Figure 1. A 3D Hypercubic P2P Grid Topology

B. *Representation of HPGRID*

Generally, the grid model integrated with P2P mode is composed of many Cubic GridPeers [13]. Each Cubic Grid-Peer represents a super management domain. Each Cubic GridPeer controls the access of a group of local computing resources. It plays two roles: one as the resource provider and the other as resource consumer. The resource provider allows its free resources to other Cubic GridPeer (consumer), while the consumer arbitrarily uses its local resources or the free resources of other Cubic GridPeers to carry out its task. The resource discovery model for HPGRID is shown in figure 2. The bottom communities of the model using the traditional grid technologies, and the P2P mode are adapted to interact the information between Cubic GridPeers. Here, Cubic GridPeer (CGP) is equivalent to a super node. When they search resources, the users first query the resources in the domain of Cubic GridPeer. If no query result, the search will be carried out through Cubic GridPeer to query the other Cubic GridPeers with P2P way.

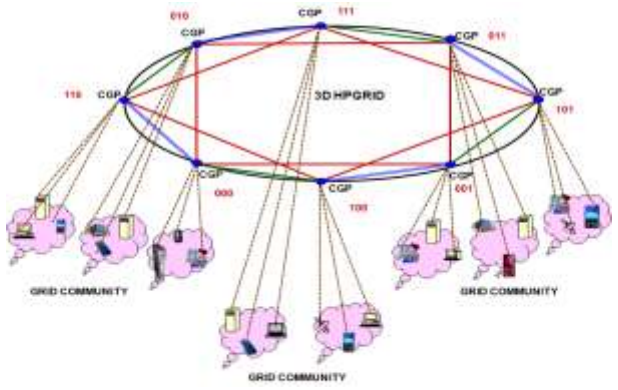
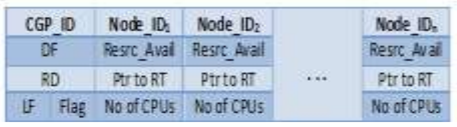


Figure 2. Overview of HPGRID

In HPGRID, each node represents a CGP where each CGP is a collection of Grid Nodes GNs. The GN that belongs to a particular CGP is called Grid Community *GC*. Each Grid node is represented using its own identifier and the identifier of the corresponding CGP. That is, grid node *&* is represented as $(ID_g, ID_{CGP(g)})$

At each CGP in the HPGRID system, it contains a CGP Header in the format represented in figure 3.



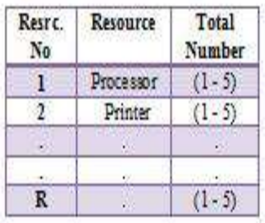
CGP_ID		Node_ID₁	Node_ID₂		Node_IDₙ
DF		Resrc_Avail	Resrc_Avail		Resrc_Avail
RD		Ptr to RT	Ptr to RT	…	Ptr to RT
LF	Flag	No of CPUs	No of CPUs		No of CPUs

Figure 3. CGP Header Format

The CGP Header field description is as follows,

- CGP_{ID} : Cubic Grid Peer Identifier,
- *DF :* Distance factor,
- R*D*: Resource Density,
- *LF* : Load Factor,
- Flag = 0,*CGP* is non-alive, Flag = 1,*CGP* is alive,
- $Node_{ID_i}$: Nodes Identifier where i = 1, 2 …*n.*
- *Resrc_Avail:* : The total number resources available at the node,
- *Ptr to RT* :Pointer to Resource Table,
- *No. of CPUs* : Total number of processing element at the node.

Each CGP contains a common resource table which has the details of all the available resources in its own *GC*. The Resource table format is described in Figure 4. It contains the resource name and number of resources corresponding to its resource number.



Resrc. No	Resource	Total Number
1	Processor	(1 - 5)
2	Printer	(1 - 5)
.	.	.
.	.	.
R	.	(1 - 5)

Figure 4. Resource Table

C. *Parameter at Cubic Grid Peer*

The parameters represent the state of a Cubic Grid Peer (*CGP*) that one must meet the following criteria: they must be small, they must facilitate identifying the fertility of a Cubic GridPeer and they must not divulge resource structure information. Based on parallel application characterization experience, we identified the following parameters.

1) *Distance Factor (DF)*

This gives an idea of how far the target CGP is from the home CGP. A home CGP is defined to be the CGP in which the program and input data are present and to which the output data will go. If it is separated by a large network distance, i.e., high latency and low bandwidth, the staging files and the arriving program and the input files to that CGP will be costly. Another reason why such a factor is important is that tasks in parallel programs might be scheduled on different CGP. Thus there will be some communication between CGP, even though such a situation will be reduced as far as possible by the search algorithm. For tightly coupled applications this may not always be possible and the scheduler might be forced to schedule them on different CGP. This parameter will make CGP between which there is large latency or low bandwidth less desirable to the CGP selector. A high value of this factor makes a CGP less desirable for scheduling.

$$DF = Min\_dist\{h(CGP), n(CGP)\}$$

where h(CGP) denotes home CGP and n(CGP) denotes neighbor CGP.

