Parallel Resource Discovery in DHT-based Systems

H. Fang, W. Wu, X. Jiang, Y.M. Teo and X. Wang

Singapore-MIT Alliance

National University of Singapore

3 Science Drive 2, Singapore 117543

email: {fanghui}@nus.edu.sg

8 July, 2005

Abstract

Increasingly, large and dynamic computing resources are organized as P2P networks for grid computing. However, due to the dynamic nature of node joining and leaving, a node that initiates a grid computation task needs to discover sufficient available resources without keeping resource information on every node. In structured P2P networks such as DHT-based Chord, resource discovery can be performed cooperatively and in parallel. In this paper, we propose a parallel resource discovery approach for structured P2P networks that partitions the whole searching space into multiple non-overlapping intervals. In each interval, a node is assigned the discovery task for each interval. This approach has a near optimal message complexity of $O(n)$, which is much better than using individual lookups and without cooperation among nodes. Our simulation shows that good trade-off between network churn and latency can be obtained by controlling the degree of parallelism.

1 Introduction

The computational grid aims to provide a mechanism for sharing and accessing large and heterogeneous collections of remote resources such as CPU cycles, storage space, online instruments, data and even applications. Peer-to-peer (P2P) technology provides a good alternative to set up the infrastructure for grid computing. However, since the nodes in P2P network are dynamically joining or leaving, each one needs to find sufficient available resources in the system to execute computational jobs cooperatively. Therefore, resource discovery is regarded as a basic service in grid systems.

P2P system based on distributed hash table (DHT) such as Chord [1] can provide efficient and scalable lookup performance in a distributed environment. Decentralization and dynamism are the key characteristics of P2P system [4]. With computing nodes joining and leaving dynamically, the computational resources shared by a node vary over times depending on its own workload. Resources can be hardware such as processor cycles and storage space, program execution environment such as Java and MPI, software package such as Matlab, files and data-sets, etc. Resources are heterogeneous and owned by various organizations or individuals. These facts make it difficult to obtain and reuse a resource for a long time. Therefore, it is necessary to design an efficient resource discovery algorithm for computational grid computing.

This paper proposes a parallel resource discovery approach for DHT-based computational grid environment. The algorithm does not rely on resource information repository; it is efficient because it dispatches parallel resource discovery queries and finishes the discovery process in $O(\log n)$ rounds with $O(n)$ messages. It can find all nodes that meet the resource requirements, with high probability. The algorithm exploits a tree-like structure to gather resource information in order to deal with the network churn and improve fault tolerance. The depth of the tree can be predefined to control the number of discovering messages. Our algorithm can be implemented in Chord-based distributed computing environment. A node initiates a resource discovery query with description of the required resource. The algorithm lets all nodes be aware of the resource requirement and reply the initiating node if it matches the requirement.

The rest of this paper is organized as follows. Section 2 describes related works on resource discovery. Section 3 presents our approach to parallel resource discovery using Chord as an example. Section 4 discusses the algorithm analysis, and section 5 presents simulation results. Section 6 summarizes our results.

2 Related Works

The problem of resource discovery for unstructured networks is introduced by Harchol-Balter et al in [3]; they treat the resource discovery problem as designing distributed algorithms that evolve a connected directed graph into a complete graph. The nodes then discover other machines by flooding the messages or by randomly (or selectively) forwarding the message to neighbors.

In the *flooding* algorithm [3], in every round each node γ contacts all of its initial neighbors and transmits the updates of $\Gamma(\gamma)$, where $\Gamma(\gamma)$ is the set consisting of γ and all the nodes that γ points to. The number of rounds needed for the flooding algorithm is equal to the diameter of the initial graph. In [3] the authors also present a *Name-Dropper* algorithm in which each node picks a neighbor randomly and passes the neighbor all its known pointers during each round. Finally, all machines find all other machines with $O(n \log^2 n)$ messages in $O(n \log^2 n)$ rounds. For strongly-connected networks, a random-

ized resource discovery algorithm named *Absorption* message complexity of $O(n)$ messages and $O(\log^2 n)$ rounds is proposed in [5]. Besides deterministic resource discovery algorithm, some asynchronous resource discovery algorithms [11, 12] are proposed.

