Performance Analysis of Distributed Search in Open Agent Systems

Vassilios V. Dimakopoulos and *Evaggelia Pitoura*
Department of Computer Science, University of Ioannina
Ioannina, Greece GR-45110
{dimako,pitoura}@cs.uoi.gr

Abstract

In open multi-agent systems agents need resources provided by other agents but they are not aware of which agents provide the particular resources. Most solutions to this problem are based on a central directory that maintains a mapping between agents and resources. However, such solutions do not scale well since the central directory becomes a bottleneck in terms of both performance and reliability. In this paper, we introduce a different approach: each agent maintains a limited size local cache in which it keeps information about k different resources, that is, for each of k resources, it stores the contact information of one agent that provides it. This creates a directed network of caches. We address the following fundamental problem: how can an agent that needs a particular resource find an agent that provides it by navigating through this network of caches? We propose and analytically compare the performance of three different algorithms for this problem, flooding, teeming and random paths, in terms of three performance measures: the probability to locate the resource, the number of steps and the number of messages to do so. Our analysis is also applicable to distributed search in unstructured peer-to-peer networks.

1. Introduction

In multi-agent systems (MAS), agents cooperate with each other to fulfill a specified task. As opposed to *closed* MAS where each agent knows all other agents it needs to interact with, in *open* MAS such knowledge is not available. To locate an agent that provides a particular resource, most open MAS infrastructures follow a central directory approach. With this approach, agents register their resources to a central directory (e.g. a middle agent [12]). An agent that requests a resource contacts the directory which in turn replies with the contact information of some agent that provides the particular resource. However, in such approaches, the central directories are the bottlenecks of the system both from a performance and from a reliability perspective.

In this paper, we introduce a new approach to the resource location problem in open multi-agent systems. Each agent maintains a limited size private cache with the contact information for k different resources (i.e. for each of the k resources, the agent knows one agent that offers it). This results in a fully distributed directory scheme, where each agent stores part of the directory. We model this system as a network of caches. There is a link from node v to node u if and only if agent v knows agent u, that is agent's u contact information is stored in v's cache.

Caching can be seen as complementary to directories. Small communities of agents knowing each other can be formed. Such a fully distributed approach eliminates the bottleneck of contacting a central directory. It is also more resilient to failures since the malfunction of a node does not break down the whole network. Furthermore, the system is easily scalable with the number of agents and resources.

In abstract terms, this results to the following problem. Let $G(V, E)$ be a directed, not necessarily connected graph, where each node (i.e. agent) $v \in V$ is connected with at most k other nodes. If there is a directed edge from a node v to another node u, we say that v *knows* about u. We address the following questions: starting from an arbitrary node v how can we reach (learn about) another node u, what is the probability to reach u and what is the associated communication cost.

The general mechanism for locating a resource is as follows. The agent that requires a resource first looks at its cache. If no contact information for the resource is found in the cache, the agent selects other agent(s) from its cache, contacts them and inquires their local cache for the resource. This procedure continues until either the resource is located or a maximum number of steps is reached. In essence, this procedure constructs directed path(s) in the network of caches. If the resource cannot be found, the agent has to resort to some other (costly) mechanism (e.g. to a middle agent) which is guaranteed to reply with the needed information. We provide a number of different algorithms for this procedure. In particular, we study the ef-

0-7695-1926-1/03/$17.00 (C) 2003 IEEE
Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS'03)

fectiveness of using single and multiple search paths.

The performance metrics we are interested in are:

- the probability to locate a resource within t steps,
- the average number of steps needed to locate a resource, and
- the average number of message transmissions required.

The probability of locating a resource is important because it is directly connected with the frequency with which an agent resorts the the 'other' mechanisms (e.g. middle agents), and should be as high as possible. On the other hand, the number of message transmissions which is but one measure of the communication cost, should be as low as possible. For each of the above quantities, we provide analytical estimations and simulation results that verify them.

The remainder of this paper is structured as follows. A summary of related work is given in Section 2. Section 3 introduces a number of searching algorithms while Section 4 presents analytical results of their performance. Section 5 includes our simulation results and, finally, Section 6 concludes the paper.

2. Related work

The only other study of the use of local caches for resource location in open MAS that we are aware of is that in [10]. However, in this work, only the complexity of the very limited case of lattice-like graphs (in which each agent knows exactly four other agents in such a way that a static grid is formed) is analyzed.

The problem that we study in this paper can be seen as a variation of the resource discovery problem in distributed networks [4], where nodes learn about other nodes in the network. However, there are important differences: (i) we are interested in learning about one specific resource as opposed to learning about all other known nodes, (ii) our network may be disconnected and (iii) in our case, each node has a limited-size cache, so at each instance, it knows about at most k other nodes.