2) *Resource Density (RD):*

This parameter represents the intensity of computing power per unit communication bandwidth. The lower the value of RD, the more will be the bandwidth between every pair of nodes. This signifies that the resources in those *CGPs* are tightly coupled. For parallel programs that have a communicator in which a small group of processes communicate a lot, a *CGP* with a low value of RD is important. For example, a SMP will have low RD whereas a network of workstations will have high RD. A similar parameter has been used to represent the computation to communication ratio in schedulers of parallel programs.

$$\text{Resource Density} = \frac{\sum CommunicationBandwidth}{\sum ProcessorSpeed}$$

3) *Load Factor (LF)*

This gives the overall load at some instant in that CGP. This is important to take care of the computation component of the parallel program. Thus parallel processes have a high computation aspect compared to communication which would prefer a better value for LF than for RD.

$$\text{Load Factor} = \sum_i \% ProcessorUtilization_i$$

These parameters would be number calculated from information about the state of resources in a particular .CGP

IV. Resource Discovery

In this section, a HPGRID resource discovery model is proposed, and is described as isomorphic partitioning and resource search algorithm.

A. *Isomorphic Partitioning*

The basic idea of isomorphic partitioning is to partition the HPGRID into $k^n/2^i$ number of hyper cubes, where i is the partition step, where $i = 1$ for $n = 3$, $i = 2$ for $n = 4$, and so on. After Partitioning, the HPGRID is divided into 4 zones namely Z_1, Z_2, Z_3 and Z_4. Each zones differ in their higher order bit as shown in figure 5. The processor space is partitioned into higher dimensional isomorphic sub-cubes and keeping the same order of dimension. Isomorphic partitioning strategy for HPGRID systems significantly improves the Subcube recognition capability, fragmentation, and complexity compared to existing methods as discusses in [3].

The following zones show the results of isomorphically partition of the 3D HPGRID into 4 zones containing the following nodes at each zone.

$$Z_1(v_{0.0}, v_{1.0}), Z_2(v_{0.1}, v_{1.1}) Z_3(v_{0.2}, v_{1.2}) Z_1(v_{0.3}, v_{1.3})$$

Thus, there is one bit difference between the neighboring zones. The partitioned HPGRID for 3D has been depicted in figure 6. The resulting partitioned sub-cubes are said to be isomorphic in the sense that they are also n-cubes, and for this reason, they retain many attractive properties of Hypercube networks, includes symmetry, low node degree (*2n*) and low diameter (*kn*).

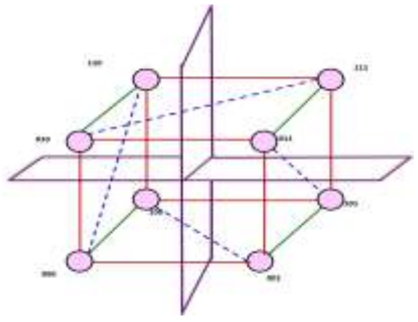


Figure 5. Isomorphic Partitioning of 3D HPGRID System

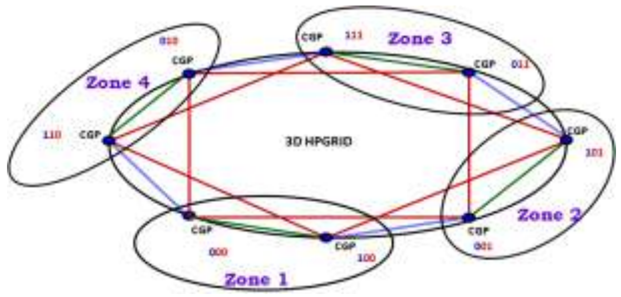


Figure 6. Partitioning 3D HPGRID System.

The 3D HPGRID is isomorphically partitioned into 4 zones containing the following nodes at each zone.

B. *Resource discovery algorithms*

In this section we present a scalable self configuring resource search algorithm (named Parameterized HPGRID

Algorithm) that is able to adapt to complex environments. It is possible to initiate a search request from any of the live nodes. For the reasons of clarity, however the examples used from now on assume that node 0 is the start node, without a loss of generality.

1) The Search Procedure in an HPGRID using Parameterized HPGRID Algorithm

The search procedure starts when a consumer wants to discover a CGP service. The consumer connects to one of the CGP nodes of the system and requests a service (a resource or some resources). The service discovery is tried first inside the requester's own CGP depending on the parameters explained in Section 3.3. If there is no provider, then the request is redirected to other CGPs. Parameterized HPGRID Algorithm gives the pseudo-code in a node when a new resource request message arrives

