Trustworthy Service Caching: Cooperative Search in P2P Information Systems

Yathiraj B. Udupi
ybudupi@csc.ncsu.edu

Pınar Yolum
pyolum@csc.ncsu.edu

Munindar P. Singh
singh@ncsu.edu

Department of Computer Science
North Carolina State University
Raleigh, NC 27695-7535, USA

ABSTRACT

We are developing an approach for P2P information systems, where the peers are modeled as autonomous agents. Agents provide services or give referrals to one another to help find trustworthy services. We consider the important case of information services that can be cached. Agents request information services through high-level queries, not by describing specific objects as in caching in traditional distributed systems. Moreover, the agents autonomously decide whom to contact for a service, whom to provide a service or referral, whether to follow a referral, and whether to cache a service. Thus the information system itself evolves as agents learn about each other and the contents of the caches of the agents change.

We study the effect of caching on service location and on the information system itself. Our main results are that (1) even with a small cache, agents can locate services more easily; (2) since the agents that cache services can act like service providers, few initial service providers are enough to serve the information needs of the consumers; and (3) agents benefit from being neighbors with others who have similar interests.

1. INTRODUCTION

The purpose of information access is to fulfill a need. In conventional practical systems, information access has been mapped to database lookup and an emphasis is placed on looking for correct or relevant results. By contrast, in open settings, we look for authoritative sources that can provide correct and relevant results. The results may be available from multiple sources and hence are not necessarily unique. The above distinction corresponds to the difference between requesting a copy of Abraham Lincoln's Gettysburg Address (of which, say, there is one definitive version) and a commentary on the American Civil War. Or, more prosaically, a copy of the first draft WSDL standard versus an analysis of Web Services. Importantly, in an open environment, we cannot rely on traditional mechanisms using regulatory restrictions for ensuring the quality of the services obtained or the trustworthiness of the peers found over the network.

Thus, the need is to develop a decentralized approach wherein peers can find peers who are trustworthy and offer high quality services. Some of the peers would be service providers, possibly specializing in apparently arcane services and catering to a niche clientele. These peers could control and exploit private knowledge bases which were unreachable by traditional search engines. Other peers would learn about and use the above peers. If a peer finds the received services useful, it may (1) use it, (2) use it and refer others to it, or (3) use it and cache it so as to serve others from its cache. Caching aids the search for information since a peer that is looking for information can find it in a nearby cache rather than having the original provider generate it again from scratch. This way, by mutually helping each other to find trustworthy services, the peers can induce the formation of communities of common interest and practice.

A number of approaches have been developed for caching information in traditional distributed systems. Even those developed for P2P systems tend to take a traditional stance in that they view information access primarily as looking up specific objects rather than fulfilling information needs. However, a suitable flexible caching technique must satisfy two important criteria:

- *Peer autonomy*. A peer should have the freedom to choose with whom it interacts and how it carries out its interactions. It should also be able to change its neighbors to get the best results.
- *Peer heterogeneity*. Peers can be heterogeneous by having different policies and offering services distinct from all others. By accommodating heterogeneity, an information system also accommodates the fact that the peers can offer different levels of trustworthiness and be perceived by others as having different levels of trustworthiness.

Organization. Section 2 introduces key elements of our referrals-based model for service caching. Section 3 describes the evaluation scheme, discusses some of the important control variables used in evaluation, and gives our results. Section 4 concludes with a discussion of some relevant literature and of directions for further work.

2. MARMARA: FLEXIBLE CACHING TECHNIQUE

We provide some essential background on Marmara, a flexible caching technique based on *m*ulti-*a*gent *r*eferrals [Yolum and Singh, 2002]. Each agent offers varying levels of trustworthiness to others and is potentially interested in knowing if other agents are trustworthy (for its purposes). The agents can keep track of each other's

.

trustworthiness by flexibly giving and taking *referrals* and by learning each other and the available services. There is no guarantee about the quality of a service provided by any agent or the suitability of a referral it offers. Hence we do not assume that any agent should necessarily be trusted by others.

2.1 Referrals-Based Protocol

When an agent anticipates the need for a service, the agent begins to look for a trustworthy provider for the specified service. The agent queries some other agents from among its *neighbors*, which are a small subset of the agent's acquaintances. A queried agent may offer its principal to perform the specified service or may give referrals to agents of other principals. The querying agent may accept a service offer, if any, and may pursue referrals, if any. Each agent maintains models of its acquaintances, which describe their *expertise* (i.e., the quality of the services they provide), and *sociability* (i.e., the quality of the referrals they provide). Both of these elements are adapted based on service ratings from the agent's principal. The interests and expertise of the agents are represented as term vectors from the vector space model (VSM) [Salton and McGill, 1983], each term corresponding to a different domain, whereas the sociability is denoted with a scalar.