In the context of structured P2P networks such as Chord [1], each node only needs to maintain a small set of information about other nodes. To execute computing application cooperatively with other possible nodes, one node has to firstly find out the nodes that satisfy the resource requirements. Chord provides the lookup function to support the single key searching in $O(\log n)$ hops, where n is the total number of nodes in Chord. However, it will be better to find out all the satisfying resources as a whole than to get these resources by using single key lookup service multiple times. To achieve this, it is necessary for all the involved nodes to cooperate with each other in an efficient way. Epichord [13] parallelizes the Chord lookup function by issuing multiple queries asynchronously in parallel. Specifically, to lookup a destination id, a node in Epichord initiates p queries in parallel to the node immediately succeeding id and to the $p-1$ nodes preceding id. By removing the $O(\log n)$-state-per-node restriction, Epichord achieves $O(1)$-hop lookup under lookup-intensive workloads, and $O(\log n)$-hop lookup under churn-intensive workloads.

Our work focuses on improving the resource discovery performance with the following differences. The goal of most existing resource discovery algorithms is to make each machine be aware of all other machines, while the goal of our approach is to let the initiating machine know all nodes or sufficient number of nodes that satisfy the search criteria. Most existing lookup algorithms for structured P2P system, including Epichord, only take care of the individual key lookup without fully exploiting the network structure property of DHT.

The work presented in [4] is similar to ours in that they intend to discover resources matching the specified requirement in decentralized Grid environments where there is no resource information repository. They proposed a set of P2P-style resource discovery algorithms with four alternative request forwarding strategies. T.N. Ellahi and M.T. Kechadi [6] also proposed a P2P resource discovery mechanism that equips the users with the capability to customize their neighborhood according to their preferences and performance requirements. The difference between these work and ours is that their mechanisms are for unstructured P2P environment while ours is designed for structured P2P networks; our algorithm makes use of the network structure and is more efficient, and the simulation shows that the proposed algorithm provides a good trade-off with varying network churn.

The other major category of related works is the resource discovery mechanisms that rely on resource information indexing. Sriram Ramabhadran et al. [7] proposed the Prefix Hash Tree which is an indexing data structure over DHTs and the aim is to support range queries. In [8], Cristina and Manish use the Hillbert SFC (Space Filling Curves) to map multidimensional information space into a 1-dimensional index space, and then map the 1-dimensional index to physical peers in a DHT based overlay network.

Magdalena Balazinska et al. [10] maps resources to resolvers by transforming the descriptions into numeric keys, and resolvers are organized by DHT overlay. DGRID visualizes an index server into a number of nodes subjected to the number of resource types registered on it and then arranges the nodes as a DHT [9]. What distinguish our resource discovery mechanism from these approaches is we do not to need maintain indexing and mapping of resources, the DHT structure is only used to connect computers in P2P systems.

3 Parallel Resource Discovery

Before exploiting the powerful capability of grid computing on P2P platform, the initiating node must gather enough available resources through resource discovery.

The main idea in parallel resource discovery is to divide the identifier space of P2P network into multiple intervals, and a node is assigned within an interval to discover resource. The designated node then returns the resource information to the initiating node. The procedure is intrinsically recursive. The discovery terminates when the size of interval becomes zero, or the discovering depth reaches a predefined maximum.

We present and compare two algorithms that exploit the structure of DHT-based system. The first is a flooding algorithm in which each node simply forwards the resource query to the selected nodes that are in its finger table. The second algorithm divides Chord into multiple non-overlapped intervals and the initiating node forwards the resource query to one node for each interval, and the destination node only needs to search in this interval. For the second algorithm, the procedure is recursively executed until the size of the interval reaches zero or the searching reaches the maximum depth.

We assume that each node can understand the resource query which describes the initiating node's requirement and can determine whether it matches the requirement. That is to say, when a node receives a resource query, it can understand the semantic of the query and can give a local search and judge whether itself is a node that matches the resource requirement.

3.1 Flooding Approach

A straightforward algorithm of parallel resource discovery in DHT-based systems can work like this: when a node receives a resource query for the first time, it firstly forwards the query to the nodes it knows, then it judges whether itself is a node matching the specified requirement. If so, it sends a message to the initiating nodes to inform the result; But when it receives the same query afterwards, it simply ignores it.