Algorithm 1: Parameterized HPGRID Algorithm

```
begin
    satisfyRequest=procRequest(message.request);
    if(satisfyrequest)then
        Calculate DF,RD,LF at its CGP;
        if (DF < Threshold) & (RD is lowerbound) &(LF < Acceptable Value) then
            Resource found at the present CGP.
        end
    end
    if(NOT satisfyrequest)then
        if (startNode) THEN
            $v_d$={0,1,2,..,n-1}
            $v_a$={};
            vst={0000};
        else
            $v_d$ =message. $v_d$;
            $v_a$ =message. $v_a$;
            vst =message. vst;
        end
        $v_d$,$d_{alive}$,$n_{non-alive}$=statusNeighbour($v_d$);
        if($n_{non-alive}$>1) then
            $v_{a2}$=addtolist($v_a$, $d_{alive}$);
        else    dalive={};
        for mk=size(vst), k=0 to (($v_d$.size() - $n_{nonalive}$-1) do
            if ($v_d$ is not in vst) then
            message. $v_d$ =creatList(k, $v_d$);
        if ($v_d$ [k] = $d_{alive}$) then
            message. $v_a$ = $v_{a2}$;
        else    message. $v_a$ = $v_a$;
            addToList(vst, $v_d$ [k]);
        end
        msg.vst=vst;
        if (mk < size(vst)) then prop=1;
        for (k=mk to(vst.size() - 1)) do
            sendToNeighbor($v_d$[k],message);
        end
        $v_a$, $n_{non-alive}$=statusNeighbour($v_a$);
        for (j=0 to ($v_a$.size()-$n_{nonalive}$ -1) ) do
            if (neighbor $v_a$ [j] is not parent
                    node) then
                if($v_d$ is not in vst)then
                    EmptyList($v_a$ [j]);
                    prop=1;
            sendToNeighbor($v_a$[j], message);
                end
            end
        end
        if (prop = 0) then
            for (d = 0) to (d < $d_i$) do
                if(d is alive and d is not in
                    vst) then
                    EmptyList(d);
              sendToNeighbor(d,message);
                end
            end
        end
    end
end
```

Algorithm 2: EmptyList(k)

```
begin
    message. $v_d$ ={};
    message. $v_a$ ={};
    message.vst= vst;
    addToList(vst, k);
end
```

2) Description of the Parameterized HPGRID Algorithm

1. When a new service request message is received by a node in the HPGRID system, the function procRequest(message.request) is called. If the request included in the message cannot be satisfied, the node sets the value of satisfyRequest to false and the request will be propagated. Otherwise, satisfyRequest is set to true. Here, it calculates the Distance factor (DF), Resource Density (RD) and Load Factor (LF) to check whether resource is available at CGPs Grid Community. If so, no propagation is performed. The message forwarded is composed of the request (message.request) and two vectors of dimensions (message.v_d and message.v_a). In case the request cannot be satisfied and the node that receives the message is the start node (startNode is true), the list v_d is initialized to v_d = {0, 1,..., n - 1} (the complete set of dimensions) and v_a is initialized to v_a = {} (an empty list). Otherwise, v_d and v_a are initialized to the lists received along with the request message. In both the cases the lists represent the set of dimensions (neighbors) along which the message must be propagated.

2. The node calls the statusneighbors function and reorders the list v_d in such a way that the dimensions corresponding to the nonalive neighbors are located at

the last positions of the list. For example, if v_d = {0,1,2} and the neighbor along dimension 0 and 1 is not responding, then v_d is reordered to {2,1,0}. The statusneighbors also returns with two integer values $n_{non-alive}$ and d_{alive}. The integer value $n_{non-alive}$ represents the number of non-alive nodes in the reordered list v_d. The integer value d_{alive} represents the dimension of the last alive neighbor stored in v_d. For example, if v_d = {2,1,0} and its neighbors in dimensions 0 and 1 are non-alive nodes, $n_{non-alive}$ = {2} and d_{alive} ={2}.

3. If the number of non-alive neighbors ($n_{non-alive}$) is more than one, the node calls the addToList(v_a, d_{alive}) function. This function appends d_{alive} to the end of the list v_a and returns to the new list (v_{a2}) else it make the d_{alive} empty.

4. When a new service request message is received by a node in the HPGRID system, the function procRequest(message.request) is called. If the request included in the message cannot be satisfied, the node sets the value of satisfyRequest to false and the request will be propagated. Otherwise, satisfyRequest is set to true. Here, it calculates the Distance factor (DF), Resource Density (RD) and Load Factor (LF) to check whether resource is available at CGPs Grid Community. If so, no propagation is performed. The message forwarded is composed of the request (message.request) and two vectors of dimensions (message.v_d and message.v_a). In case the request cannot be satisfied and the node that receives the message is the start node (startNode is true), the list v_d is initialized to v_d = {0, 1,..., n - 1} (the complete set of dimensions) and v_a is initialized to v_a = {} (an empty list). Otherwise, v_d and v_a are initialized to the lists received along with the request message. In both the cases the lists represent the set of dimensions (neighbors) along which the message must be propagated.