A similar problem appears also in resource discovery in peer-to-peer (p2p) systems. In this case, a peer searches for a resource provided by some other peers. Flooding-based approaches, in which each peer contacts all peers in its neighbor have been proposed in this context as well. Gnutella [2] is an example of such an approach. [5] suggests the use of the Gnutella network to help agents locate infrastructure components. While there has been a lot of empirical studies (e.g. [9]) and some simulation-based analysis (e.g. [6]) of flooding and its variants for p2p systems, analytical results are lacking. Here, we analytically evaluate various alternatives of flooding-based approaches.

Besides flooding-based search, in p2p research, more sophisticated approaches (such as CAN [7], Chord [11], Past [8] and Tapestry [13]) build a distributed hash table to provide efficient search. With distributed hashing, each resource is associated with a key and each node (peer) is assigned a range of keys. For hashing to work, the network must be highly structured, in that resources should not be placed at random peers but at peers at specified locations.

Finally, flooding has also been used in ad-hoc routing (e.g. [3]). Here, the objective is to ensure that a message starting from a source node reaches its destination.

3. Search algorithms

We assume a multi-agent system with N nodes/agents, where each agent provides a number of resources. We assume that there are R different types of resources. To fulfill their goals, agents need to use resources provided by other agents. To use a resource, an agent must contact the agent that provides it. However, an agent does not know which agents provide which resources. Furthermore, it does not know which other agents participate in the system. A common approach is to introduce middle agents or directories that maintain information about which agents provide a resource. Thus to find a resource, an agent has first to contact the middle agent.

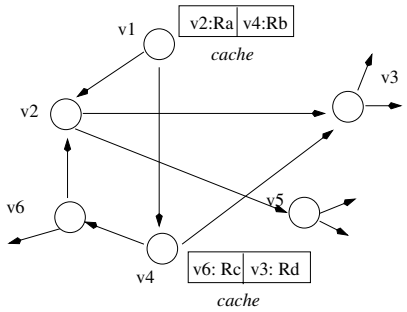

Figure 1. Part of a cache network, each agent v_i maintains in its cache the contact information for two resources ($k = 2$)

In this paper, we take a complementary approach. We assume that each agent can locally store part of what a middle agent knows. In particular, we assume that each agent has a private cache of size k. An agent stores in its cache information about k different resources, that is, for each of the k resources the contact information of one agent that provides it. The system is modeled as a directed graph $G(V, E)$, called the *cache network*. Each node corresponds to an agent along with its cache. There is an edge from node v to node u if and only if agent v has in its cache the contact

0-7695-1926-1/03/$17.00 (C) 2003 IEEE
Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS'03)

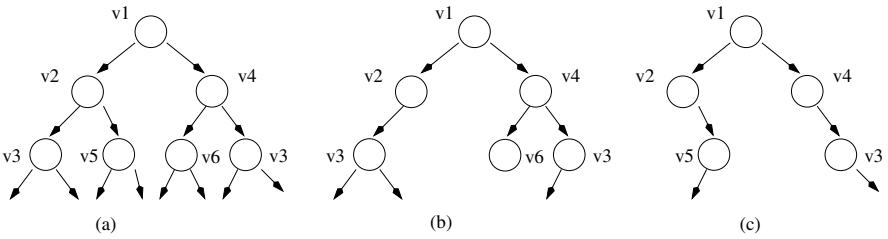

Figure 2. Searching the cache network of Figure 1: (a) flooding, (b) teeming, (c) random paths ($p = 2$)

information of agent u. There is no knowledge about the size of V or E. An example is shown in Fig. 1. An agent may provide two or more resources, thus the same agent may appear more than once in another agent's cache. Consequently, there may be less that k outgoing edges from a node, i.e. a node knows about *at most* k other agents.

We address the following problem: Given this cache network, how can an agent A, called the *inquiring agent*, that needs a particular type of resource x, find an agent that provides it? Agent A initially searches its own cache. If it finds the resource there, it extracts the corresponding contact information and the search ends. If resource x is not found in the local cache, A sends a message querying a *subset* of the agents found in its cache, that is, some of A's neighbors, which in turn propagate the message to a subset of their neighbors and so on.

Due to the possibility of non-termination, we limit the search to a maximum number of steps, t. In particular, the inquiring message contains a counter field initialized to t. Any intermediate node that receives the message first decrements the counter by 1. If the counter value is not 0, the agent proceeds as normal; while if the counter value is 0 the agent does not contact its neighbors and sends a positive (negative) response to the inquiring agent if x is found (not found) in its cache.