The idea behind this algorithm is that in Chord each node maintains a finger table of size $\log n$, where

n is the total number of nodes in Chord. If each node forwards the resource query to the nodes in its finger table, it is with high probability that all nodes in Chord will finally receive the resource query. Thus the initiating node can discover all nodes that match its requirement.

For example, in a Chord of 16 nodes, in node 0's finger table there are the fingers to nodes 1, 2, 4 and 8, when node 0 has a task and needs to find appropriate nodes to run the task, it issues parallel resource queries to node 1, 2, 4 and 8, in the next round, node 1, 2, 4, 8 all forward the query to the nodes in their finger tables: node 1 forwards to node 2, 3, 5, 9; node 2 forwards to 3, 5, 6, 10; node 4 forwards to 5, 6, 8, 12; node 8 forwards to 9, 10, 12, 0. In this round, some nodes receive the query the first time and go on the forwarding, some nodes receive the query the second time and ignore it. Figure 1 shows the running rounds and message passing sequences of this algorithm, note that figure 1 shows only the messages of round 1 and 2.

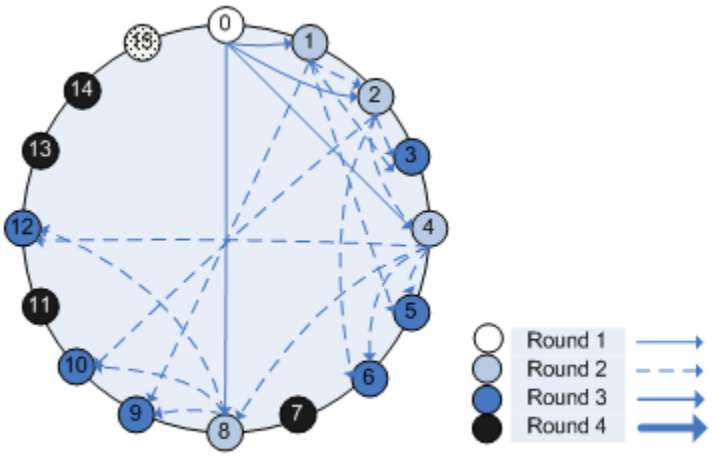

Figure 1: Flooding Resource Discovery Approach

3.2 Parallel Resource Discovery Approach

In the flooding algorithm, many duplicate messages are generated. We present an algorithm to eliminate duplicate messages by dividing the whole discovery space into many smaller spaces that can be searched independently and in parallel.

In the algorithm, a node that initiates the resource discovery sends query messages to the selected neighbors. The resource discovery message therefore is multicast recursively to all the nodes which may own the needed resources. Upon receiving the query message, each node forwards the query to other neighbors within its discovery area, and evaluates the query and sends a reply message to the initiating node if a match is successful. To remove duplicate messages, each node obtains only an interval of Chord identifier space to forward.

Assume the parent node x who owns L-length spacing will divide its spacing into p intervals and distribute them among its p active fingers, and the length of i-th interval is S_i. Therefore a non-overlapping division for the discovery area of node x needs to satisfy: $\sum_{i=1}^{p} S_i = L$. Finally the spacing L of an interval will be decreased to 0 which means that node doesn't need to forward the query any more.

Hence the algorithm of our parallel resource discovery includes mainly two parts: one is spacing partition; another one is parallel querying. The Figure 2 shows the parallel resource discovery algorithm. Each node processes the incoming request independently.

procedure x.**parallelRD**
Parameter: parallelism p, and maximum searching depth D
Input: resource discovery request RD
$L = RD.spacing$;
$initiator = RD.initiatingNode$;
$depth = RD.depth$;
$depth$++;

if ($depth < D$)
  A = the subset of x's finger list, sorted and within spacing L;
  partition A into (at most) p intervals. Assume the i-th interval has size S_i;
  for $i = 1$ to p **in parallel**
    f_i = the first finger lying within the i-th interval;
    send resource discovery request to f_i with spacing S_i;
  end for
end if

res = searching result on local resources;
Output: report res to initiator.