5. The node calls the statusneighbors function and reorders the list v_d in such a way that the dimensions corresponding to the nonalive neighbors are located at the last positions of the list. For example, if v_d = {0,1,2} and the neighbor along dimension 0 and 1 is not responding, then v_d is reordered to {2,1,0}. The statusneighbors also returns with two integer values $n_{non-alive}$ and d_{alive}. The integer value $n_{non-alive}$ represents the number of non-alive nodes in the reordered list v_d. The integer value d_{alive} represents the dimension of the last alive neighbor stored in v_d. For example, if v_d = {2,1,0} and its neighbors in dimensions 0 and 1 are non-alive nodes, $n_{non-alive}$ = {2} and d_{alive} ={2}.

6. If the number of non-alive neighbors ($n_{non-alive}$) is more than one, the node calls the addToList(v_a, d_{alive}) function. This function appends d_{alive} to the end of the list v_a and returns to the new list (v_{a2}) else it make the d_{alive} empty.

7. For each position k in the list v_d represents an live neighbor node, the node calls the createList(k, v_d) function which creates a new list composed of all the dimensions located after position k in the ordered list v_d. In other words, if the number of elements in v_d(v_d.size()) is q, the function returns [{ v_d [k+1], ..., v_d [q-1]}] For example, if v_d = {2,1,0} and k = 1, the call to createList(k, v_d) will return {1,0 }. Also for each alive neighbor, the v_a list is initialized. The request, v_d, and v_a are sent to the corresponding neighbor in the v_d[k] dimension inside a new message by calling the sendToNeigbor(v_d [k], message) function. See Figure 7 (a complete example using Parameterized HPGRID Algorithm) where the start node (000) sends v_d ={1,0} and v_a ={2} to its last alive neighbor (the only one in this case).

8. Finally, the node propagates the request to each of the neighbors along with v_a dimensions only if the corresponding neighbor is not its parent node. Now this propagation takes place under two cases.

Case 1: If the number of elements in v_d is not equal to 0 i.e., v_d.size() !=0, then the request travels inside a message together with v_a and v_d as empty lists.

Case 2: If the number of elements in v_d is equal to 0 i.e., v_d.size()==0, then the node calls the statusneighbors(v_a) function and reorders the list v_a in such a way that the dimensions corresponding to the $n_{non-alive}$ neighbors are located at the last positions of the list. The status neighbors(v_a) also returns two integer values $n_{non-alive}$ and d_{alive}. The integer value $n_{non-alive}$ represents the number of non-alive nodes in the reordered list v_a. The integer value d_{alive} represents the dimension of the last alive neighbor stored in v_a.

9. For each position k in the list v_a that represents a live neighbor node, the node calls the createList(k, v_a) function which creates a new list composed of all the dimensions located after position k in the ordered list v_a. Also, for each alive neighbor, the v_d list is initialized as empty. The request, v_d and v_a are sent to the corresponding neighbor in the v_d[k] dimension inside a new message by calling the sendToNeigbor(v_a [k],message) function. For the remaining elements in the list v_a represents non-alive neighbor node, the request travels inside the message together with v_a and v_d as empty lists.

10. Propagating the requests in this way, the effect of non-alive nodes is reduced. Consequently, the algorithm tries to isolate the nodes that are in a non-alive state so that they become leaf nodes (if it is possible) under such circumstances, each node has only one non-alive neighbor, and then all live nodes can be reached. On the other hand, the nodes that are unreachable because of inaccessible or crashed nodes along a path to them, can be reached eventually via other nodes - using the v_a list.

V. PERFORMANCE EVALUATION

In this section, we present the simulation results using the GridSim simulator [14]. The Parameterized HPGRID algorithm has been implemented for three and four dimensional HPGRID system. This algorithm has been evaluated in comparison with the existing algorithm described in [5]. In order to evaluate we describe the following test cases.

- BEST CASE: All the nodes in the network are in alive state and the request has been satisfied at the first hop itself.
- AVERAGE CASE: Some of the nodes are in alive and some are in non-alive state and the request is satisfied at the next hop.
- WORST CASE: Most of the nodes are in non-alive state and the request is satisfied at the last zone.

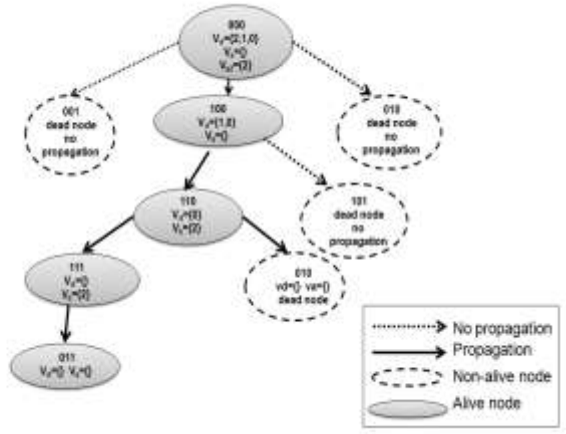

000 V_d={2,1,0} V_r={} V_a={2}
001 dead node no propagation
100 V_d={1,0} V_r={}
010 dead node no propagation
110 V_d={0} V_r={2}
101 dead node no propagation
111 V_d={} V_r={2}
010 vd={} va={} dead node
011 V_d={} V_r={}
No propagation
Propagation
Non-alive node
Alive node