When the search ends, the inquiring agent A will either have the contact information for resource x or a set of negative answers. In the latter case, agent A assumes that the cache graph is *disconnected* i.e. that it cannot locate x through the cache graph. In this case, it will have to resort to other methods, e.g. use a middle agent. Note that disconnectedness may indeed occur because the network is dynamic: caches evolve over time.

In the following sections we propose three different strategies for choosing what subset of its neighbors each node contacts.

3.1. Flooding

In flooding, A contacts *all* its neighbors (i.e. all the agents listed in its cache), by sending an inquiring message, asking for information about resource x. Any agent that receives this message searches its own cache. If x is found in there, a reply containing the contact information is sent back to the inquiring agent. Otherwise, the intermediate agent contacts all of its own neighbors (agents in its cache), thus propagating the inquiring message. The scheme, in essence, broadcasts the inquiring message. It is not difficult to see that this scheme floods the network with messages. As the messages are sent from node to node, a "tree" is unfolded rooted at the inquiring agent (Fig. 2(a)). The term "tree" is not accurate in graph-theoretic terms since a node may be contacted by two or more other nodes but we will use it here as it helps to visualize the situation.

The flooding scheme has a number of disadvantages. One is the excessive number of messages that have to be transmitted, especially if t is not small. Another drawback is the way disconnectedness is determined. The inquiring agent has to wait for all possible answers before deciding that it cannot locate the resource. This introduces a number of problems. There is a large number of negative replies. Furthermore, since the network is not synchronized, messages propagate with unspecified delays. This means that the reply of one or more nodes at the tth level of the tree may take quite a long time. One solution is the use of timeout functions; at the end of the timeout period the inquiring agent A decides that the resource cannot be located, even if it has not received all answers.

3.2. Teeming

To reduce the number of messages, we propose a variation of flooding that we call *teeming*. At each step of teeming, if the resource is not found in the local cache of a node, the node propagates the inquiring message only to a *random subset* of its neighbors. We denote by ϕ the fixed probability of selecting a particular neighbor. In contrast with flooding, the search tree is not a k-ary one any more (Fig. 2(b)). A node in the search tree may have between 0 to k children, $k\phi$ being the average case. Flooding can be seen as a special case of teeming for which $\phi = 1$.

3.3. Random paths

Although, depending on ϕ, teeming can reduce the overall number of messages, it still suffers from the rest of flood-

ing's problems. One approach to eliminate these drawbacks is the following: each node contacts only one of its neighbors (randomly). The search space formed ends up being a single random path in the network of caches. This scheme propagates one single message along the path and the inquiring agent will be expecting one single answer.

In order to speed up the search, we generalize the aforementioned scheme, as follows: the root node (i.e. the inquiring agent A) constructs $p \geq 1$ random paths. If x is not in its cache, the inquiring agent A asks p out of its k neighbors (not just one of them). All the other (intermediate) nodes construct a simple path as above, by asking (randomly) exactly one of their neighbors. This way, we end up with p different paths unfolding concurrently (Fig. 2(c)). The algorithm, clearly, produces less messages than flooding or teeming but needs more steps to locate a resource.

4. Performance analysis

In this section, we analyze the performance of the proposed algorithms. In particular, we assume that the algorithms operate for a maximum of t steps and derive analytically three important performance measures:

- The probability, Q_t that the resource is found within the t steps. This probability determines the frequency with which an agent avoids using the other locating mechanisms available; Q_t should be as high as possible.
- The average number of steps, $\overline{S_t}$, needed for locating a resource (given that the resource is found), which naturally should be kept low.
- The average number of message transmissions, $\overline{M_t}$, occurring during the course of the algorithm. Efficient strategies should require as few messages as possible in order to not saturate the underlying network's resources (which, however, may lead to a higher number of steps).

4.1. Preliminaries

Here is a summary of the notation we will use:

R	number of resource types
k	cache size per agent/node
a	$= 1 - k/R$
$PC(j)$	probability that a particular resource is in at least one of j given caches
t	maximum allowable number of steps
s_i	prob. of locating a resource in exactly i steps
Q_t	prob. of locating a resource within t steps
$\overline{S_t}$	average # steps needed to locate a resource
$\overline{M_t}$	average # of message transmissions.

The network of caches is assumed to be in steady-state, all caches being full, meaning that each node knows of exactly k resources (along with their providers). The content of each cache is assumed to be completely random; in other words the cache's k known resources are a uniformly random subset of the R available resources.

