A DISTRIBUTED PROXY ARCHITECTURE FOR SERVICE DISCOVERY IN PEER-TO-PEER NETWORKS

Marcos Madruga[1], Thais Batista[2] and Luiz Affonso Guedes[3]
[1]Federal University of Rio Grande do Norte, CT- DEE, Campus Universitario - Lagoa Nova,59072-970 - Natal - RN – Brazil, madruga@unp.br; [2]Federal University of Rio Grande do Norte, CCET -DIMAp, Campus Universitario - Lagoa Nova,59072-970 - Natal - RN – Brazil, thais@ufrnet.br; [3]Federal University of Rio Grande do Norte, CT-DCA, Campus Universitario - Lagoa Nova,59072-970 - Natal - RN – Brazil, affonso@dca.ufrn.br

Abstract: In this work we present a service discovery system that supports flexible queries using partial keywords and wildcards. It is built upon a Chord network and it guarantees that any existing data that match a query is found. The main feature of this service is to use a proxy server layer with a mechanism for data distribution that reduces the number of nodes involved in the searching process.

Key words: service discovery, peer-to-peer, proxy, Chord, distributed searches.

1. INTRODUCTION

With the phenomenal success of Internet and the availability of a great amount of information, one of the main challenges nowadays is to find specific information. This problem was first noticed when it started to be hard finding specific information on web sites. This made sites like Google and Yahoo, that search the web for documents and other information, to become very popular. However, with the power of distributed computing and its different paradigms, the problem of finding components that provide a given service is going to a new dimension.

Several protocols have been proposed and we can generally divide them in two groups: (1) protocols developed to be use in limited environments, like local area networks such as SLP (Service Location Protocol) [12], UPnP (Universal Plug and Play Protocol) [13], Jini [14], and others; (2) a new series of protocols [5, 6, 7, 8, 9] developed to work in large scale, for instance those built on Peer-to-Peer networks [1, 2, 3, 4].

In this work we present a system that fits into the second category and uses a Chord [2] network as its foundation. It allows users to search for information by specifying keywords and wildcards. It also guarantees that existing data are found. The main feature of our system, when compared to others, is that it uses a proxy structure to accelerate searches and define a scheme to decrease the number of nodes searched for potential matches. This schema seams efficient even when the user gives little information to be used in the searching process.

This paper is structured as follows. Section 2 presents the background of the work that consists of briefly presenting Chord network. Section 3 presents the proposal of this work including the idea of distributed proxies and the changes we suggest in Chord. Section 4 presents the searching protocol. Section 5 comments about related work. Finally, section 6 presents the final remarks.

2. BACKGROUND

Chord [2] is a distributed lookup service used in peer-to-peer systems. It is not a storage system. It is based on the notion of consistent hashing and of an identifier space that is mapped to a set of nodes. A node is a process or a host identified by an IP address and a port. A chord identifier is associated with each node. Thus, high level names are translated into chord identifiers.

This work is built on a Chord Network, which is a peer-to-peer network and uses DHT (Distributed Hash Table) and consistent hashing [10, 11] techniques, to associate an identifier to each search key and each IP address of the nodes of the network. This is done through a hash function, like SHA-1, for instance, to the key and the IP address. It organizes the network in a ring layout, that is, to each key it is possible to determine the IP address of the server that contains it. In short: suppose that <hash(key)> is equal to 45. This means that this key will be on the server that has a IP address which hash function(IP) >= 45. When there is no server with hash(IP) = hash(key), the server related to that key will be the next in the ring that has an hash(IP) greater than that key.

To identify the next servers in the ring, each server provides a type of *router table.* The server localization contains at most O(Log N) messages traded by servers. In addition, the protocol supports the insertion and removal of nodes in the network and allows warning this to any application responsible for implementing the key migration to the new server (just the ones it is responsible for).

3. DISTRIBUTED PROXIES

Although the DHT (Distributed Hash Tables) based data publishing and localization schemes, as in the Chord Net, are already capable of distributing data between several servers, we use a technique that reduces dramatically the load on each of them through the insertion of a *proxy layer* between the nodes that query for data and the nodes that store them. Each proxy caches the query and serves several clients. Thus, instead of the nodes receive queries of N clients, they receive just from M proxies, where, of course, M is far less than N.