Figure 7. A complete example using Parameterized HPGRID Algorithm. A request of resource started at node 000 in a three-dimensional hypercube

Simulation has been done on a 3D HPGRID for the worst case keeping the nodes source nodes as (000) making the zeroth, first and second dimension nodes as non alive depicted in Figure 8, the following figure 9 gives the complete traversal example for 3D HPGRID system starting from node (000), having all its neighbor non alive namely (001,010,100) except the node present in the additional link node 110 is alive. Sample part of the gridsim output for resource discovery on a best cast 4D HPGRID is shown in Figure 10. Resource search path traversal for the best case in the 4D HPGRID system is depicted in figure 11.

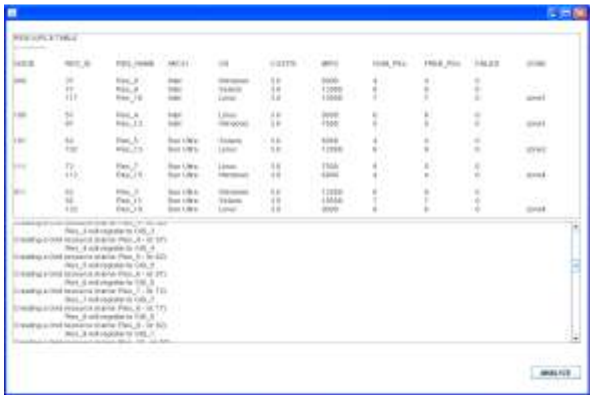

[illegible]

Figure 8. Gridsim output of Parameterized HPGRID Resource discovery Algorithm for a worst case 3D HPGRID

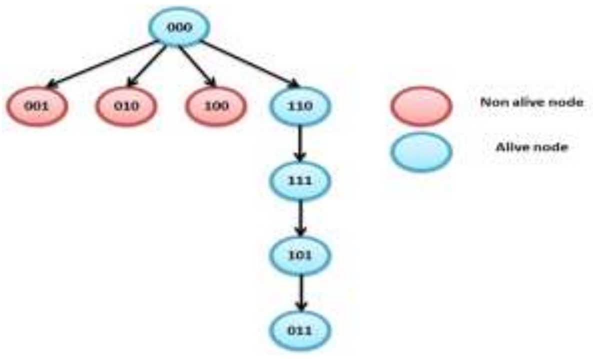


Figure 9. Parameterized HPGRID Resource discovery Algorithm for a worst case 3D HPGRID

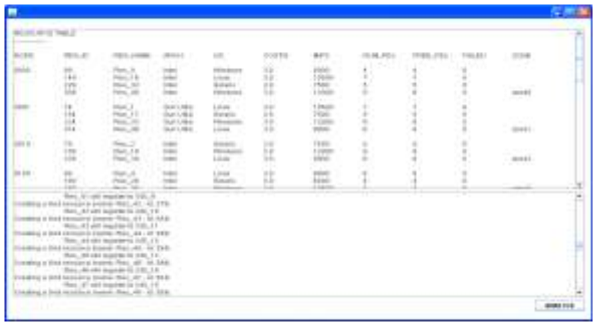

[illegible]

Figure 10. Part of Gridsim output of Parameterized HPGRID Resource discovery Algorithm for a best case 4D HPGRID

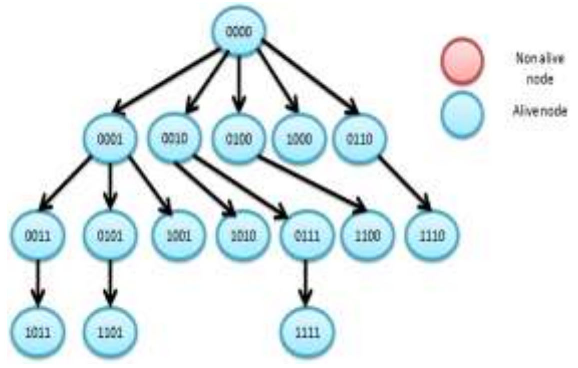


Figure 11. Parameterized HPGRID Resource discovery Algorithm for a best case 4D HPGRID

Figure 12 and 13 depicts that HPGRID algorithm outperforms the existing algorithm in comparison with the number of hops needed to complete the resource discovery process simulated using Gridsim [15] for both the 3D and 4D HPGRID systems in comparison with the HGRID system which does traversals in a normal hypercube.