Given a resource x, the probability that x is present in a particular cache is equal to:

$$PC(1) = P[x \in \text{cache}] = 1 - P[\text{every cache entry} \neq x].$$

The number of ways to choose k elements out of a set of R elements so that a particular element is not chosen is $\binom{R-1}{k}$. Since the k elements of the cache are chosen completely randomly, the last probability above is clearly equal to: $\binom{R-1}{k}/\binom{R}{k} = (R-k)/R$, which, gives $PC(1) = k/R$. In what follows, we let $a = 1-k/R$, so that $PC(1) = 1-a$.

If we are given j such caches, the probability that x is in at least one of them is:

$$PC(j) = 1 - (1 - PC(1))^j = 1 - a^j. \quad (1)$$

Now let us denote by s_i the probability of locating x at exactly the ith step of an algorithm. Then the probability of locating x in any step (up to a maximum of t steps) is simply given by:

$$Q_t = \sum_{i=0}^{t} s_i. \quad (2)$$

An important performance measure is the average number of steps needed to find a resource x. Given that a resource is located within t steps, the probability that we locate it at the ith step is given by s_i/Q_t, and the average number of steps is given by:

$$\overline{S_t} = \sum_{i=1}^{t} i\frac{s_i}{Q_t} = \frac{1}{Q_t}\sum_{i=1}^{t} i s_i. \quad (3)$$

4.2. Performance of flooding

In flooding, upon receiving the inquiring message, each node transmits it to all its neighbors (unless the required resource x is contained in its cache). As the algorithm progresses, a k-ary tree is unfolded rooted at the inquiring node. This search tree has (at most) k^i different nodes in the ith level, $i \geq 0$, which means that at the ith step of the algorithm there will be (at most) k^i different caches contacted.*

Suppose that we are searching at the ith level of this tree for a particular resource x. The probability that we find it there is approximately given by $\ell_i = PC(k^i)$ since in the

*Since an agent A may offer more than one resource, it may appear more than once in another node's cache. Also, there may exist more than one caches that know of A. Both those facts may limit the number of different nodes in the ith level of the tree to less than k^i.

0-7695-1926-1/03/$17.00 (C) 2003 IEEE
Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS'03)

ith level there are k^i caches. The approximation overestimates the probability since, as noted above, the number of different caches may be less than k^i. However, it simplifies the analysis and does not introduce significant error as shown by our simulation results.

The probability of locating x at exactly the ith step is given by:

$$s_i = \ell_i \prod_{j=0}^{i-1}(1-\ell_j), \tag{4}$$

that is, we locate it at the ith level and in none of the previous ones. Substituting yields:

$$s_i = PC(k^i)\prod_{j=0}^{i-1}(1-PC(k^j)) = \left(1-a^{k^i}\right)\prod_{j=0}^{i-1}a^{k^j},$$

or,

$$s_i^{(F)} = \left(1-a^{k^i}\right)a^{\frac{k^i-1}{k-1}}, \tag{5}$$

where F is used to denote the flooding scheme. Substituting in (2), and after some manipulation (see Appendix A), we obtain:

$$Q_t^{(F)} = 1 - a^{\frac{k^{t+1}-1}{k-1}} \tag{6}$$

The average number of steps needed to locate a resource is found by substituting (5) into (3). After some straightforward manipulations (given in Appendix A), the average number of steps is found to be:

$$\overline{S_t^{(F)}} = \frac{1}{Q_t^{(F)}}\left(a-(t+1)(1-Q_t^{(F)}) + \sum_{i=2}^{t+1} a^{\frac{k^i-1}{k-1}}\right) \tag{7}$$

We know of no closed-form formula for the sum in (7).

Let us now compute the number of messages in the flooding algorithm. If the resource is found in the inquiring node's cache there will be no message transmissions. Otherwise, there will be k transmissions to the k neighbors of the root, plus the transmissions internal to each of the k subtrees T rooted at those neighbors. Symbolically, we have:

$$\overline{M_t^{(F)}} = (1-PC(1))(k+km(t-1)),$$

where $m(t-1)$ are the transmissions occurring within a particular subtree T with $t-1$ levels. For such a subtree T, if x is found in its root node there will be 1 positive reply back to the inquiring node; otherwise, there will be k message transmissions to the children of the root plus the transmissions inside the k subtrees T' (with $t-2$ levels) rooted at the node's children. We are thus led to the following recursion:

$$\begin{aligned} m(t-1) &= PC(1) + (k+km(t-2))(1-PC(1)) \\ &= akm(t-2) + ak + 1 - a, \end{aligned}$$

with a boundary condition of $m(0)=1$ since the last node receiving the message (at the tth step) will always reply to the inquiring node whether it knows x or not. The solution to the above recursion is:

$$m(t-1) = (ak)^{t-1} + \frac{(ak)^{t-1}-1}{ak-1}(ak+1-a),$$

which gives:

$$\overline{M_t^{(F)}} = c^t + c + c(c+1-a)\frac{c^{t-1}-1}{c-1}, \quad c = ak. \tag{8}$$

Eq. (8) shows that (as anticipated) the flooding algorithm requires an exponential number of messages with respect to cache size (k).