Algorithm 1 Ask-Query()

```
1: if (hasCachedAnswer) then
2:    Evaluate cached answer
3:    Determine if it has to stay in cache and then return
4: else
5:    while (more matching neighbors to ask) do
6:       Send query to matching agents
7:       Receive message
8:       if (message.type == referral) then
9:          Send query to referred agent
10:      else
11:         Add answer to answerset
12:      end if
13:   end while
14:   for i = 1 to |answerset| do
15:      Evaluate answer(i)
16:      Cache answers
17:      Update agent models
18:   end for
19: end if
```

Each agent is initialized with the same model for each neighbor; this model being rigged to encourage the agents to both query and generate referrals to their neighbors. An agent that is generating a query follows Algorithm 1. Since we do not have actual principals (i.e., humans) in the evaluation, the queries and the answers are generated by the system. More precisely, an agent generates a query by slightly perturbing its interest vector, which denotes that the agent asks a question similar to its interests. Next, the agent checks its own cache if it has a similar query cached already (line 1). If there is no answer found, the agent sends the query to a subset of its neighbors (line 6). The main factor here is to determine which of its neighbors would be likely to answer the query.

One way to choose the neighbors is through the capability metric, which measures how sufficient the expertise vector is for a given query vector [Yu and Singh, 2003; Singh et al., 2001]. Essentially, it resembles cosine similarity but it also takes into account the magnitude of the expertise vector. This means that expertise vectors with greater magnitude indicate higher capability for the given query vector. In Formula 1, Q ($\langle q_1 \ldots q_n \rangle$) refers to a query vector, E ($\langle e_1 \ldots e_n \rangle$) refers to an expertise vector and n is the number of dimensions these vectors have.

$$Q \otimes E = \frac{\sum_{t=1}^{n} (q_t e_t)}{\sqrt{n \sum_{t=1}^{n} {q_t}^2}} \quad (1)$$

If $Q \otimes E$ is greater than a threshold, for two agents A and B, then B is capable of answering A's query. Keeping the threshold high results in choosing only the most capable agents. Figure 1 shows an example network, where the nodes denote the agents and a dotted line from an agent A to an agent B means that B is a neighbor of A. The vectors marked with I are the actual interest vectors of the agents.

An agent that receives a query acts in accordance with Algorithm 2. An agent answers a question if its expertise matches a question. If the expertise matches the question, then the answer is the perturbed expertise vector of the agent. When an agent does not answer a question, it decides if it is interested in the query (line 5). If it is interested, then it applies the Ask-Query() method (Algorithm 1) to forward the query, find the answer itself, and return the answer it gets. Otherwise, it chooses some of its neighbors to refer.

Algorithm 2 Answer-Query()

```
1: if (hasCachedAnswer) then
2:    Return cached answer
3: else if (hasEnoughExpertise) then
4:    Generate answer
5: else if (interestedInQuery) then
6:    Ask-Query()
7:    Return answer
8: else
9:    Refer neighbors
10: end if
```

Back in Algorithm 1, if an agent receives a referral to another agent, it sends its query to the referred agent (line 9). After an agent receives an answer, it evaluates the answer by computing how much the answer matches the query (line 15). In a real application, user feedback would be involved here, but in the simulation, the agent itself evaluates the answer. Thus, implicitly, the agents with high expertise end up giving the correct answers. After the answers are evaluated, the agent caches only the useful (good) answers (line 16). When a good answer comes in, the modeled expertise of the answering agent and the sociability of the agents that helped locate the answerer (through referrals) are increased. Similarly, when a bad answer comes in, these values are decreased (line 17).

Each agent has equal number of neighbors. At certain intervals during the simulation, each agent can choose new neighbors from among its acquaintances. Usually the number of neighbors is fixed. Therefore, adding new neighbors means dropping some existing neighbors. When choosing neighbors, the acquaintances are ranked based on an equal weight of their modeled expertise and the sociability. The top ranked agents become the neighbors. In other words, it is best to interact with agents that can provide useful answers or useful referrals or both.

As the agents change neighbors, they find neighbors that best suit their interests. When searching a service, each agent follows referrals from its neighbors, who in turn give referrals to their neighbors, and so on. Since each agent follows referrals from the agents that it trusts, it is possible to locate trustworthy providers.

