

Mobile agent-based approach for resource discovery in peer-to-peer networks

J. Gaber and M. Bakhouya

Laboratoire Systemes et Transports (SeT)
Université de Technologie de Belfort-Montbéliard (UTBM)
90010 Belfort, France
gaber@utbm.fr, bakhouya@utbm.fr

Abstract. Large scale networks such as peer-to-peer networks are distributed computing infrastructures that can provide globally available network resources. Their size and complexity continue to increase and permit an almost ubiquitous availability of resources; users become able to access network resources irrespective to their location. Therefore, new resource management and access models are required and need to be highly flexible in order to cope with a dynamically changing environment. In this paper, a cloning approach for resource discovery in peer-to-peer network is proposed. This approach is based on the mobile agent paradigm and uses random walks to allow dynamic and adaptive resource discovery. We analyze this approach through three distributed resource discovery scenarios by NS2 simulator.

1 Introduction

The rapid growth in personal computing and commodity networking has created large-scale distributed-computing environments such as computational grids [1]. These environments have the potential ability to integrate large-scale computing resources. Trends today are towards networking resources. User ability to compute will no longer be limited to the resources he has currently at hand or those localized statically on a set of hosts known a priori. For example, a user with a diskless machine can load a word-processor for a punctual need. He no longer needs necessarily a machine with a disk and his own word-processing software installed on it. In addition, he may not know in advance from which host he get the resource throughout his current session. Also, users as well as resources can be mobile or partially connected. The availability of the host and its resource depends on the dynamic behavior of the network (e.g., failure or network disconnection). When the user submits its request, eventually with QoS requirements, a resource discovery system will process it according to the network status at that instant, and in order to satisfy the QoS constraints [22]. In peer-to-peer network, centralized architecture cannot meet the requirements of both scalability and adaptability simultaneously. Issues are that network resources must be able to scale and able to adapt to dynamic conditions in the

network. In this paper, a cloning approach based on mobile agent for resource discovery in peer-to-peer network is presented.

The rest of the paper is organized as follows. Section 2 presents the related work. In section 3, the proposed approach is presented. Section 4 presents the simulation results. Conclusion is given in section 5.

2 Related work

Resource discovery is an important issue in peer-to-peer network; given a user request, a resource discovery mechanism should locate and return a set of peer addresses that match the description of the requested resources. Resources can be divided into two basic categories [1], [6]: system resources and application resources. System resources are bound to specific hosts, representing hardware devices (e.g. disk) or logical system objects. Application resources are software entities managed by an application. In this paper, a service is considered to be composed by a set of resources that users need to discover and select [22], [23]. These resources could be interfaced by web services. Web services are applications that permit to describe software components; in particular they define identification and accessing methods that enable the discovery and the use of these components (i.e., the resources). For example, for a punctual need, a user peer who would like to open video files or create video CD might need the following resources: a video player, format transcoding software, the MPEG-4 codec (for his wireless laptop), video effect or edge detection algorithms, etc.

Typical resource discovery architecture consists of three entities: resource providers that create and publish resources, resource brokers that maintain a repository of published resources to support their discovery, and resources requesters that search the resource broker's repositories. Repositories have traditionally a hierarchical architecture consisting of multiple repositories that synchronize periodically [1], [7], [8]. In a large-scale network, hierarchical architecture cannot meet the requirements of both scalability and adaptability simultaneously. The way in which they have typically been constructed is often very inflexible due to the risk of bottlenecks and the difficulty of repositories updating [1], [2], [3]. Also, peer-to-peer network is a dynamic environment where the location and availability of resources are constantly changing. More precisely, some resources could be disconnected from the network and new ones may join it at any time.

Therefore, today, a resource discovery mechanism should be decentralized to avoid bottlenecks and guarantee scalability and adaptability. Several systems implement non hierarchical decentralized infrastructures. These systems can be classified as either structured or unstructured [12]. Structured systems (Chord [9], Pastry [10], and Tapestry [11]) use Distributed Hash Tables (DHTs) to assign responsibility for each file to specific peers. This structure permits to implement a direct search algorithm to efficiently locate files. In contrast, unstructured systems (Gnutella [14]) have no pre-

cise control over the file emplacement and use flooding search protocols. Both systems present some drawbacks. Structured systems use overlay network (i.e., DHT) between peers that are generally hard to maintain. In particular, peer join/leave operations could incur huge overheads [13]. In contrast, unstructured systems allow the peer to enter and leave the systems without overheads.