4.3. Performance of teeming

In teeming, a node propagates the inquiring message to each of its neighbors with a fixed probability ϕ. If the requested resource x is not found, it is due to two facts. First, the inquiring node does not contain it in its cache (occurring with probability $1-PC(1)$). Second, none of the k "subtrees" unfolding from the inquiring node's neighbors replies with a positive answer. Such a subtree has $t-1$ levels; it sends an affirmative reply only if it asked by the inquiring node and indeed locates the requested resource. Thus, the probability of not finding x is given by the following recursion:

$$1 - Q_t^{(T)} = (1-PC(1))\left(1-\phi Q_{t-1}^{(T)}\right)^k,$$

which gives:

$$Q_t^{(T)} = 1 - a\left(1-\phi Q_{t-1}^{(T)}\right)^k. \tag{9}$$

where T is used to denote the teeming algorithm.

The average number of steps is found to be (see Appendix A):

$$\overline{S_t^{(T)}} = t - \frac{1}{Q_t^{(T)}}\sum_{i=0}^{t-1} Q_i^{(T)}. \tag{10}$$

The average number of messages is computed almost identically with the flooding case; the only difference is that since a node transfers the message to a particular child with probability ϕ, the average number of steps will be given by:

$$\overline{M_t^{(T)}} = (1-PC(1))(k\phi + k\phi m(t-1)),$$

where $m(t-1)$ is the transmissions occurring within a particular subtree with $t-1$ levels. The recursion (see Section 4.2) takes the form:

$$m(t-1) = PC(1) + (k\phi + k\phi m(t-2))(1-PC(1)),$$

0-7695-1926-1/03/$17.00 (C) 2003 IEEE
Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS'03)

which finally gives:

$$\overline{M_t^{(T)}} = c^t + c + c(c+1-a)\frac{c^{t-1}-1}{c-1}, \quad c = ak\phi. \quad (11)$$

Teeming also requires an exponential number of messages, which however grows slower than flooding's case; its rate is controlled by the probability ϕ.

4.4. Performance of the random paths algorithm

When using the random paths algorithm, the inquiring node transmits the message to $p \geq 1$ of its neighbors. Each neighbor then becomes the root of a randomly unfolding path. There is a chance that those p paths meet at some node(s), thus they may not always be disjoint. However, for simplification purposes we will assume that they are completely disjoint and thus statistically independent. This approximation introduces negligible error (especially if p is not large) as our experiments showed.

At each step i, $i > 0$, of the algorithm p different caches are contacted (one in each of the paths). The probability of finding resource x in those caches is $PC(p) = 1 - a^p$. Therefore, the probability of finding x after *exactly* i steps is equal to (analogously to (4)):

$$s_i^{(p)} = \begin{cases} 1-a & i = 0 \\ a(1-a^p)a^{p(i-1)} & i \geq 1. \end{cases} \quad (12)$$

Given t steps maximum, we can easily calculate the probability $Q_t^{(p)}$ that what we are looking for is found, using (2) and (12):

$$Q_t^{(p)} = 1 - a^{pt+1}. \quad (13)$$

Similarly, the average number of steps will be given by:

$$\overline{S_t^{(p)}} = \frac{a - (1+t-ta)(1-Q_t^{(p)})}{(1-a^p)Q_t^{(p)}}. \quad (14)$$

The derivation is given in Appendix A. Setting $p = 1$ the above formulas give the corresponding performance measures for the single-path algorithm.

Finally, the number of message transmissions can be calculated using arguments similar to the ones in Section 4.2. If x is not found at the inquiring node's cache, then there will be p message transmissions to p of its children, plus the message transmissions along each of the p paths:

$$\overline{M_t^{(p)}} = (1 - PC(1))(p + pm(t-1)),$$

where $m(t-1)$ is the transmissions occurring within a particular path P of $t-1$ nodes. For such a path P, if x is found in its root node there will be one positive reply back to the inquiring node; otherwise, there will be one message transmission to the next node of the path plus the transmissions inside the subpath P' (with $t-2$ nodes) rooted at the next node. We are thus led to the following recursion:

$$\begin{aligned} m(t-1) &= PC(1) + (1 + m(t-2))(1 - PC(1)) \\ &= am(t-2) + 1, \end{aligned}$$