2.2 Cache Properties

In Marmara, a cache consists of a set of entries that contain the data item as well as some information about the quality or the appropriateness of the given data item. The query is modeled as a vector by modeling different domains as different dimensions of the vector. So an agent can specify the different keywords of search which can be represented by different values for the different dimensions of the vector. The answer is again a vector which represents the different domains of the query. The quality of the cache entry is modeled as to how appropriate it is to the owner of the cache based on its interest.

The peers insert items into their caches based on their own interests. The sizes of the caches are usually limited. Hence, a cache entry can be automatically replaced with a new entry based on the cache replacement policy in use. An agent can also delete an entry due to lack of interest in the item. Even after the owner of a cached entry deletes the original item, the peers that have a cached copy of the item are free to keep the item in their caches. An agent formulates queries by specifying a list of keywords which will be translated into a query vector. The keywords based query gives flexibility to the search, in that the source peer does not need to know the exact name or the unique identifier of the data item it is looking for.

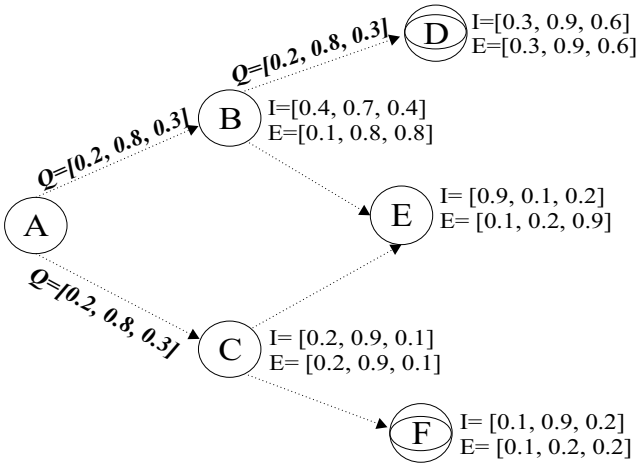

Figure 1: An example search

Let's walk through an example. Agent A is looking for an item that can be cast into a query vector of $[0.2, 0.8, 0.3]$. The peers D and F have the desired item in their caches. Following Algorithm 1, A starts the search by computing the query vector. Next, it decides which of its neighbors (B, C) to send the query to using (Equation 1). Even with a fairly high threshold, both B and C qualify. So, A sends its query to both B and C as indicated by the query vector on the links from A to B and A to C. Agents that receive the query follow Algorithm 2. Accordingly, when B receives the query, it checks its own cache to see if the item exists there. Not finding it in its own cache, B decides to send the query to its own neighbors, since its interest is similar to the query, i.e., it is curious about the results itself. After B obtains the desired item itself, it can forward it to A. Notice that among B's neighbors (D, E), D is the only one whose expertise vector matches the query, so B sends the query only to D. After B receives the item from D, it puts the item into its cache and forwards it to A. Meanwhile, since C is not interested in the query, it just sends a referral back to A.

Notice that a search can yield multiple results. Some of the peers may have more than one item whose content matches the query vector. For each item, we can specify how good a match the item is to the given query, which allows the returned items to be trivially ranked based on the strength of their apparent match.

3. EVALUATION

We evaluate Marmara using simulations over an agent platform, which enables the agents to query each other, obtain responses or referrals, and cache useful answers. We now briefly describe the simulation environment and some of the important parameters involved in testing.

Our simulation contains 400 agents and 4 domains in the vector model. There are 20 or 40 *experts* in the system (5 or 8 in each domain) who provide services—these are the agents who originate the information that others may cache. Experts give good answers in their domains of expertise. The remaining agents are service consumers, whose interests may span several domains. Service providers do not generate queries and therefore do not have any neighbors. The service consumers all have four randomly chosen neighbors. During the course of the simulation, after every three queries each agent has a chance of changing neighbors. The simulations are run for the duration of 10 neighbor changes.

3.1 Control Variables

We describe here some of the variable parameters used in our study and discuss how they affect the functioning of Marmara.

- **Threshold for forwarding query:** When an agent receives a query that it is also interested in, it can search for the answer itself. We compute the necessary level of interest by using the capability metric between the query and the interest vector (Algorithm 2, line 5). If this similarity is greater than the threshold for forwarding query, the agent performs the search itself. Setting high values for this threshold results in a situation when the agent will search only for queries that are highly similar to its own interests, whereas setting low values results in a situation where the cache would have answers to a broad area of interests. A low value for the threshold reduces its usefulness to the agent's own principal. Therefore, we set the threshold high so that most of the forwarded queries will closely match the agent's own interest.