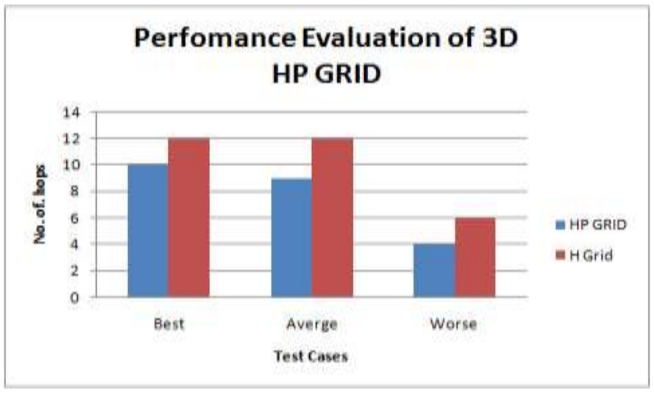


Figure 12. Performance Evaluation of 3D HPGRID

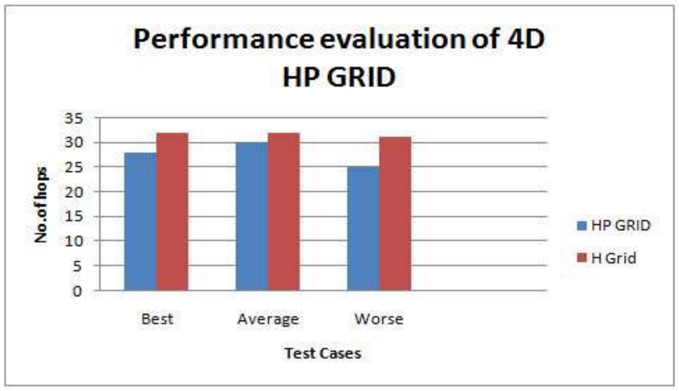
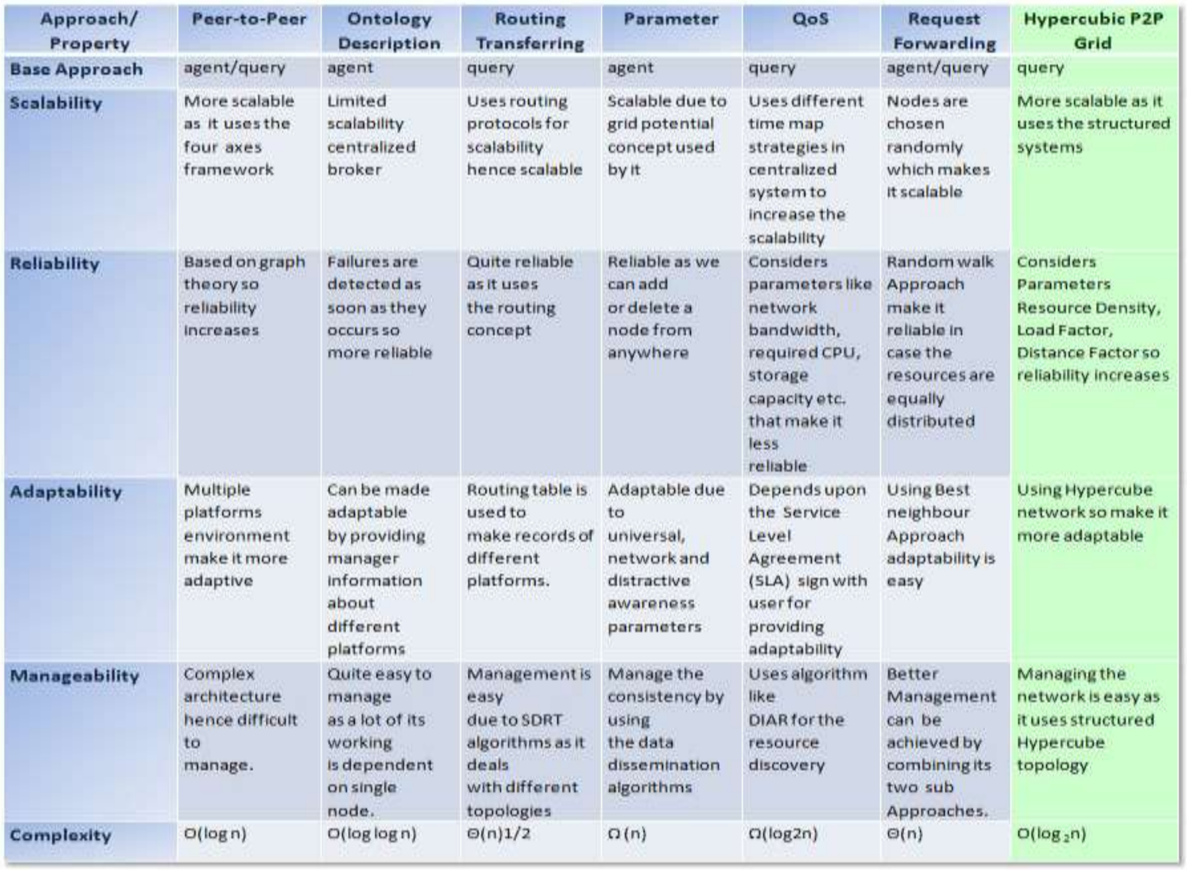