In unstructured systems, the most typical localization method is flooding, where the request is broadcast to all neighbors within a certain radius with TTL mechanism (TTL for Time To Live) [12]. More precisely, in order to find a file on the network, search queries were flooded to peer nodes. Each query had an attached time to live to control the number of hops that a query can be propagated. Each node that passed a query to its peer would decrement the TTL. When the TTL reached 0, the query was no longer forwarded. However, it is not possible to guarantee the success or failure of a query. In other words, a file may not be found even though it does in fact exist on the network. To overcome this disadvantages, the mechanism of dynamic TTL or expanding ring is proposed in [13]. Principle of expanding ring is as follows: a peer starts a flood with small TTL, and waits to see if the search is successful. If it is, then the peer stops the flooding. Otherwise, the peer increases the TTL and starts another flood. The process repeats until the object or file is found or covering all network. According to [12], [13], expanding ring guaranties that if the file is presents in the network it will be found compared with regular flooding with a fixed TTL. However, expanding ring mechanism solves the TTL selection problem, but does not address the message duplication issue inherent in flooding that can generate large loads on the network [12], [13]. Random walk-based search mechanism, which forwards a query message (walker) to a randomly chosen neighbor at each step until the service is found, is a well-known technique that can avoid this problem. Using one walker, it cuts down the message overhead significantly. However, it increase delay of successful searches [12], [13]. To decrease the delay, a requesting peer sends k query messages, and each query message takes its own random walk. However, it is difficult to determine the number of walks and when this number is big enough, the message traffic increases significantly [12]. The replication mechanisms, such as cache some objects in the reverse path of queries, are proposed in [13] in order to reduce the lookup length and decrease the message traffic. However, in dynamic and distributed setting, it is difficult to maintaining the coherence of duplicated objects.

In this paper, we will use both random walks and a cloning mobile agent-based approach for resource discovery in unstructured peer-to-peer network wherein peers might dynamically and unpredictably join/leave the network, such that any complete knowledge of these changes and modifications is difficult even impossible to obtain.

3 Resource discovery approach

Unstructured peer-to-peer network can be viewed as an indirect connected graph $G=(S, V)$, where S is the set of peers ($|S|=n$) and V the set of links ($|V|=m$) connecting the peers. A peer p_i is considered to be connected to a peer p_j , if there is a link be-

tween p_i and p_j . In this section, the possibility of embedding the mobile agent paradigm and the random walk approach in designing distributed algorithm for resource discovery in unstructured peer-to-peer network is analyzed. A mobile agent is a software entity which may move from location to location to meet other agents or to access resources provided at each location. The mobility of agents is the basic difference from the client-server approach [17], [18]. A random walk on a graph is a stochastic process that iteratively visits the vertices of the graph. From a given peer, the walk process moves at the next step to an adjacent peer chosen uniformly at random [4], [20].

To use mobile agents for a resource discovery, let consider the following three scenarios. The first scenario associates a unique agent to each peer request while the second scenario involves multiple agents for each request. The third scenario uses cloning operation to clone an agent during its random walk.

In the first scenario, to locate a service, the requester peer (origin of the request) creates a mobile agent, called *request agent*, and gives it the service to be located. A service can be composed of one or a set of resources (r_1, r_2, \dots, r_m) . The mobile agent starts from a requester peer and then uses links within it to get access to other peers. The mobile agent chooses randomly between these peers, determines the IP address of the chosen one and moves to the corresponding peer. The mobile agent repeats this process with the new peers reached until it find the required resources. Upon mobile agent termination (i.e. success or fail) it starts a backtracking phase. During this phase, the agent comes back using the path computed between the peer holding the last remaining resource to collect (i.e., an end point of walk) and the requester peer. It should be noted that compared to client/server approach, in this scenario, the single mobile agent eliminates the transfer of intermediate results across the network and thus reduces the end-to-end latency [18]. However, the time to resolve the request could be unreasonable in large scale network where peers have no preexisting knowledge of where resources are located so searching for them could require $O(n^2 \log(n))$ steps [21]. More precisely, $O(n^2 \log(n))$ steps are required for a mobile agent to cover a given graph with n nodes i.e., the mobile agent visits all nodes of the graph.

To reduce this latency problem, multiple mobile agents and mobile agent cloning scenarios could be more suitable scenarios for the resolution request process. In the multiple agents' scenario, an initial population of mobile agents is initially created and dispatched randomly for resource discovery. This scenario should allow the agents to resolve request in a reasonable amount of time compared to the single mobile agent scenario. However, it is difficult to determine the initial mobile agent population size. When this number is big enough, the agents traffic increases significantly, but the delay of successful searches is decreased. Also, the use of very small number decreases the agents traffic and increases delay of successful searches.