- **Cache size:** This is the number of entries that can be stored by an agent in its cache. This factor plays a big role in the system. Having unlimited or a high cache size results in every good answer being cached in the system. This results in a majority of answers being directly retrieved from the agent's own cache, leading to a reduced exploration of the system. Limiting the cache size means that we will encounter cache overflows to be addressed according to a cache replacement policy (discussed next). Our experiments consider caches of size 15, 30, or unlimited.

- **Cache replacement policy:** When items in the cache are replaced, it is important to maintain cache quality, i.e., the quality of the answers to their corresponding queries stored in the cache. In our initial experiments, we experimented with a *random replacement policy*, which randomly replaces any entry in the cache whenever there is an overflow. When this policy is in effect, we observe that the cache quality of the system varies randomly. Hence we consider a more efficient policy which uses the quality of each entry before deciding the entry to be replaced. This *quality based replacement policy* replaces the entry with the least quality if the

quality of the entry to be added is higher than the least one. When this policy is used, we observe that with more replacements the overall quality of the cache improves.

- **Fidelity:** This threshold value is used in looking up the cache for a matching query. If the similarity between the current query and a previous query exceeds this threshold, then we consider that there is a cache hit. If there are multiple hits, the best cached answer is returned. A value of 1.0 for fidelity means an exact match between the queries is required, which would be rare, but a high fidelity improves the quality of the answers received. Hence we keep the value of fidelity in the range 0.9 to 1.0.

3.2 Results

We evaluate this caching technique by comparing the performance of the system with and without caching and by varying some of the control parameters discussed above. The evaluation depends on some measures of the performance and structure of the information system.

3.2.1 Success Rate

This measures the ratio of the queries for which at least one good answer is found. Let T be the total number of queries. Let G be the total number of queries for which at least one good answer is found. Then success rate is given by G/T. The success rate is calculated for each agent for one neighbor change of the simulation and then averaged over all service consumers. Since the service providers do not generate any queries, they are not factored into the calculation. An agent can find a good answer for its query, either from other agents or from its own cache. Initially, more answers are found through other agents. As agents fill their caches, more answers are retrieved from the caches, without the agents having to perform another search. This enables the agents to reach answers faster than before.

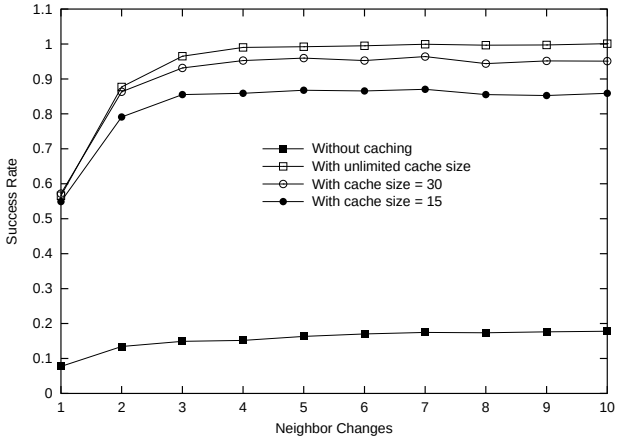

Figure 2: Effect of caching on success rate

As it is not practical to have an unlimited cache size, we limit the cache size and study the system. Figure 2 plots the success rate for different levels of caching when the population has 20 experts. The line labeled *Without caching* shows the success rate of the system when there is no caching. The remaining lines denote the success rates with different cache sizes. Even with a small cache size (size 15), the number of good answers received increases tremendously compared to the system without caching, as seen by the jump of success rate in Figure 2.

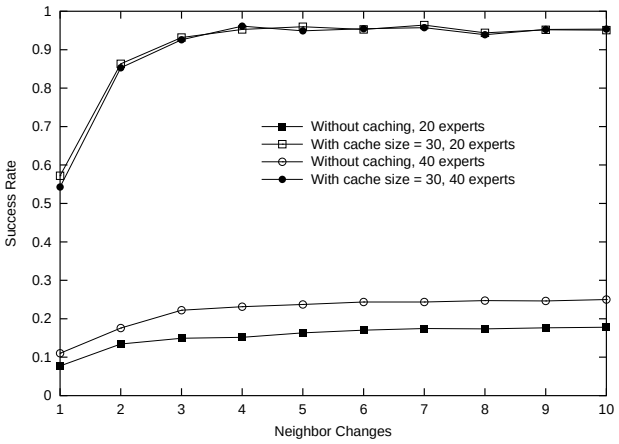

Figure 3: The effect of the number of experts on success rate

There is a big improvement in the success rate when we change from no caching to a cache size of 15. However, the success rate does not improve as much when the cache size is increased further. For example, the success rate improves only a little when the cache size is increased to 30 and only slightly more when the cache is allowed to be unlimited. This saturation can be explained by the fact that after a certain size, the number of good answers retrieved from the cache tends to be the same.