Figure 13. Performance Evaluation of 4D HPGRID

TABLE I. COMPARISON OF RESOURCE DISCOVERY ALGORITHM



Approach/ Property	Peer-to-Peer	Ontology Description	Routing Transferring	Parameter	QoS	Request Forwarding	Hypercubic P2P Grid
Base Approach	agent/query	agent	query	agent	query	agent/query	query
Scalability	More scalable as it uses the four axes framework	Limited scalability centralized broker	Uses routing protocols for scalability hence scalable	Scalable due to grid potential concept used by it	Uses different time map strategies in centralized system to increase the scalability	Nodes are chosen randomly which makes it scalable	More scalable as it uses the structured systems
Reliability	Based on graph theory so reliability increases	Failures are detected as soon as they occurs so more reliable	Quite reliable as it uses the routing concept	Reliable as we can add or delete a node from anywhere	Considers parameters like network bandwidth, required CPU, storage capacity etc. that make it less reliable	Random walk Approach make it reliable in case the resources are equally distributed	Considers Parameters Resource Density, Load Factor, Distance Factor so reliability increases
Adaptability	Multiple platforms environment make it more adaptive	Can be made adaptable by providing manager information about different platforms	Routing table is used to make records of different platforms.	Adaptable due to universal, network and distractive awareness parameters	Depends upon the Service Level Agreement (SLA) sign with user for providing adaptability	Using Best neighbour Approach adaptability is easy	Using Hypercube network so make it more adaptable
Manageability	Complex architecture hence difficult to manage.	Quite easy to manage as a lot of its working is dependent on single node.	Management is easy due to SDRT algorithms as it deals with different topologies	Manage the consistency by using the data dissemination algorithms	Uses algorithm like DIAR for the resource discovery	Better Management can be achieved by combining its two sub Approaches.	Managing the network is easy as it uses structured Hypercube topology
Complexity	O(log n)	O(log log n)	Θ(n)1/2	Ω(n)	Ω(log2n)	Θ(n)	$O(\log_2 n)$

The hypercubic P2P grid approach for resource discovery has been compared with the existing approaches discussed in [10]. The following table gives the comparison study of resource discovery algorithm described in Table I.

VI. Conclusions and Future Work

Our resource discovery scheme in HPGRID system uses Parameterized HPGRID algorithm which reaches all the alive nodes with minimum number of hops. The proposed algorithm is scalable in terms of time, because it keeps the maximum number of time steps required to resolve a resource request, to a logarithmic scale with respect to the total number of nodes. Moreover, each node has knowledge of the overlay CGP using the parameters defined. Therefore, our approach is also scalable and reaches all the alive nodes even in the lesser dimension of its search. Furthermore, scalability is also maintained by querying each node only once at the most (if possible). This important property (scalability) also extends to the number of nodes in the CGP. By using the deep multidimensional interconnection of a hypercube with additional neighborhood links, we provide enough connectivity so that resource requests can always be propagated in spite of non alive nodes. This makes our proposed algorithm much more fault-tolerant when it is compared with other topologies such as centralized, hierarchical or trees. In the absence of non alive nodes, it is able to offer lookup guarantees. Using isomorphic partitioning scheme if the resource needed not in the start node zones, then the number of resources and the number of tasks under examination are reduced by a single hop, thereby reducing resource discovery time The future work could be integrating of other resource management issues in this topology which could be extended to generalized topology like k ary n-cube systems. It could be extended considering the scheduling, security, QoS issues and also design and maintenance of new protocols in HPGRID

References

[1] S. Basu, S. Banerjee, P. Sharma, and S.J. Lee. NodeWiz: Peer-to-peer resource discovery for grids. In Proceeding of Cluster Computing and the Grid (CCGrid), 2005.

[2] M. Cai, M. Frank, J. Chen, and P. Szekely. MAAN: A mulit-attribute addressable network for grid information services. Journal of Grid Computing, 2(1):3–14, 2004.

[3] D. Doreen Hephzibah Miriam, T. Srinivasan, and R. Deepa. An efficient SRA based isomorphic task allocation scheme for k-ary ncube massively parallel processors. In Proceedings of the International Symposium on Parallel Computing in Electrical Engineering (PARELEC' 06), 2006.

[4] I Foster and Adriana Iamnitchi. On death, taxes, and the convergence of peer-to-peer and grid computing. In 2nd International Workshop on Peer-to-Peer Systems (IPTPS03), pages 118–128, 2003.

[5] A. Gallardo, L. Daz de Cerio, and K. Sanjeevan. HGRID: A hypercube based grid resource discovery. In Proceeding of the 2nd International Conference on Complex, Intelligent and software Intensive Systems (CISIS), pages 411–416, 2008.

[6] F. Heine, M. Hovestadt, and O. Kao. Ontology-driven p2p grid resource discovery. In Proceedings of the Fifth IEEE/ACM International Workshop on Grid Computing (GRID'04), pages 76–83, 2004.

