GRID Resource Searching on the GridSim Simulator

Antonia Gallardo[1], Luis Díaz de Cerio[2], Roque Messeguer[1], Andreu Pere Isern-Deyà[3], and Kana Sanjeevan[1]

[1] Departament de Arquitectura de Computadors, Universitat Politécnica de Catalunya, Avda. del Canal Olímpic s/n, 08860 Castelldefels, Barcelona, Spain
[2] Departamento de Automática y Computación, Universidad Pública de Navarra, Campus Arrosadía s/n, 31006 Pamplona, Spain
[3] Departament de Matemàtiques i Informàtica, Universitat de les Illes Balears, Ctra. de Valldemossa, km. 7.5, 07122, Palma de Mallorca, Illes Balears, Spain
{agallard,meseguer,sanji}@ac.upc.edu, luismanuel.diazdecerio@unavarra.es, reupere.isern@uib.es

Abstract. Nowadays, the Grid is the focus of multiple researches. Our work is centered on Resource Management for Grids as it is an opened and current research area. Decentralized, scalable and efficient resource search algorithms are one of the key issues for resource management in large Grid systems. Resource search is required in order to allocate applications and data efficiently and to maintain the quality of service at runtime, just to mention some examples. In this work, we propose a scheme that presents essential characteristics for self-configuring search and is able to handle dynamic resources, such as memory capacity. Our approach consists on a hypercube topology connecting the nodes and a scalable and self-configuring search procedure. The algorithm improves the probability of reaching the alive nodes in the Grid even in the presence of non-alive ones (inaccessible, crashed or heavy loaded nodes). In this paper, after the theory's description, we present some results obtained by running our search protocol on the GridSim simulator. We have evaluated 6 different metrics performing several resources searches and we show the arithmetic media for each measure.

Keywords: GridSim, Hypercube, Self-Configuring, Search Algorithms.

1 Introduction

The Grid includes a large number of dynamic and heterogeneous resources that are geographically distributed. Its main objective is to use the available resources provided by administrative domains, also named Virtual Organizations (VO) [1]. Grid Resource Management in general and Resource Search in particular is an opened research topic. This challenge plays a fundamental role, for example, allowing the system to allocate grid-enabled applications and data efficiently and to maintain the quality of service of the applications at runtime. In most of the current grids Resource Management solutions are based on centralized or hierarchical structures that are not appropriate for large systems because they are not scalable enough. By the other hand,

G. Allen et al. (Eds.): ICCS 2009, Part I, LNCS 5544, pp. 357–366, 2009.

several scalable and decentralized Peer-to-Peer (P2P) resource searching algorithms have been proposed for Grid systems. Nevertheless, P2P searching techniques were designed for non-dynamic content as files, and the Grid requires addressing dynamic resource data such as available memory, processor load, etc., for which, the P2P search is not suitable. Besides, as pointed out by Ian Foster and al. [2], "it is necessary to address failure, using scalable self-configuring protocols so to have a worldwide computer within which access to resources can be negotiated as and when needed". Motivated by this open research area, we present a scalable and decentralized architecture that allows the search of distributed resources preserving the VO's autonomy. The architecture is based on an overlay network with hypercube topology that interconnects nodes and each node represents a VO. We also present a self-configuring resource search algorithm that is able to adapt himself to environments where some nodes might be non-alive (crashed, inaccessible, high-loaded,…) when a resource is queried. Finally, we present some results obtained by running our search protocol on the GridSim simulator. We have evaluated 6 different metrics performing several resources searches and we show the arithmetic media for each measure.

One of the challenges of Grid Resource Searching is to be resilient in the presence of node failures. This resilience has different aspects: *static resilience* and *routing recovery*. As the present work is focused on the resource discovery algorithm this paper only addresses *static resilience* [3], that is, how well our approach can locate required resources before routing tables are updated by the routing recovery algorithm in order to remove non-alive nodes from the overlay. The other issue, *routing recovery*, deals with the fact that when failures occur, the routing tables are depleted in the remaining nodes. Routing recovery is not addressed in this paper, as this issue is related to the building and maintaining of the overlay topology.

The rest of the paper is organized as follows: In Section 2 we present an overview of our network architecture. Section 3 describes our resource search procedure. An overview of related work is presented in Section 4. In Section 5, the performance of the algorithm is shown. Conclusions and our future work can be found in Section 6.