The distinguishing features of the proposed architecture are: it is automatic (no configuration is necessary to point which proxy to use), highly distributed (each node acts as a proxy) and each proxy takes charge of just a subset of keys, providing a greater rate of cache hit.

3.1 Chord Subnets

The model we propose is based on the creation of several Chord subnets inside the Chord global net. These subnets, however, are just logical subnets, as they are composed of the same machines of the global network. This idea, of course, requires changes in Chord in order to support that the same node be part of two networks (one of subnets and also the global network). These modifications are basically to duplicate the structure of the data that contain information about the nodes (and the keys) in the Chord ring, and to create an identifier for each network, that will be provided in the operation made through it (the network). The identifier will indicate which set of data structure should be used to determine the nodes that will be used.

Figure 1 shows a situation where there are three Chord subnets inside the Chord global network (containing nodes 1, 3, 6, 7, 10, 11, 15,19, 20 and 24) that allows 24 different keys. The first subnet has s1 as identifier and has nodes 1, 7 and 10. The second subnet has s2 as identifier and contains nodes 3, 11 and 19. Finally, the third subnet has s3 as identifier and contains nodes 6, 15, 20 and 24. Figure 2 shows the nodes and their related keys in the third subnet and in the global network.

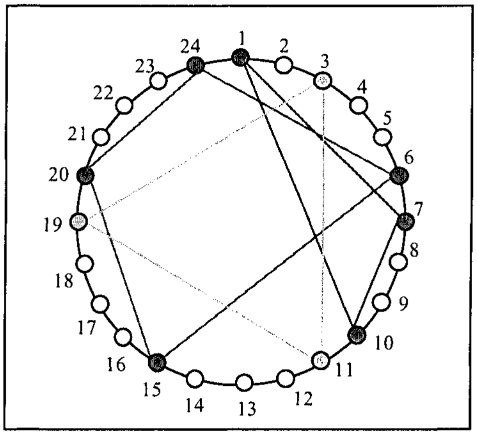

Figure 1. Chord Subnets.

Node	Keys	Node	Keys
1	1	11	11
3	2-3	15	12-15
6	4-6	19	16-19
7	7	20	20
10	8-10	24	21-24

Node	Keys
6	1-6
15	7-15
20	16-20
24	21-24

a)Global Network b) Third subnet: s3

Figure 2. Key distribution in the Chord Networks: Global Network and Third subnet

Each Chord subnet (sub ring) works as a proxy network (each node is a proxy). When a node is looking for data related to a specific key, it sends the query initially to the nodes of the Chord subnet to which it is part of, and these nodes (the proxies) are the ones that search for the data in the nodes of the global network. It is important to note that the nodes of the subnets store just the data associated to the keys in the subnet, as these indicate the set of data to each the proxy should be used.

3.2 Proxy and Operation Mode

The proxy identification (subnet node) to be used in the searching process for a specific key is done in the same way as in any Chord network. The only different aspect is the fact that the subnet identifier must be informed, so that the nodes are able to know whether they should use the data structure (that determine the next node) related to the global network or to the subnet. Also, we should notice that length of the key and hash function used are the same, for both subnet and global network. To use the same key length is possible due to the Chord architecture, which allows the responsibility of a set of N keys to be distributed in a network with any number of physical nodes. Also, the use of the same hash function simplifies and optimizes the system performance, as it allows calculating this function only once, with the proxy just forwarding the key search in the global network.

When a proxy server receives a query for a specific key with the network identifier being the local subnet, it is changed to the one of the global network and it forwards the query to the next node of the global network. This is done after searching its own cache to check if it already has that value stored locally.