Figure 2: Proposed Parallel Resource Discovery Algorithm

In the parallel algorithm, different partition strategy can be chosen to be adaptive with parallel querying. For example, the p destination nodes could be the first p fingers in a row. In this case, generally the child nodes closer to parent will obtain a relatively narrow spacing, and the child nodes far from parent will obtain a relatively wider spacing. This is because the distance between two neighboring fingers of a node generally increases as the finger index increases, reminding that the nodes in finger table have a distance of $2^i (0 \leq i \leq \log n)$ from the node. Another example is that all the interval sizes could be equal. Therefore the workload for each discovering node is relatively same. As seen from the algorithm, no matter which strategy is used, one advantage of the partition on discovering area before parallel

querying is that the following p queries will not interfere with each other, since the intervals are non-overlapped and the discovering node is the first node in its interval. The local search result will be reported to the initiating node directly.

Figure 3 is a showcase for parallel resource discovery. In a Chord with 16 nodes present, if node 0 initiates a resource discovery with parallelism $p = 4$ and maximum depth $D = 4$, it will forward this information to its four fingers 1, 2, 4, 8. Then node 4 obtains an interval [4, 8) and will forward again to node 5 and node 6. ... Node 7 can not forward request any more because its assigned interval reaches zero. The whole process is specified by four rounds:

$$
\begin{aligned}
&\textit{round}\ \ 1: \quad 0 -> \{1, 2, 4, 8\} \\
&\textit{round}\ \ 2: \quad 2 -> \{3\}, 4 -> \{5, 6\}, 8 -> \{9, 10, 12\} \\
&\textit{round}\ \ 3: \quad 6 -> \{7\}, 10 -> \{11\}, 12 -> \{13, 14\} \\
&\textit{round}\ \ 4: \quad 14 -> \{15\}
\end{aligned}
$$

Although the example process is described as four steps, we can discern that the algorithm itself is asynchronous and parallel since each discovering path is independent and no interference with each other.

For both algorithms, the stabilization of the P2P system is achieved by Chord. In this parallel algorithm, if all finger nodes in an interval fail, the current node can initiate the stabilization and refresh finger nodes in this failed interval. Then forward the request to the first finger node in the interval.

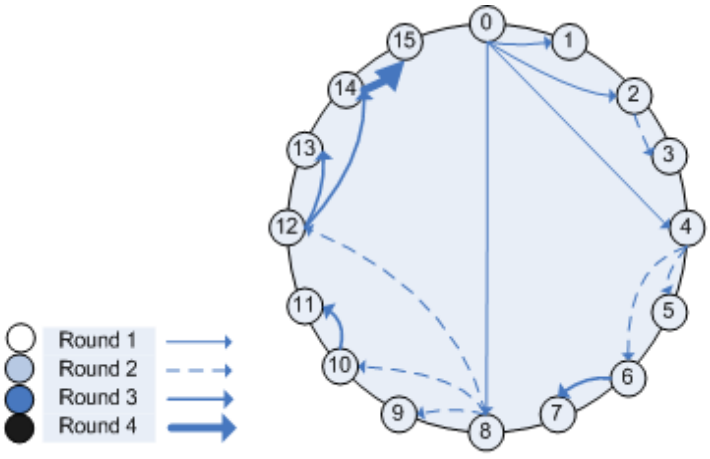

Figure 3: Parallel Resource Discovery Approach

4 Algorithm Analysis

In this section, we analyze the performance of the two algorithms presented in section 3. The metrics to be used in analysis are number of messages and number of rounds. Here number of rounds is used to show the time complexity.

In the flooding resource discovery algorithm, it is obvious that: (1) each node in Chord will receive the resource query if each node forwards the query when it receives, this is guaranteed by the Chord structure; (2) each node forwards the query to the nodes in its finger table only once, this is ensured by the algorithm because a node only forward a query when it receives the query in the first time. Combine the two observations, we know that the number of messages incurred in the algorithm is $O(n \log n)$ where n is the number of nodes in Chord, because each node has a finger table of size $\log n$ and each node forwards the query only once to the nodes in finger table. Thus the number of messages for forwarding query is $n \log n$. Only the node meeting the resource requirement will reply a message to the initiating node, thus the number of reply messages is at most n. So the total number of messages is at most $n \log n + n$, which is $O(n \log n)$. The number of rounds needed by the flooding resource discovery algorithm is $O(\log n)$, because the diameter of Chord structure is $O(\log n)$. This is equal to the maximal number of hops needed for key search in Chord [1].