[7] A. Iamnitchi and Ian Foster. On fully decentralized resource discovery in grid environments. In Proceedings of the Second International Workshop on Grid Computing, pages 51–62. Springer-verlage, 2001.

[8] D. Oppenheimer, J. Albrecht, D. Patterson, and A. A. Vahdat. Design and implementation tradeoffs for wide-area resource discovery. In Proceeding of the International Symposium of High Performance Distributed Computing(HPDC), 2005.

[9] M. T. Schlosser, M. Sintek, S. Decker, and W. Nejdl. HyperCuP - hypercubes, ontologies, and efficient search on peer-to-peer networks. Lecture Notes in Computer Science, 2530:112–124, 2002.

[10] A. Sharma and S. Bawa. Comparative analysis of resource discovery approaches in grid computing. Journal of Computers, 3(5):60–64, May 2008.

[11] P. Trunfio, D. Talia, C. Papadakis, P. Fragopoulou, M. Mordacchini, M. Pennanen, K. Popov, V. Vlassov, and S. Haridi. Peer-to-peer resource discovery in grids: Models and systems. Future Generation Computer Systems, 23(7), 2007.

[12] P. Trunfio and Domenico Talia. Toward a synergy between p2p and grids. IEEE Internet Computing, 7(4):94–96, 2003.

[13] Z. Xiong, Xuemin Zhang, and Jianxin Chen. Research on grid resource discovery scheme integrated p2p mode. In International Symposium on Electronic Commerce and Security, pages 105–109, 2008.

[14] Rajkumar Buyya and Manzur Murshed. Gridsim: A toolkit for the modeling and simulation of distributed resource management and scheduling for grid computing. CONCURRENCY AND COMPUTATION: PRACTICE AND EXPERIENCE (CCPE), 14(13):1175–1220, 2002.

[15] Agustin Caminero, Anthony Sulistio, Blanca Caminero, Carmen Carrion, and Rajkumar Buyya. Extending gridsim with an architecture for failure detection. Parallel and Distributed Systems, International Conference on, 1:1–8, 2007.

[16] Manfred Hauswirth and Roman Schmidt. R.: An overlay network for resource discovery in grids. In In: 2nd International Workshop on Grid and Peer-to-Peer Computing Impacts on Large Scale Heterogeneous Distributed Database Systems, 2005.

[17] A. Iamnitchi and Ian Foster. On death, taxes, and the convergence of peer-to-peer and grid computing. Lecture Notes in Computer Science, 2735:118–128, 2003.

[18] Adriana Iamnitchi, Ian Foster, and Daniel C. Nurmi. A peer-to-peer approach to resource discovery in grid environments. In In High Performance Distributed Computing. IEEE, 2002.

[19] Moreno Marzolla, Matteo Mordacchini, and Salvatore Orlando. Peer-to-peer systems for discovering resources in a dynamic grid. Parallel Computing, 33(4-5):339–358, 2007.

[20] Carlo Mastroianni, Domenico Talia, and Oreste Verta. A super-peer model for resource discovery services in large-scale grids. Future Generation Computer Systems, 21(8):1235–1248, 2005.

[21] Koen Vanthournout, Geert Deconinck, and Ronnie Belmans. A taxonomy for resource discovery. Personal Ubiquitous Computing, 9(2):81–89, 2005.

[22] ZHENG Xiu-Ying, CHANG Gui-Ran, TIAN Cui-Hua and LI Zhen. A Model for Resource Discovery Among Grids. The 10th IEEE International Conference on High Performance Computing and Communications, 678-682, 2008.

[23] Rozlina Mohamed and Siti Zanariah Satari. Resource Discovery Mechanisms for Peer-to-peer Systems. Second International Conference on Computer and Electrical Engineering, 100-104, 2009

[24] Honggang Xia and Hongwei Zhao A Novel Resource Discovery Mechanism in Grid. International Conference on Future BioMedical Information Engineering, 493-495, 2009.

[25] Abdelkader Hameurlain, Deniz Cokuslu, Kayhan Erciyes. Resource discovery in grid systems: a survey. International Journal of Metadata, Semantics and Ontologies , 5(3): 251-263,2010

AUTHORS PROFILE

D. Doreen Hephzibah Miriam is currently a Research Scholar at the Department of Computer Science and Engineering at Anna University, Chennai. She received her B.Tech in Information Technology from Madras University, Chennai, and M.E degree in Computer Science and Engineering from Anna University, Chennai. Her research interests include parallel and distributed computing, peer to peer computing and grid computing.

K. S. Easwarakumar is a Professor & Head at the Department of Computer Science and Engineering at Anna University, Chennai. He received his M.Tech in Computer and Information Sciences from Cochin University of Science and Technology, Cochin and Ph.D in Computer Science and Engineering from Indian Institute of Technology, Madras. His research interests include parallel and distributed computing, Data Structures and Algorithms, Graph Algorithms, Parallel Algorithms, Computational Geometry, Theoretical Computer Science and Molecular computing.