Suppose that node 24 in Figure 1 is trying to find key 9. Initially it searches, in the local network (s3): it checks its subnet table (see Figure 2b) and determines that the node (proxy) responsible for that key is node 15. Thus, it forwards the query to that node and informs that subnet s3 is being referred to, that it is an operation inside the subnet. When node 15 receives the query, it changes the Chord net identifier from subnet to global network. It verifies in the global network table (see Figure 2a) that node 10 is responsible for key 9, and forwards the query to that node (in case it does not have the data related to this key in its cache). When the query is received, node 10 is informed by the network identifier that it is a global search. Thus, it will not act as a proxy, but as a normal node just recalling the data associated to the informed key of its local database.

3.3 Performance

It is well known that the performance of a Chord network is influenced by its capacity to determine the IP address of the node responsible for a specific key. This is done by contacting at most O (Log N) nodes, whereas N is the number of nodes in the network. However, as each proxy searches for just a one subset of the total keys, we can use an efficient cache system for the IP addresses of the nodes responsible for each key. In other words, each proxy besides caching the searched keys and the data associated to each of them, it also caches IP addresses of the nodes associated to each of them. This way we reduce the complexity of the searches from O(Log N) to O(Log S), where S is the number of nodes in the subnet. Thus, it is far less than N.

4. SEARCH PROTOCOL

A Chord network is able to process simple key searches. It is possible to determine in which node a given identifier is stored. This way it cannot be used directly as a protocol for complex searches with detailed descriptions of services. Therefore, usually a Chord network (or other peer-to-peer networks, like CAN [3]) is used as basis for building refined search protocols. Some desirable features in such protocols are: (1) Ability to distribute the registry of services, even if they are of the same kind, to different servers. This avoids demanding services to end up overflowing specific nodes; (2) Load balance, so that the loading of performing searches is distributed between each server; (3) Exact matches of complete sentences must be found in a very precise way; (4) Guaranteed success in the search if the data is available in the network.

Services can be described by a pair (attribute, value) or by a XML document. In this work we will use the former, and not the XML approach, even though they are very similar. Also, we assume that a service description has an attribute identifying each service type, for example: printer.

4.1 Service Registry

The service registry process consists of the storage of each attribute of the service in a different node, together with the service type identifier and a link (an URL, for instance) to the complete description of the service. Each link should identify the service in a unique way, as it will associate the various attributes of the service.

Although the attributes are registered separately, complex searches dealing with various attributes are also possible by splitting the search into several simple searches. The simple search deals with just one of the attributes. After, the results are grouped according to the operators used (OR, AND and so on). Another way of searching, that is more reliable for searches dealing with the operator AND, is to send the search to the node responsible for one of the specified attributes. From this node each one selects the services that match the specified criteria and forwards the search to the node responsible for the next attribute. The services found would be forwarded from node to node are refined at every new step.

The identification of a node where an attribute should be stored is calculated in the following way:

1. It is calculated the hash function of the text resulted from the concatenation of the name of the type of service and the name of the attribute, that is: hash(kind_service+name_attribute).

2. It is calculated the numeric value for the data in the attribute, using table 1, where each character is associated to a value. The calculation consists of adding the numeric value of each character of the text.
3. For each word in the attribute, we search for the character with the least numeric code associated and subtract this value from the numeric code associated to the character with greatest value. The biggest value is used.
4. It is divided the keys from the one obtained in step 1 in intervals of *n* keys, where *n* is the code of the greatest value in table 1.
5. The value obtained in step 2 is added to the key calculated in step 1 creating, then, a new key.
6. It is verified to which of the intervals calculated in step 4 the obtained key is part of. The data will be stored in the node *m* of this interval, where *m* is the value calculated in step 3.

Table 1. Table of characters codes

Code	***Character***	***Code***	***Character***	***Code***	***Character***
6	A	8	C	10	E
7	B	9	D	...	...

Figure 3 shows the process of identifying the node responsible for the *color* attribute of a service called "*car*". Note that the hash function result is *node 3* and that adding to this number the value obtained by the sum of the numeric code of all characters of the attribute value, that is, black, we obtain the value 8 is result. Assuming that the greatest code in the characters table is 4, the nodes from node 3 obtained through the hash function, are divided in intervals of 4 nodes. Whereas, interval 1 has nodes 3, 4 ,5 and 6. Interval 2 has nodes 7, 8, 9 and 10.. As node 8 is in the second interval the data will be stored in any of its nodes. The exact node of the interval is determined by the difference between the codes of the characters 'k' and 'a' obtained from the value "black". As for our example we assumed that this difference is 4, the resulting node is node 10.