In the parallel resource discovery algorithm, each node receives the resource query message at most once, this is ensured by our interval division and message forwarding mechanism. The number of query messages is at most n. Similar with the flooding algorithm, the number of reply message is at most n. Thus the total number of messages is at most $2n$. Also, it can be proved that the rounds needed for the algorithm to finish is $O(\log n)$.

5 Simulation and Comparison

The performance of parallel resource discovery algorithm in Chord can be judged by the following metrics: number of resources found, number of messages, discovery time.

The MIT chord simulator [2] is modified for our experiments. We did experiments to study the performance of the parallel resource discovery algorithm. The experiments are conducted as follows: We create a Chord network consisting of 10,000 nodes. Figure 4 shows the message complexity for both flooding and parallel resource discovery algorithms. To examine the effects of parallel degree, we change the parameter p (equals to 8 or 15) to see the number of resources found as a function of discovery time (Figure 5), where the ordinary Chord lookup is also presented for comparison; to examine the effects of churn rate, results shown in Figure 6 are cases when nodes leave on average of 1 second (low churn rate) and 0.5 second (high churn rate).

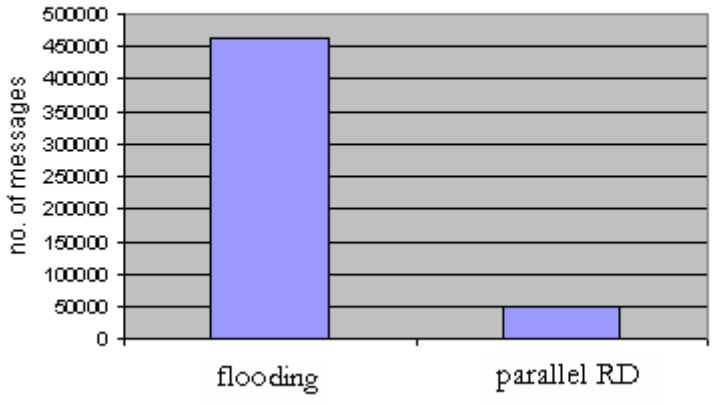

Figure 4: Message Complexity: Flooding versus Parallel

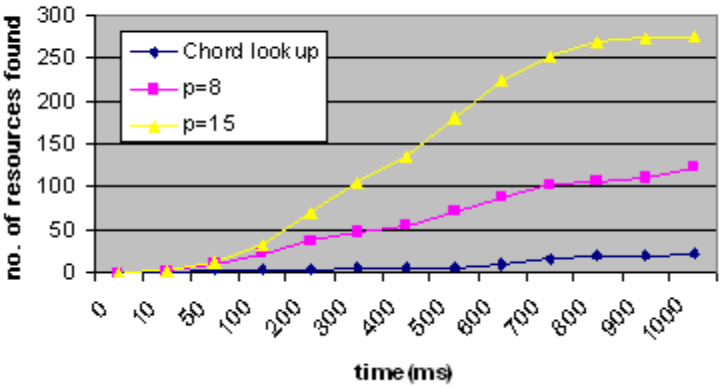

Figure 5: Effect of Parallelism

As shown in Figure 4, the message complexity for flooding is much bigger than parallel resource discovery, which is already explained in the previous section.

As observed in Figure 5, when the parallel degree p increases, the execution time is decreased dramatically when discovering the same number of required resources. Generally the parallel resource discovery is much better than multiple individual lookups in Chord, because of the power of node cooperation.

Figure 6 shows that when the network churn is faster (for p=8), it becomes harder for node to find the specified number of resources. This is because the dramatically changing network will definitely destroy the stability of Chord structure. Therefore when the forwarding node in the path of resource discovery fails, it can not finish the task as expected. The network churn will compromise the effort to reduce the

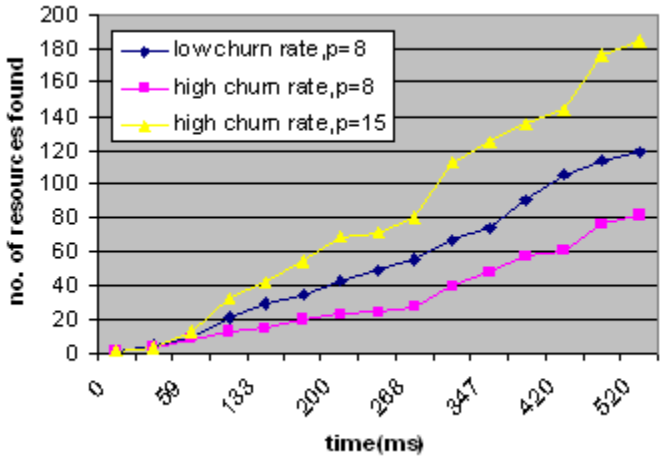

Figure 6: Effect of Network Churn

message complexity during parallel resource discovery. However, the impact from network churn can be relieved by increasing the degree of parallelism to p=15.