where, as in Section 4.2, $m(0) = 1$ since the last node receiving the message will always reply to the inquiring node whether it knows x or not. The solution to the above recursion is easily found to be equal to $m(t-1) = (1-a^t)/(1-a)$ which gives:

$$\overline{M_t^{(p)}} = ap + ap\frac{1-a^t}{1-a}. \quad (15)$$

5. Performance comparison and simulation

The three performance measures are shown in Fig. 3 for all the proposed strategies. In the plots we have assumed cache sizes equal to 5% of the total number of resources R, which was taken equal to 200. The flooding and teeming algorithms depend on k (the cache size) while the random paths algorithm is only dependent on the ratio k/R. The graphs show the random paths strategy for p = 1, 2 and 4 paths. For the teeming algorithm, we chose $\phi = 1/\sqrt{k}$, that is, on the average $\sqrt{k}$ children receive the message each time. Larger values of ϕ will yield less steps but more message transmissions as is evident from Eqs. (10) and (11).

Flooding/teeming yield higher probabilities of locating the requested resource and within a smaller number of steps, as compared to random paths. However, the number of message transmissions is excessive. Teeming constitutes possibly the better trade off if the probability ϕ is chosen appropriately. The random paths strategy performs quite poor for very small values of p (e.g. 1 or 2). However, for 4 paths or more, and larger cache sizes, its performance seems the most balanced of all.

To validate the theoretical analysis, we developed simulators for each of the proposed strategies. The simulators initially construct a table mapping each of the R available resources to random agents (which will provide the resource). Next, the caches of all agents are filled with random k-element subsets of the available resources. After the initialization, simulation sessions take place with agent 0 issuing one location request for a (uniformly) random resource x each time. Obtained statistics include the number of messages, the path length and a flag denoting whether the resource was found or not. For each of the proposed algorithms, at least 1000 such sessions are performed and the accumulated results are averaged.

In Fig. 4 we provide sample simulation results (patterned lines) along with the theoretical ones (unpatterned lines) for two of the strategies: teeming (with $\phi = 1/\sqrt{k}$) and random

paths. The plots include the probability of not finding the required resource ($= 1 - Q_t$) and the average number of message transmissions ($= \overline{M_t}$). The number of resources used was $R = 200$ and the cache sizes varied from 1% to 30% of R.

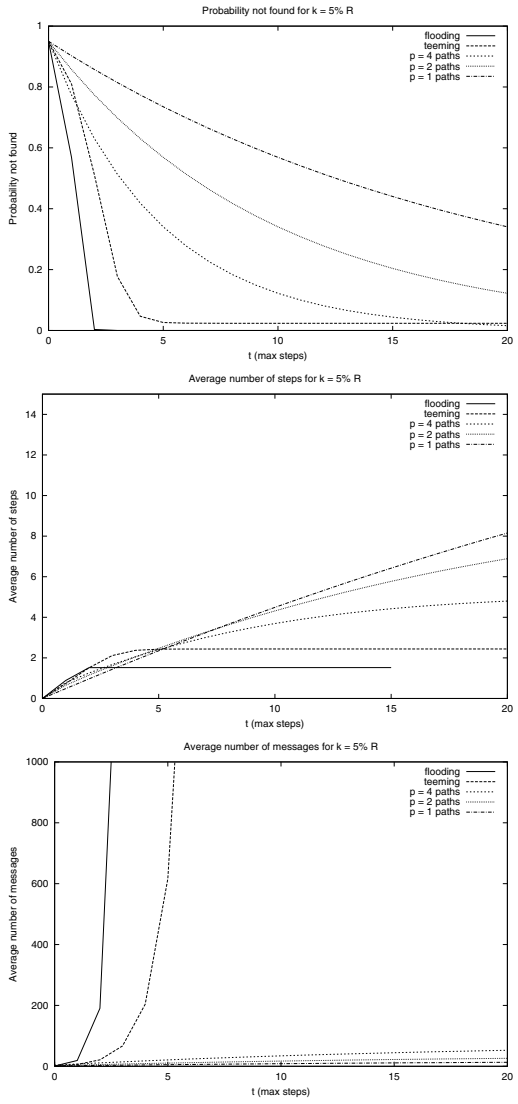

Figure 3. Comparison of the proposed algorithms: probability of not finding the resource ($= 1 - Q_t$), mean path length ($= \overline{S_t}$) and average number of message transmissions ($= \overline{M_t}$). The teeming algorithm uses $\phi = 1/\sqrt{k}$.