4.2 Queries

When a search for an attribute is composed of the complete value of the data in the attribute, we just need to make calculations identical to those done in the registry process, already explained, in order to obtain the specific node where the attribute is located. Note that, this approach can guarantee the requirement that exact searches must be as efficient as possible.

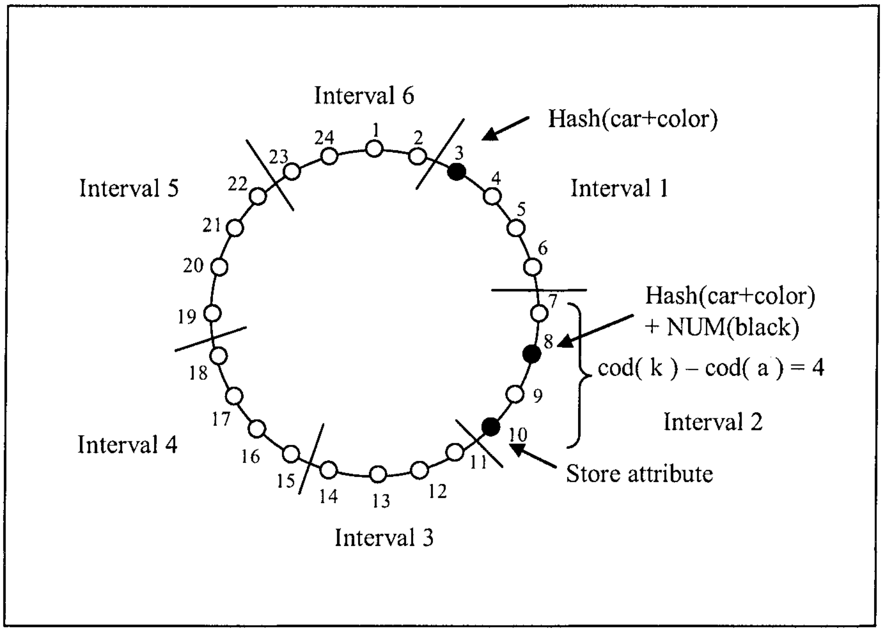

Figure 3. A identification of the node responsible for storing the value of the attribute "color=black" for a "car" service. We assume that the sum of the codes of the characters of "black", NUM(black), is equal to 5 and that the difference between letters 'k' and 'a' (greatest and least characters of "black") is equal to 4.

However, in a partial search, searching for keywords or expressions like comput*, for instance, does not return the specific nodes where the data are. The motivation of using a method that ascribes a numeric value to the data in the attribute through the sum of the codes in its characters is that this places a substring in every node right before the one responsible for the complete string. Also, as words characters are given by the user, it will be closer to the node containing the data. Therefore, in a partial search the results of the calculations inform the node where the search must start from, but the subsequent nodes in the Chord ring must also be queried.

This composition of nodes can be very big. In order to address this problem, a technique was developed to divide the nodes after the base node (pointed by the application of the hash function) in intervals and use the difference between the codes with greatest and least character of the data to help determine the exact node where the data is. Our goal is that if just part of the complete text contained in the attribute is informed, the difference between the greatest and the least character in the informed text will always be less or equal to this difference calculated in the complete text. Thus, for each range from the one containing the node calculated as initially, just the node above the value of the difference between the greatest and the least characters informed in the search must be searched.

4.3 Optimizations

The number of the intervals to be searched can also be reduced if it is identified the data that creates the greatest numeric value possible (sum of the characters codes) for the searched attribute. To get this information it is necessary to use the length of the attribute field. To handle this issue, we concatenate to the searched data as many characters as necessary to hit the maximum length of the field. The character used should be the character with the greatest numeric value associated (see table 1). To determine the field length, the proxies should analyze the information obtained for the attribute and use the greatest data size already recalled, to calculate the maximum size.