6 Conclusion

In this paper, we discuss the resource discovery problem in DHT-based computational grid computing networks. We proposed a parallel resource discovery approach and compare it with the simple flooding scheme. Both algorithms discover all nodes that match a resource requirement in $O(\log n)$ rounds. But the parallel approach has a better message complexity of $O(n)$ over the flooding algorithm with $O(n \log n)$ messages. Our proposed algorithm exploits the finger table of Chord structure and provides a tradeoff between message complexity and latency. Simulation experiments show that the degrees of parallelism can be varied to achieve different lookup performance. We also show that higher churn rate can affect the lookup performance but this can be relieved by increasing the degree of parallelism. Further work focuses on different partitioning strategies for parallel resource discovery and in improving fault tolerance.

References

[1] I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balarikishnan, *Chord: A Scalable Peer-To-Peer Lookup Service for Internet Applications*, in Proceedings of the 2001 ACM SIGCOMM Conference, pp. 149-160, August 2001.

[2] The Chord software package. [Online]. Available: http://www.pdos.lcs.mit.edu/chord/.

[3] M. Harchol-Balter, T. Leighton, and D. Lewin, *Resource Discovery in Distributed Networks*, in Proc. 15th ACM Symp. on Principles of Distributed Computing, pp. 229-237, May 1999.

[4] A. Iamnitchi and I. Foster, *On fully decentralized resource discovery in grid environments*, in International Workshop on Grid Computing, Denver, Colorado, 2001.

[5] C. Law and K.Y. Siu, *An o(logn) randomized resource discovery algorithm*, in Brief Announcements of the 14th International Symposium on Distributed Computing, Technical Report FIM/110.1/DLSIIS/2000, Technical University of Madrid, pages 5C8. October 2000.

[6] T.N. Ellahi and M.T. Kechadi, *Distributed resource discovery in wide area grid environments*, The 1st International Workshop on Active and Programmable Grids Architectures and Components APGAC'04, Krakow, Poland, June 6-9, 2004.

[7] S. Ramabhadran and S. Ratnasamy, *Prefix hash tree an indexing data structure over distributed hash tables*, in 23rd Annual ACM SIGACT-SIGOPS Symposium on Principles Of Distributed Computing, Newfoundland, Canada, July 2004.

[8] C. Schmidt and M. Parashar, *Flexible information discovery in decentralized distributed systems*, in Proc. 12th High-Performance Distributed Computing (HPDC '03), IEEE Press, 2003,pp. 226-235.

[9] Y.M. Teo, V. March and X. Wang, *A DHT-based grid resource indexing and discovery scheme*, in Proceeding of Singapore-MIT Alliance Symposium, Natinal University of Singapore, Singapore, January 2005.

[10] M. Balazinska, H. Balakrishnan, and D. Karger, *INS/Twine: a scalable peer-to-peer architecture for intentional resource discovery*, in Pervasive 2002 - International Conference on Pervasive Computing, Zurich, Switzerland, August 2002.

[11] I. Abraham and D. Dolev, *Asynchronous Resource Discovery*, in Proc. of the 22nd annual symposium on Principles of distributed computing, p.143-150, Boston, USA, July 13-16, 2003.

[12] S. Kutten and D. Peleg, *Asynchronous Resource Discovery in Peer to Peer Networks*, in Proceedings of the 21st IEEE Symposium on Reliable Distributed Systems (SRDS'02), p.224, October 13-16, 2002.

[13] B. Leong, B. Liskov, and E.D. Demaine, *Epichord: Parallelizing the Chord Lookup Algorithm with Reactive Routing State Management*, MIT Technical Report MIT-LCS-TR-963, Cambridge, MA, Aug. 2004