2 The Hypercube Overlay Architecture

In actual Grid environments, the administrative domain (VOs), do not join or leave the Grid continually but occasionally. Most of the current VOs that form a Grid have powerful servers within their high performance local area networks and maybe they are interconnected by very high speed core networks. The servers seldom fail and they join and leave the system infrequently. Therefore the Grid Resource System can be organized in a stable and regular topology with low-diameter configurations, efficient searching and routing algorithms that address node failures. Each VO belonging to our environment, named HGrid, provides available resources and makes them accessible through software entities named Grid Information System (GIS) [4].

In HGrid, the interconnections between nodes have the topology of a hypercube. An n-dimensional hypercube (H_n) has $N = 2^n$ nodes, where each node represents a GIS and they have an identifier from 0 to 2^n-1. Two nodes are neighbors in the m-th dimension if the binary representations of their identifiers differ exactly by the m-th bit. Then, in a complete hypercube H_n, each node has exactly n neighbors. Fig. 1

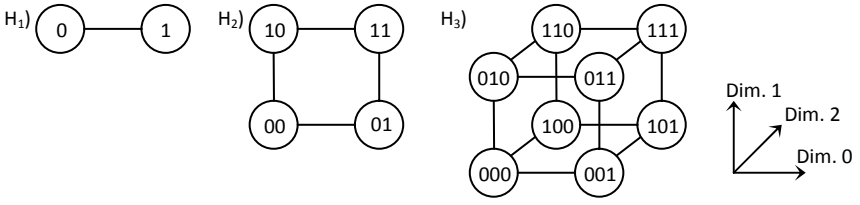

Fig. 1. H_1) The architecture for the interaction among 2 nodes, a one-dimensional hypercube, H_2) 4 nodes, a two-dimensional hypercube and H_3) 8 nodes, three-dimensional hypercube

illustrates the architecture for 2, 4 and 8 nodes respectively. An overlay network with a hypercube topology connecting each GIS in a grid environment allows each VO to contribute their resources while assuring their autonomous management. The resources offered by a VO can join or leave the system at any time updating its own GIS. Every GIS keeps a small routing table of only n entries ($n = \log_2 N$) corresponding to their n neighbors. Also, each GIS has the responsibility to verify which of its neighbors are still alive. The term alive is applied to a neighbor node that can be reached across the network. Then, a node that has crashed, is inaccessible or is heavily loaded by traffic is considered a non-alive node.

3 Resource Search Using HGRID

We present a scalable self-configuring resource search algorithm that is able to adapt him automatically to dynamic environments. Inside HGrid, the approach named Algorithm-H, is performed to search a resource/s requested by a client. It is a self-configuring protocol by adapting itself when some nodes are in a non-alive state (inaccessible, crashed or heavy loaded nodes). As we said before, we do not address the building and maintaining of the hypercube topology when a node joins or leaves the overlay. Changes in the overlay network make the routing tables be re-mapped and this does result in some overhead. However, some of the results published regarding the scalability of hypercube overlays used in other environments seem to address this challenge and offer an adequate solution [5].

We assume that in Grid environments the VOs joins or leaves the Grid occasionally, so the overhead of building and maintaining the hypercube overlay is smaller than in an extremely transient environment - where a significant fraction of the nodes are joining or leaving at any time. Since an incomplete hypercube could be re-built as a complete hypercube where the void spaces are completed by replicating some of the GIS, we have considered the number (N) of GIS is always a power of 2, so all the nodes have n-neighbors - where n is the dimensionality of the hypercube. Finally, in our proposal it is possible to initiate a search request from any of the GIS nodes alive and to propagate the request to the rest of nodes. But for generality reasons, all the examples used from now on assume that GIS 0 is the start node.

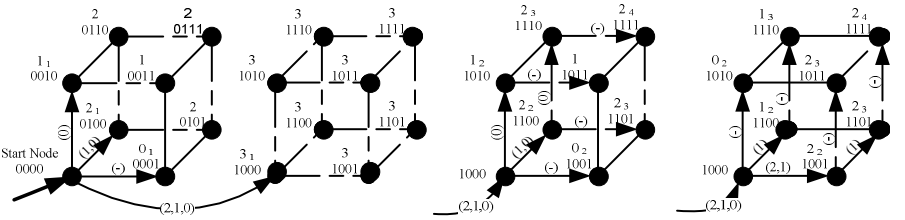

Fig. 2. The Search Procedure in a H_4 hypercube. a) Decomposing a hypercube H_4 into four hypercubes. b) Searching in the 3-dimensional hypercube marked with 3 in a). Each reached node applies the same procedure based on the received vector. c) A possible reorganization of the case showed in b). Before sending the query, every reached node can reorganize its hypercube of several forms.