Although this optimization provides the results expected from most of the searches, some data may not be found. Therefore, a mechanism should be provided to allow the user activate or not this optimization.

5. RELATED WORKS

Squid [8] is a searching system that also supports searches by using keywords and wildcards. It uses a space filling curve based on an index scheme to map data elements to nodes using keywords. The main features of our system is that it requires some keywords to both describe the service and the search, while it keeps its efficiency even with a small amount of information provided by the user. In addition, even though systems provide a fairly similar mechanism (based on the association of the numeric codes) to determine the set of nodes to match the query, our approach is different.

The INS/Twine [5] system registers services by calculating a hash function that involves both the field name and the given data. This calculation is processed several times (for the different fields of the service). Then, it stores the complete description of the service in each node. This approach decentralizes the services register but allows just exact searches. Our approach also uses the idea of registering the service based on the value of its attributes, but it registers just the data related to the attribute in each server and modifies the way that the hash function is used. Thus, partially searches can be made.

6. CONCLUSIONS

In this paper we have presented a service localization mechanism built upon a Chord network. However, it proposes some changes to Chord in order to support a proxy server layer, containing the network's own nodes, that cache the queried data in order to accelerate the searching process. The proxy architecture allows the automatic identification of the proxy to be used and it also supports fault tolerance. Another important contribution is a service distribution method between the nodes in the network, that allows the dramatically reduction of the number of potential matches in a search.

References

1. P. Rowstron and P. Druschel.: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. Lecture Notes in Computer Science, 2218, 2001.
2. A.Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan. Chord: Scalable Peer-To-Peer lookup service for internet applications.In Proc. 2001 ACM SIGCOMM Conference, pages 149–160, 2001.
3. S. Ratnasamy et al.,"A Scalable Content-Addressable Network," Proc. ACM SIGComm, ACM Press, 2001, pp. 161–172.
4. C. Plaxton, R. Rajaraman, and A.W. Richa: "Accessing Nearby Copies of Replicated Objects in a Distributed Environment," Proc. ACM SPAA, ACM Press, 1997, pp. 311–320.
5. M. Balazinska, H. Balakrishnan, and D. Karger. INS/Twine: A scalable peer-to-peer architecture for intentional resource discovery. In Proceedings of the First International Conference on Pervasive Computing, pages 195–210, Zurich, Switzerland, August 2002. Springer-Verlag. College, February 2002.
6. D. Spence and T. Harris. Xenosearch: Distributed resource discovery in the xenoserver open platform. In *Proc. of HPDC 2003*, Seattle, WA.
7. C.Tang, Z. Xu, and M. Mahalingam, PeerSearch: Efficient Information Retrieval in Peer-to-Peer Networks, tech. report HPL-2002-198, HP Labs, 2002.
8. C. Schmidt and M. Parashar, Enabling Flexible Queries with Guarantees in P2P Systems, - Internet Computing Journal, Vol. 8, No. 3, May/June 2004
9. R. Huebsch, J. M. Hellerstein, N. Lanham, B. T. Loo, S. Shenker, and I. Stoica. Querying the internet with pier. In Proc. of the 29th International Conference on Very Large Data Bases, September 2003.
10. D. Karger, E. Lehman, F. Leighton, M. Levine, D. Lewin, and R. Panigrahy. Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the world wide web. In *Proceedings of the 29th Annual ACM Symposium on Theory of Computing* (May 1997), pp. 654–663.
11. D. Lewin. Consistent hashing and random trees: Algorithms for caching in distributed networks. Master's thesis, MIT, 1998. Available at the MIT Library, http://thesis.mit.edu.
12. E. Guttman. Service location protocol: Automatic discovery of IP network services. *Internet Computing*, July/August 1999.
13. UPnP Forum: Understanding Universal Plug and Play: A white paper. http://upnp.org/download/UPNP_UnderstandingUPNP.doc (2000)
14. S. Microsystems. Jini architecture specification, December 2001.