In the mobile agent cloning scenario, a mobile agent starts, at its first step, on its requester peer. At each hop, mobile agent determines the IP address of randomly

chosen neighboring peers, creates replication (i.e. clone) to itself, passes tasks to this clones that move to further peers. This scenario should allow mobile agents to cover a much wide area of network peers in a reasonable amount of time compared to the single mobile agent scenario and the multiple mobile agents' scenario. It's worth noting that in multiple mobile agents' scenario, initial population size does not change at each step but in mobile agent cloning scenario, it evolves during the random walks.

More precisely, the algorithm of mobile agent cloning scenario is as follows. The peer willing to locate a service creates a mobile agent, called *request agent*. This agent initiates a random walk in the network until it meets appropriate peers that can resolve the request or it terminates its random walk. At each hop, the mobile agent can clones itself. The request discovery process is made in two phase: forwarding phase and backtracking phase. During the forwarding phase, request agents seek peers that can provide the required resources. When the all required resources are discovered, a request agent stops cloning itself and starts the backtracking phase. In this phase, mobile agent travels back to its initial peer following back the founded path. The role of this backtracking phase is to perform a reinforcement learning mechanism on links between peers [16]. The objective of this backtracking phase is to permit for peers to learn from mobile agents satisfactions on past requests to carry out biased random walk in order to improve performance of future requests.

During its random walk, a request agent stores the list of the visited peers. Based on this stored list, called *service path*, the agent chooses moving to peers that are not visited yet. However, mobile agents require a mechanism to terminate their walks. To this aim, a mobile agent starts with an initial TTL. If the required service is found, it stops the search and starts the backtracking phase. Otherwise, the agent checks if the requester has already get the service from another clone. If it is the case, the agent is killed. If not, the requester could assign a new initial TTL to the agent and initiates a new random walk.

It is worth noting that, in the agent cloning scenario, the increasing of the agent population size with cloning operation will increase resource demands in the network which will affect the overall performance. The distributed algorithm proposed by Amin and Mikler in [5] is used to regulate and control dynamically the number of clones spawned in the network. This approach is inspired by stigmergetic propriety of "ant colony" to facilitate coordination between mobile agents. More precisely, mobile agents with minimum cognitive capabilities communicate with each other using pheromones that assist them to select an appropriate action. The intensity of pheromones disposed by agents at each node visited is determined by the equation $e^{-\lambda \Delta t}$, where Δt is the time since the deposition of pheromone and λ is a constant value fixed between 0 and 1. The controller of each agent contains the action selection algorithm that is defined as follows. An agent visiting a node at time t_b extracts the value of the value of the pheromone that was disposed at time t_a ($t_a \leq t_b$) using the

equation $e^{-\lambda(t_b-t_a)}$. If this value is above a certain termination threshold Max , the agent kills itself. On the other hand, if the pheromone value reduces below a cloning threshold Min , the agent clones itself. But, if the pheromone is comprised between the termination and cloning threshold, the agent neither clones nor kills itself. In this case, it migrates to another peer node. Another auto-adaptive distributed algorithm inspired by the Human immune system proposed by Bakhouya and Gaber in [24] could also be used to regulate the agents population size in large scale networks. The immune system has emergent properties to make self-regulating and self-adapting in dynamically changing environment [22], [23]. In this algorithm, each mobile agent selects an appropriate behavior to its environment state from the following ones : death, moving or cloning without using any thresholds parameters [24].

Figure 1 shows an example of how request resolution process works with cloning scenario. Peer P_1 possesses the resource R_1 and desires locate a service $S=(R_1,R_2,R_3,R_4)$.

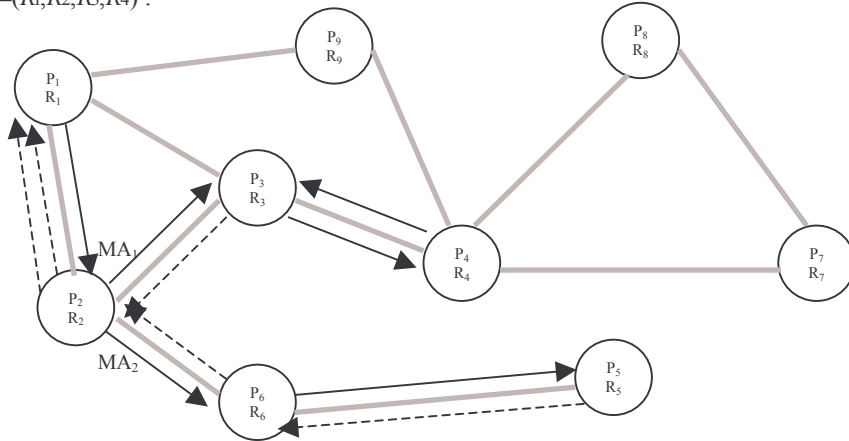


Fig. 1. The request forwarding and backtracking phase to locate the service $S=(R_1,R_2,R_3,R_4)$ with mobile agents cloning scenario.