3.1 The Search Procedure in a H_4

The search starts when a client connects to a GIS node and it requests a resource/s. If the start node does not have the resource/s requested, then it starts a search. Next, we show the search in a 4-dimensional hypercube:

1) The start node 0, in decimal notation, of a 4-dimensional hypercube is connected across its 4 neighbor nodes to 4 different hypercubes. These 4 hypercubes are a hypercube of 2^0 nodes in dimension 0, one of 2^1 nodes in dimension 1, one of 2^2 nodes in dimension 2 and finally, one of 2^3 nodes in dimension 3. In Fig. 2.a) we have marked these hypercubes with 0, 1, 2 and 3, respectively. Notice that these 4 hypercubes contain all the nodes of the overlay except the initial node.
2) When the start node does not have the requested resource/s, it starts a search by sending the query to each of its neighbors. It is the 1st step of 4. In this step, the nodes reached are marked by the sub-index 1 in Fig. 2.a) as 0_1, 1_1, 2_1 and 3_1.
3) Each reached node receives a first vector along with the request. This vector indicates the dimensions through which the request will be send in case of it cannot be satisfied by the reached node.
4) Each reached node applies the same procedure described in previous steps - 1), 2) and 3) - for its neighbours formed by the received dimensions. We show in Fig. 2.b) the 3-dimensional hypercube marked with 3 in Fig. 2.a). Each GIS node has an identifier in H_4, the decomposed hypercube dimension of which it forms a part, the vector received and the search algorithm step when it is requested. In Fig. 2.b), the node 1010 forms a part of *a* 1-dimensional decomposed hypercube, the node receives the vector 0 and it is reached in the step 2 of the search procedure.
5) Before sending the query, each reached node can reorganize its hypercube of several forms. We show in Fig. 2.c) a possible reorganization of the case illustrated in Fig. 2.b) for the 3-dimensional hypercube marked with 3 in Fig. 2.a). In Fig. 2.c), the node 1010 forms a part of a 0-dimensional decomposed hypercube, the node receives an empty vector (marked by (-)) and it is reached in the step *2* of the search algorithm. When a node receives an empty vector, it does not propagate this vector anymore. Notice that the nodes covered in 2c) are the same nodes those reached in 2.b), nevertheless, the node 1010 propagates the

query to nobody in Fig. 2.c) and propagates to one neighbour (the node 1011) in Fig. 2.b).

6) What does happen if node 1011 has the required resource(s) and node 1010 is a non-alive node? In this case, if node 1000 knows that node 1010 is not alive (maybe it is crashed), it reorganizes its hypercube as in Fig. 2.c) and the node 1011 is not reached across the non-alive node by trying to be queried to another.
7) The search procedure reorders the vector received with the resource query in order to send the query at maximum number of nodes that was possible.
8) Each reached node receives a second vector, too. The search procedure tries to requested GIS nodes through nodes in an alive state, by avoiding the non-alive ones. This second vector is not illustrates in Fig. 2, but it is shown in Fig. 3.