The plots show that our analysis matches the simulation results closely; the approximations in Sections 4.2 and 4.3 produce negligible error which only shows up in cases of very small cache sizes. A more detailed discussion, including similar results for flooding, can be found in [1].

6. Conclusions and future work

In this paper, we focused on resource location in multi-agent systems. We proposed and analytically estimated the performance of a number of variations of flooding-based search in such systems. We are currently working on making the system adaptive in many different ways: cached data may change over time, giving rise to cache replacement policies; agent locations may also do so, possibly invalidating cache entries.

References

[1] V. V. Dimakopoulos and E. Pitoura. Location Mechanisms for Distributed-Directory Open Agent Systems. Technical Report TR2002-02, Univ. of Ioannina, Dept. of Computer Science, Jan 2002.

[2] Gnutella website. http://gnutella.wego.com.

[3] Z. Haas, J. Y. Halpern, and L. Li. Gossip-Based Ad Hoc Routing. In *IEEE Proc. of INFOCOM 2002*, pages 1707–1716, 2002.

[4] M. Harchol-Balter, T. Leighton, and D. Lewin. Resource Discovery in Distributed Networks. In *PODC '99, Principles of Distributed Computing*, pages 229–337, 1999.

[5] B. Langley, M. Paolucci, and K. Sycara. Discovery of Infrastructure in Multi-Agent Systems. In *Agents 2001, Workshop on Infrastructure for Agents, MAS, and Scalable MAS*, 2001.

[6] Q. Lv, P. Cao, E. Cohen, K. Li, and S. Shenker. Search and Replication in Unstructured Peer-to-Peer Networks. In *Proc. ICS2002, 16th ACM Int'l Conf. on Supercomputing*, pages 84–95, 2002.

[7] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Schenker. A Scalable Content-Addressable Network. In *Proc. of ACM SIGCOMM*, pages 161–172, 2001.

[8] A. Rowstron and P. Druschel. Storage Management and Caching in PAST, a Large-scale, Persistent Peer-to-Peer Storage Utility. In *Proc. of SOSP 2001, 18th ACM Symp. on Operating System Priciples*, pages 188–201, 2001.

[9] S. Saroiu, P. K. Gummadi, and S. D. Gribble. A Measurement Study of Peer-to-Peer File Sharing Systems. In *Proc. MMCN '02, Multimedia Computing and Networking 2002*, 2002.

[10] O. Shehory. A scalable agent location mechanism. In *Proc. ATAL '99, 6th Int'l Workshop on Intelligent Agents, Agent Theories, Architectures, and Languages*, volume 1757 of *LNCS*, pages 162–172. Springer, 2000.

[11] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications. In *Proc. of ACM SIGCOMM*, pages 149–160, 2001.

[12] K. Sycara, M. Klusch, S. Widoff, and J. Lu. Dynamic Service Matchmaking Among Agents in Open Information Environments. *SIGMOD Record*, 28(1):47–53, March 1999.

[13] B. Y. Zhao, J. D. Kubiatowicz, and A. D. Joseph. Tapestry: An Infrastructure for Fault-tolerant Wide-area Location and Routing. Technical Report UCB/CSD-01-1141, U. C. Berkeley, April 2001.