To locate this service, P_1 creates a request, initiates one mobile agent MA_1 and gives it the list of resources to be located, the IP address and the initial TTL. The agent add the peer P_1 to its visited peers list (i.e., service path) and moves to the peer P_2 chosen randomly. Since, this peer provides the required resource R_2 , it is added to the visited peers list. At this step, MA_1 residing in peer P_2 creates another agent MA_2 that walk to peer P_6 . These agents repeat the same process until they find the required service or their TTL is expired (i.e., TTL becomes equal to 0). The visited peers of request forwarding phase of two mobile agents are shown with thicker arrows. The mobile agent (MA_2) with service path (R_1,P_2,R_6,R_5) fails, while the second mobile agent (MA_1) with service path (R_1,P_2,R_3,R_4) terminates with success. During the backtracking phase, the mobile agents goes back from the last peer visited, via the intermediate peers on the

founded service path, to the initial peer. The backtracking phase is started for this two mobile agents on the reverse path shown in figure 1 with dotted arrows.

4 Simulation results

Our simulator is implemented by NS2 [15]. The objective of the simulation is to compare these three mobile agent-based scenarios for the resource discovery. A network of 100 peers is generated randomly with BRITE generator [19]. Each peer provides one resource of ten kinds of resources. The simulation abstracts any considerations about networking issues such as bandwidth constraints and time processing. The code and state size of mobile agent are constant and do not change during the simulation. Recall that the objective of the resource discovery system is to discover and select peers that can resolve the user request.

In the figure 2, the cloning mobile agent scenario shows a complexity time better than that of single and multiple mobile agents scenarios independent of searched service (one, five or ten resources). This is due to the number of agents launched by cloning operation for request discovery. Intuitively, mobile agents may be cloned and dispatched in different directions. This end allows mobiles agents to cover a much wide area of network peers in a shorter time. Also, multiple mobile agent scenarios allow a requester peer to create a fixed number of mobile agents (5 mobile agents in this simulation) and dispatch them in different directions. This scenario allows mobile agent to work in parallel. Since, it shows a time a little greater than that of cloning agents scenario.

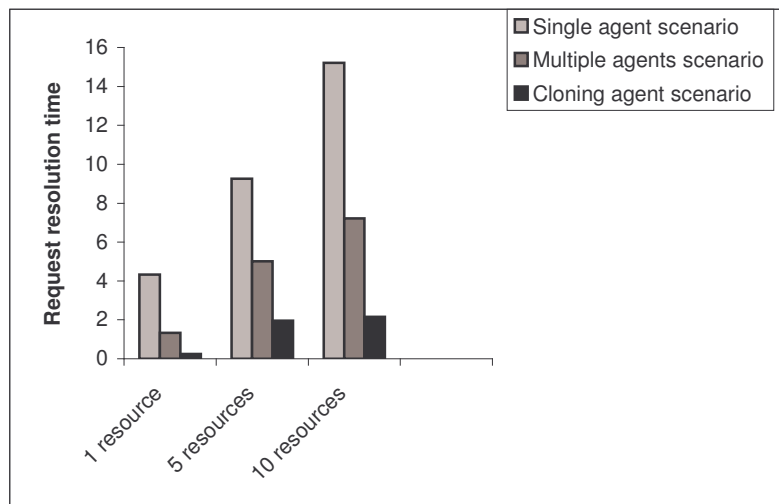


Fig. 2. Resolution request time for mobile agent scenarios

5 Conclusion

This paper analyzes three scenarios for resource discovery in unstructured peer-to-peer network based on the mobile agent-based approach together with a random walks technique. From the simulation, the agent-based approach with cloning scenario out performs the single agent scenario and the multiple agent scenario.

Future work will address the specification issue of the proposed approach together with additional simulations with ns2 to evaluate the approach performance with various biased random walks schemes.