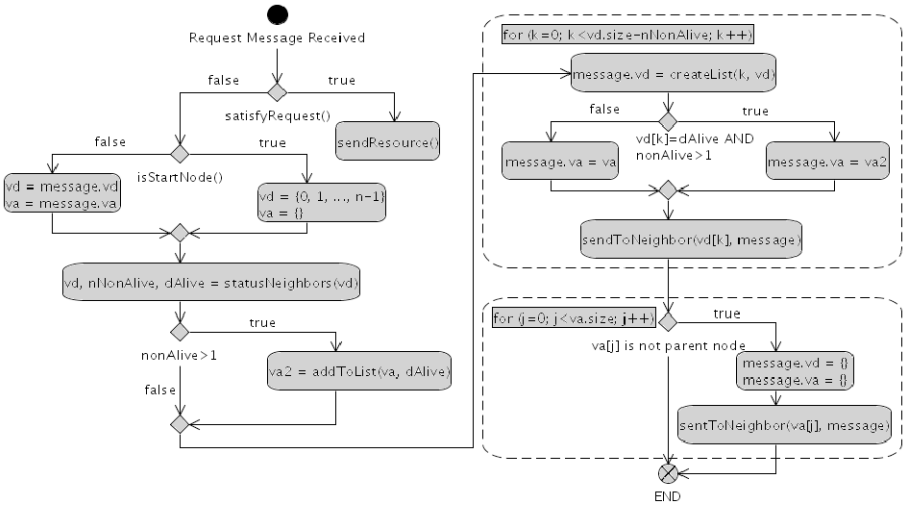

Fig. 3. Flow Diagram of the Algorithm-H

Propagating the requests in this way, the effect of non-alive nodes is reduced. Making the arrangement in the vector received, non-alive nodes would propagate the request to fewer neighbors than alive ones. 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. For the start node and for each node that receives a non-empty vector, if only one neighbor node of those to which the search must be propagated is in a non-alive state, the total number of nodes reached at the algorithm last step is not affected. Related to the alive nodes unreachable due to the fact its non-alive parent's node, Algorithm-H tries to reach them using the v_a list. In Fig.3, the flow diagram is illustrated.

3.2 A Complete Example Using Algorithm-H

Fig. 4 illustrates a complete example. We transform the hypercube representation to that of a *tree-like* structure in order to illustrate better our search procedure (notice, some *child* nodes could appear more than once during subsequent time steps).

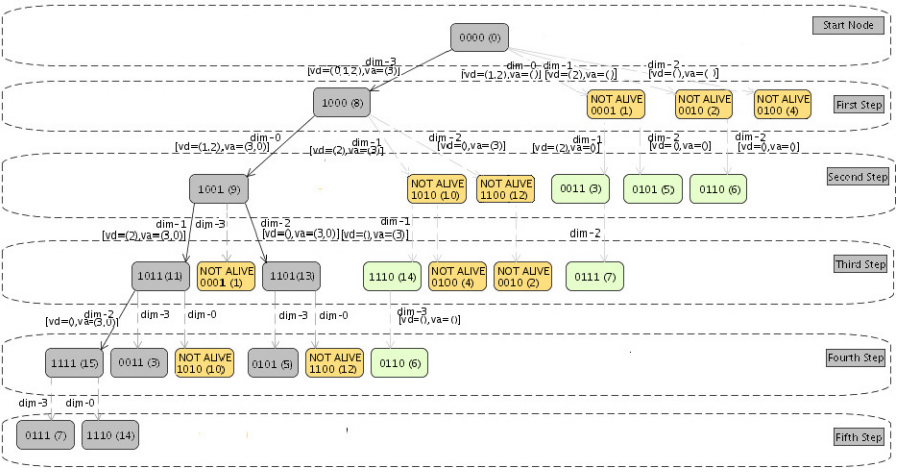

Fig. 4. Algorithm-H: A request of a resource/s P started at node 0000 in a 4-dimensional hypercube (A complete example)

A request for service P starts at node 0000 in a four-dimensional hypercube. We assume that none of the nodes has the service requested (note that this is the worst case). In the example, the value of the list vd at the start node is {0, 1, 2, 3} and the ordering after calling the statusNeighbors() function is {3, 0, 1, 2}. In this case 0, 1 and 2 are located at the last three positions of vd = {3, 0, 1, 2} because we assume that neighbors in dimensions 0 (0001), 1 (0010), 2 (0100) and are non-alive nodes. The neighbor in dimension 3 (1000) is the last alive node - the only one in this case.

In the first step, the start node's neighbor in dimension 3 (1000) receives the service request *P*, the list $v_d = \{0, 1, 2\}$ and $v_a = \{3\}$ since it is the last alive neighbor. The algorithm tries to reach nodes 0010, 0101 and 0110 (whose parent nodes are non-alive) by sending the list $v_a = \{3\}$ to node 0010 to be used in the third stage.

In the second step, looking at node 1001, the message composed of the resource request *P* along with the lists $v_d = \{1, 2\}$ and $v_a = \{3\}$ is received. If the node is unable to satisfy the request - *processRequest()* returns *false* -, v_d is not sorted because its neighbor in dimension 1 (1011) and its neighbor in dimension 2 (1101) are alive nodes. Although $v_a = \{0\}$ is received by the node 1001, it does not propagate the message to its neighbor in dimension 0 (1000) because it is its parent node.