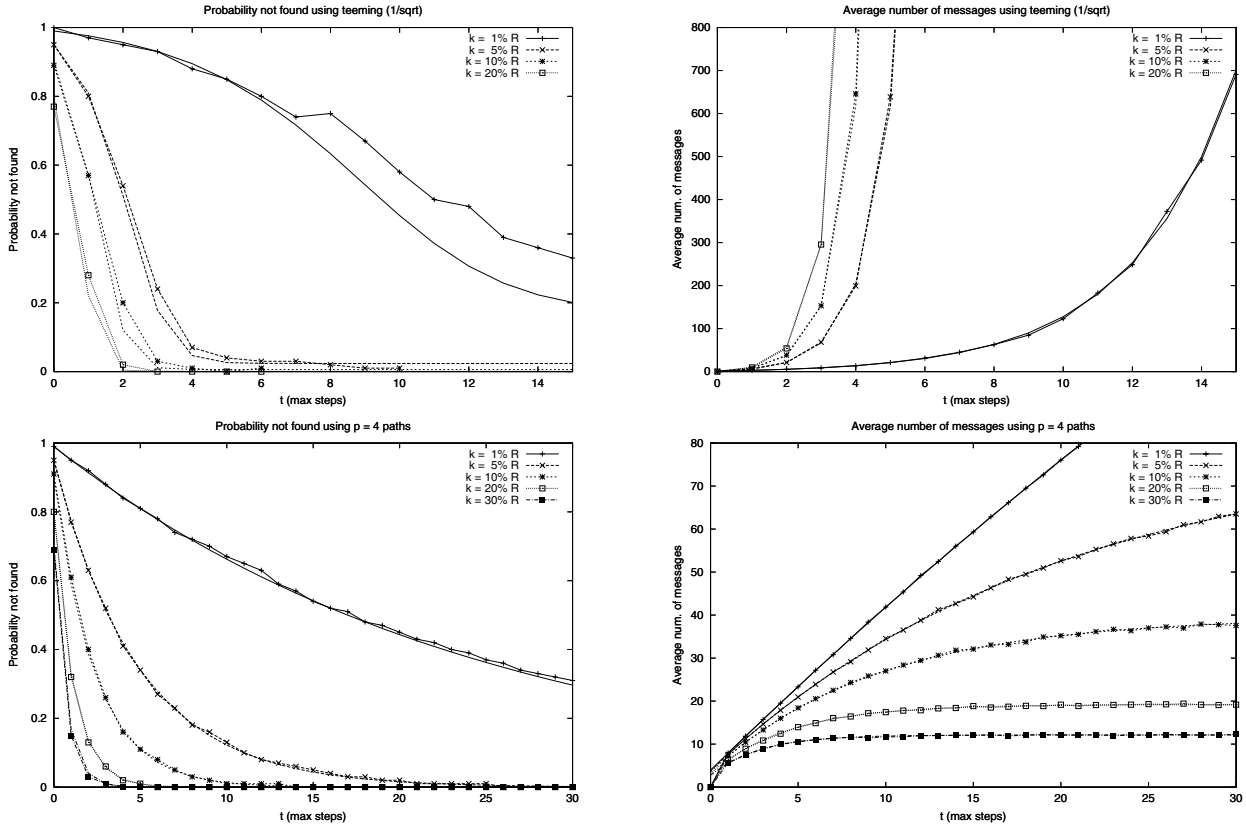

Figure 4. Simulation results (patterned lines) and analytical curves (unpatterned)

A. Formulas

Eq. (6) Let $b = a^{1/(k-1)}$. Then:

$$\begin{aligned} Q_t^{(F)} &= \sum_{i=0}^{t} s_i^{(F)} = \sum_{i=0}^{t} \left(1 - a^{k^i}\right) a^{\frac{k^i-1}{k-1}} \\ &= b^{-1} \left(\sum_{i=0}^{t} b^{k^i} - \sum_{i=0}^{t} b^{k^{i+1}} \right) = 1 - b^{k^{t+1}-1}. \end{aligned}$$

Eq. (7) Letting $b = a^{1/(k-1)}$ and working exactly as above, we obtain:

$$\begin{aligned} \overline{S_t^{(F)}} &= \frac{1}{Q_t} \sum_{i=1}^{t} i \left(1 - a^{k^i}\right) a^{\frac{k^i-1}{k-1}} \\ &= \frac{b^{-1}}{Q_t} \left(\sum_{i=1}^{t} i b^{k^i} - \sum_{i=1}^{t} i b^{k^{i+1}} \right) \\ &= \frac{b^{-1}}{Q_t} \left(b^k - (t+1) b^{k^{t+1}} + \sum_{i=2}^{t+1} b^{k^i} \right). \end{aligned}$$

Eq. (7) follows easily.

Eq. (10) We drop (T) from our notation for clarity. Using (3), we obtain:

$$\overline{S_t} = \frac{1}{Q_t} \left(\sum_{i=1}^{t-1} i s_i + t s_t \right) = \frac{1}{Q_t} (Q_{t-1} \overline{S_{t-1}} + t s_t).$$

From (2) it is seen that $s_t = Q_t - Q_{t-1}$. We thus obtain the following recursion on the number of steps:

$$\overline{S_t} = \frac{Q_{t-1}}{Q_t} \overline{S_{t-1}} - t \frac{Q_{t-1}}{Q_t} + t.$$

This recursion has the solution given in (10).

Eq. (14) Using (3) and (12),

$$\begin{aligned} \overline{S_t^{(p)}} &= \frac{1}{Q_t^{(p)}} \sum_{i=1}^{t} i a (1 - a^p) a^{p(i-1)} \\ &= \frac{a(1-a^p)}{Q_t^{(p)}} \sum_{i=1}^{t} i \left(a^p\right)^{i-1} \\ &= \frac{a(1-a^p)(1 - a^{p(t+1)} - (t+1) a^{pt} (1-a^p))}{Q_t^{(p)} (1-a^p)^2} \end{aligned}$$

which, using (13) and after some manipulation gives (14).