References

1. Krauter K., Buyya R., Maheswaran M.: A taxonomy and survey of grid resource management systems for distributed computing. *Software Practice and Experience*, Vol. 32, Number 2, pages 135-164, 2002.
2. Iamnitchi A., Foster I, Nurmi D.: A peer-to-peer approach to resource discovery in grid environments. In *HPDC'02*.
3. Talia D., Trunfio P.: Web services for peer-to-peer resource discovery on the grid. 2002. www.grid.it/.
4. A. Broder Z., Karlin A. R., Raghavan P., Upfal E.: Trading Space for Time in Undirected s-t Connectivity, *ACM STOC'89*, pages 543-549.
5. Amin K.A, Mikler A.R: Dynamic Agent Population in Agent-based Distance Vector Routing, *Second International Workshop on Intelligent Systems Design and Applications*, Atlanta, USA, August 2002.
6. Gidron Y., Holder O., Ben-Shaul I, Aridor Y.: Dynamic configuration of access control for mobile components in Fargo. *Concurrency and Computation*, 13(1):5–21, January 2001.
7. Czerwinski S., Zhao B., Hodes T., Joseph A., Katz R.: An architecture for a secure service discovery service. *Proceeding of ACM MobiCom'99*, Sep. 1999.
8. Xu D., Nahrstedt K., Wichadakul D.: Qos-aware discovery of wide-area distributed services. In *First IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid)*, 2001.
9. Stoicay I, Morrisz R., Liben-Nowellz D., Kargerz D.R., Frans Kaashoekz M., Dabekz F., Balakrishnanz H.: Chord : A scalable peer-to-peer lookup protocol for internet applications. <http://www.pdos.csail.mit.edu/papers/ton:chord/paperton.pdf>, 2001.
10. Rowstron A., Druschel P.: Pastry : Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. *Lecture Notes in Computer Science*, N_2218, 2001.
11. Zhao B.Y, Kubiatoiwicz J.D, Joseph A.D.: Tapestry : An infrastructure for fault-tolerant wide-area location and routing. UC Berkeley, UCB/CSD-01-1141, 2001.
12. Wang C., Li B: Peer-to-peer overlay networks : A survey. <http://comp.uark.edu/cgwang/Papers/TR-P2P.pdf>, 2003.
13. Lv Q., Cao P., Cohen E., Li K., Shenker S.: Search and replication in unstructured peer-to-peer networks, in *Proceedings of 16th ACM International Conference on Supercomputing (ICS'02)*, New York, USA, June 2002.
14. Gnutella: <http://www.gnutella.com/>
15. The NS-2 patch, available at the Faculty of Information Technology, Mathematics and Electrical Engineering, Department of Telematics web site: <http://www.item.ntnu.no/~wittner/ns/index.html>

16. Bakhouya M., Gaber J.: A reinforcement learning of link affinities and user requests for Self-adaptive graph emergence from an arbitrary graph, Technical report, SeT-UTBM, pp 1-12, Juillet 2003.
17. Carzaniga A., Picco G.P, Vigna G.: Designing distributed applications with mobile code paradigms. Proceedings of the 19th International Conference on Software Engineering, Boston, MA, 1997.
18. Straber M., Schwehm M.: A performance model for mobile agent systems. Proc. Int. Conf.on. Parallel and Distributed Processing Techniques and Application (PDPTA'97), Las Vegas, pages 1132–1140, 1997.
19. The BRITE generator, available at the Computer Sciences Department of Boston University web site: <http://www.cs.bu.edu/brite/>.
20. Baala H., Flauzac O., Gaber J., Buid M., El-Ghazawi T.: A self-stabilizing distributed algorithm for spanning tree construction in wireless ad hoc networks, Journal of Parallel and Distributed Computing Volume 63, Issue 1, pp. 97-104, January 2003.
21. Broder A., Karlin A.: Bounds on the Cover Time, Journal of Theoretical Probability, 2(1):101--120, January 1989.
22. Bakhouya M., Gaber J. : Adaptive Approches for Ubiquitous computing. To appear in Mobile networks and wireless sensor networks, Eds. H. Labiod, ISBN 2-7462-1292-7, Lavoisier-Hermes Science, pp 129-163, Mars 2006.
23. Bakhouya M.: Self-adaptive approach based on mobile agent and inspired by human immune system for service discovery in large scale networks, PhD Thesis, Universite de Technologies de Belfort-Montbeliard, 2005.
24. Bakhouya M., Gaber J. : Distributed autoregulation approach of a mobile agent population in a network. Technical report. Laboratoire SeT-UTBM. pp. 1-14, December 2002.