In the third phase, looking at node 1011, the message composed of the resource request *P* along with the lists $v_d = \{2\}$ and $v_a = \{3,0\}$ is received. If *processRequest()* returns *false*, its neighbor in dimension 2 (1111) receives *P*, $v_d = \{\}$ and $v_a = \{3,0\}$ and nodes (0111) and (1110) receive *P*, $v_d = \{\}$ and $v_a = \{\}$, due to the list $v_a = \{3,0\}$.

In the fourth stage, the resource request *P* is received from node 1011 (0011 is the neighbor in dimension $v_a[0] = 3$) to the node 0011. Notice that 0011 was not reached in the 2nd step due to the non-alive node 0001 but it is reached now.

In five steps almost all of the alive nodes inside the four-dimensional hypercube are visited (except node 0110) even when three neighbors of the start node (0001, 0010 and 0100) and two more nodes (1010 and 1100) are presumed to be non-alive.

4 Related Work

The Globus Toolkit's Monitoring and Discovery System (MDS) defines and implements mechanisms for resource discovery and monitoring in Grid environments [6]. Motivated by these issues, recently there have been several studies using the P2P model to build a decentralized architecture of VOs. Most of them adopt Distributed Hash Tables (DHTs) and a few of them introduce unstructured P2P topologies. All these studies indicate that some P2P models could help to overcome the challenges posed by the dynamic environment in Grids. Adriana Iamnitchi and Ian Foster [7] suggest a decentralized architecture similar to the Gnutella P2P system. This approach is not able to guarantee that some required information that exists in the system can be found even when the system has no failures. Moreover, a peer could be often reached several times by the same query. Our approach assures that nodes are reached only once. In the absence of failures all nodes in the system are reached and if some failures occur the most nodes are reached.

DHT based systems handle unexpected node failure through redundancy in the network and some of them also do node asynchronous lookups periodically to compensate for disappeared nodes – for example, Kadmelia [8]. To enable efficient searches a DHT needs to have the data-item distributed. Our approach does not require distributing the data-items but each request sends from 0 to N-1 messages.

Keeping the state of highly dynamic data-items updated - such as available memory or CPU processing - requires sending a large amount of messages in DHTs approaches– as in SWORD [9]. In DHTs, the data-item that belong to a VO, are geographically distributed and, an updating operation reaches O(log N) of distributed peers before inserting the data item (where N is the total number of peers). Our approach takes advantage of the high quality network available inside a VO (probably a LAN): updates are more efficient inside a VO than among distributed VOs. Furthermore, HGrid is more efficient in the insert phase as it is performed locally.

A node in the traditional DHT has no control over the distribution of its data items, and the number of data items belonging to others that it has to store. DGRID [10], a model for supporting GIS over the DHT Chord, maintains the resource information in the originating VO by increasing the total number of DHT nodes. Unlike traditional DHTs, DGRID is by design resilient to node failures without the need to replicate data items. The meaning of resilient to node failures is defined as the ability to locate existing resources whose originating VO is still alive. The approach presented in this paper has the same resiliency to node failures as in DGRID and also guarantees that any data item can be located in O(log N) overlay hops.

In HGrid, changes in the overlay network when a grid node joins or leaves the system do not cause resource information (data-items) to be remapped, whereas in traditional DHTs, it causes both routing tables and data-items to be remapped.

Recently, an unstructured topology based on hypercube has been proposed for use on Data Grids [11]. The nodes in this work contain pointers to shared data. Data Grids need to improve locality among distributed data - which are stored as pointers in the overlay nodes. In order to improve the locality of data, the paper imposes a hypercube

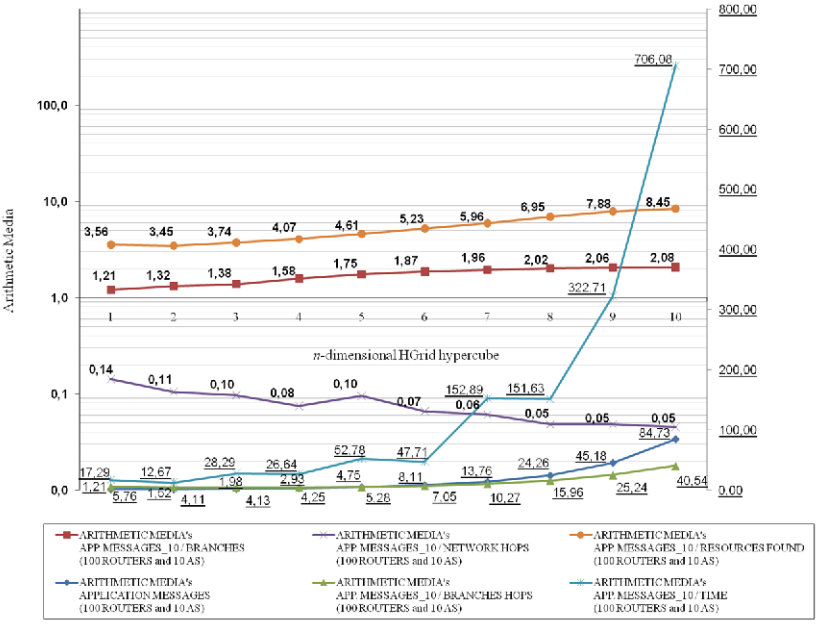

Fig. 5. HGrid and Algorithm-H: For each parameter, once set the (P_{GIS}, P_{REC}) values, we performed 20 resources searches and we obtained the arithmetic media

Characteristics	Computer-1	Computer-2
CPU	P4 2.6GHz	C2D 1.6GHz
RAM Memory	1.5 Gb	2 Gb
Hard Disk	1 x 120GB + 1 x 160GB	1 x 160GB

Fig. 6. The Computers used on our experiments with HGrid and Algorithm-H's searches

GIS's topology - named DGIS. After this, it proposes a transposition algorithm in order to optimize the overlay network's topology according to the access statistics between peers - that is, to improve the data locality. However, the algorithm showed does not address non-alive nodes and failures.

Finally, we have evaluated HGrid and Algorithm-H in previous works. For example, in [12], we show the percentage of average of failed paths across different search algorithms (HaoRen et al.'s algorithm - a non fault-tolerant algorithm described in [11] - and Algorithm-P - our previous Algorithm-H fault-tolerant protocol described in [13].

5 Performance Evaluation

In Fig.5, we tested the *static resilience* of Algorithm-H inside 1 to 10-dimensional HGrid overlays. The experiments run in 2 computers described in Fig.6. All HGrid's GIS had a P_{GIS} probability - probability of failure - and a probability P_{REC} - probability

to have the resource/s requested when the queried arrives to a GIS. P_{GIS} can be seen as the percentage of non-alive nodes that can be in and P_{REC} as the percentage of resources requested presents in the HGrid hypercube when a search is performed. The BRITE [14] and the GridSim [15] was used to create automatically the IP Internet topology and the HGrid, respectively. Given a (P_{GIS}, P_{REC}) we started 20 requests for service P at and we calculated the arithmetic media for each measure.

We have evaluated the following metrics for each resource/s search in HGrid using Algorithm-H: a) the total number of messages sent in the hypercube overlay, b) number of branches opened where a branch is each of the searching ways in which it is divided the requested message initiated by the client, c) for each opened branch, the number of application hops, d) for each branch established, the network hops performed through routers in the overlay, e) by simulation, the search estimated time, where the time was set as the difference between the time when the request resource arrives at the last GIS and the moment in which the user initiates his search and the number of the resources found during the search in HGrid. In Fig.5 we show the inter-relation between the metrics b) - e) and the total messages application sent - a).

To summarize, our results confirm that the static resiliency of the algorithm shown is very efficient for current non-extremely transient Grid environments. It offers high lookup guarantees and it seems to be scalable with the number of nodes.

6 Conclusions and Future Work

The present approach allows the search of geographically distributed resources while preserving the autonomy of each individual VO. Unlike traditional approaches based on DHTs our scheme is suitable for efficiently handling dynamic attributes such as memory capacity without generating overhead across distributed nodes. HGrid using Algorithm-H is able to adapt him automatically to environments where nodes could be heavily loaded or even crashed, without requiring any node to have the global state information. We refer to this property as self-configuring. In the absence of non-alive, if some node present in the overlay is able to satisfy the request this node will be found in few steps (less than the hypercube dimension). Therefore the proposed scheme offers lookup guarantees in the absence of faulty nodes. If non-alive nodes are present, the algorithm also offers lookup guarantees in some cases.

To conclude, there are interesting areas that have opened up as a result of this work as comparing several metrics between the present approach and some actual DHT-approaches, to evaluate if the router layer affects to the overall resource searches, new search algorithms - design and simulation -, the incorporation of topology maintenance protocols and performing studies related to other topologies such as ring or mesh. Finally, deploy the algorithms into a real GRID is a good form to end this work, basically, a desirable final step.

Acknowledgements

This work was supported by the Ministry of Science and Technology of Spain and the European Union (TIN2007-68050-C03-01 and TIN2007-68050-C03-02).

References

1. Nabrzyski, J., Schopf, J.M., Weglarz, J.: Grid Resource Management. State of the Art and future Trends. Kluwer Publishing, Academic publishers (2004)
2. Foster, I., Iamnitchi, A.: On Death, Taxes, and the Convergence of Peer-to-Peer and Grid Computing. In: Kaashoek, M.F., Stoica, I. (eds.) IPTPS 2003. LNCS, vol. 2735. Springer, Heidelberg (2003)
3. Gummadi, K., Gummadi, R., Gribble, S., Ratnasamy, S., Shenker, S., Stoica, I.: The Impact of DHT Routing Geometry on Resilience and Proximity. Applications, Technologies, Architectures and Protocols for Computer Communications, 381–394 (2003)
4. Buyya, R., Murshed, M.: GridSim: A Toolkit for the Modeling and Simulation of Distributed Resource Management and Scheduling for Grid Computing. Concurrency and Computation: Practice and Experience 14(13-15), 1175–1220 (2002)
5. Liebeherr, J., Beam, T.K.: HyperCast: A Protocol for Maintaining Multicast Group Members in a Logical Hypercube Topology. In: Rizzo, L., Fdida, S. (eds.) NGC 1999. LNCS, vol. 1736, pp. 72–89. Springer, Heidelberg (1999)
6. The Globus Toolkit, `http://www.globus.org/` (last access 15/02/2009)
7. Iamnitchi, A., Foster, I.: On fully decentralized resource discovery in grid environments. In: Lee, C.A. (ed.) Grid 2001. LNCS, vol. 2242, pp. 51–62. Springer, Heidelberg (2001)
8. Maymounkov, P., Mazières, D.: Kademlia: A Peer-to-Peer Information System Based on the XOR Metric. In: Druschel, P., Kaashoek, M.F., Rowstron, A. (eds.) IPTPS 2002. LNCS, vol. 2429, pp. 53–65. Springer, Heidelberg (2002)
9. Oppenheimer, D., Albrecht, J., Patterson, D., Vahdat, A.: Design and Implementation Tradeoffs for Wide-Area Resource Discovery. In: 14th IEEE Symposium on High Performance Distributed Computing, Research Triangle Park, NC USA (HPDC 2005) (2005)
10. March, V., Teo, Y.M., Wang, X.: DGRID A DHT-Based Resource Indexing and Discovery Scheme for Computational Grids. In: 5th Australasian Symposium on Grid Computing and e-Research (AusGrid 2007), pp. 41–48 (2007)
11. Ren, H., Wang, Z., Liu, Z.: A Hyper-cube based P2P Information Service for Data Grid. In: Conference on Grid and Cooperative Computing (GCC 2006), pp. 508–513 (2006)
12. Gallardo, A., Díaz de Cerio, L., Sanjeevan, K.: Scalable Self-Configuring Resource Discovery for Grids. In: V Brazilian Workshop on Grid Computing and Applications (WCGA 2007) (2007)
13. Gallardo, A., Díaz de Cerio, L., Sanjeevan, K., Bona, L.C.E.: HGRID: An Adaptive Grid Resource Discovery. In: 2nd International Conference on Complex, Intelligent and Software Intensive Systems (CISIS 2008) (2008)
14. Medina, A., Lakhina, A., Matta, I., Byers, J.: BRITE: Universal Topology Generator from a User's Perspective, pp. 1–47 (2001), `http://www.cs.bu.edu/brite/` (last access 15/02/2009)
15. Buyya, R., Ranjan, R., Broberg, J., Dias de Assuncao, M.: GridSim: A Grid Simulation Toolkit for Resource Modelling and Application Scheduling for Parallel and Distributed Computing, `http://www.gridbus.org/gridsim/` (last access 15/02/2009)