3.2.2 Effect of the Number of Experts on Success Rate

Figure 3 plots the success rates for the case without caching and with a caching size of 30, both with 20 experts and 40 experts in a system. In the systems with caching, the lines for the cases of 20 experts and 40 experts overlap. This means that the number of experts in the system does not really affect the success rate of a system with caching. On the other hand, there is an improvement in success rate for the case with 40 experts over 20 experts in a system without caching. We obtain similar results with unlimited caching and a cache size of 15. Intuitively, in a system with caching, there is a reduced dependence on finding an expert since after a while many service consumers have good answers in their caches. That is, any agent who returns an answer from its cache takes on the role of a quasi-expert, even though it may not truly be an expert.

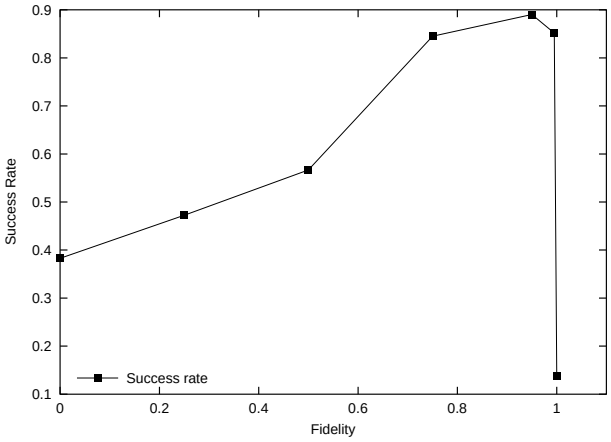

Figure 4: The effect of fidelity on success rate

3.2.3 *Effect of Fidelity on Success Rate*

The fidelity threshold plays a crucial role in determining the accuracy of the results returned from the cache. Figure 4 plots the success rate for different values of fidelity in a system with 20 experts and a cache size of 30. We observe that the success rate keeps deteriorating as we lower the value of this threshold. On the other hand, we cannot have a perfect value of 1.0 for this threshold as the queries will have to be matched perfectly to be retrieved from cache and this happens rarely if at all. There is a drastic drop in the success rate for 1.0 as it reduces to the case of no caching.

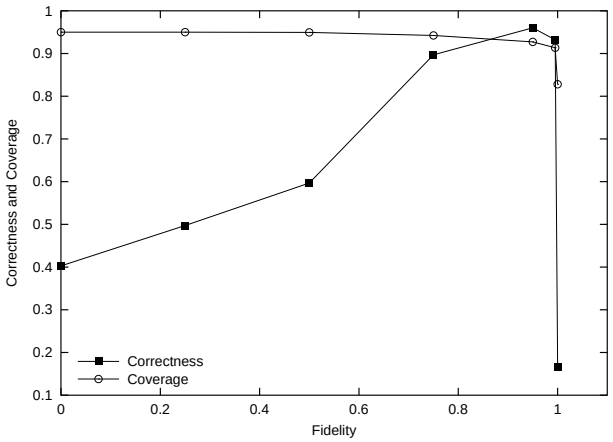

Figure 5: The effect of fidelity on coverage and correctness

3.2.4 *Coverage and Correctness*

The following two measures guide the behavior of success rate when fidelity is varied. Let G and T be as in the definition of success rate. Let A be the total number of queries for which an answer is found. *Coverage* is the ratio of the total number of queries answered to the total number of queries posed, i.e., A/T. *Correctness* is the ratio of the total number of queries with a good answer (i.e., answered correctly) to the total number of queries answered, i.e., G/A. The product of these two measures gives the success rate.

Figure 5 plots coverage and correctness against fidelity. We observe that coverage does not vary much as we increase fidelity, but there is a sudden decrease when fidelity approaches 1.0. Correctness increases steadily with fidelity, but this also drops suddenly for the fidelity value of 1.0. This can be explained with the following. When the fidelity is 1.0, the system is essentially reduced to the case of no caching, hence a low number of queries are answered as observed earlier in Figure 2. Comparing Figure 4 and Figure 5, we observe that the success rate increases gradually with the increase in correctness. It reaches a maximum when the correctness reaches a maximum, while the coverage does not vary much. When fidelity approaches 1.0, as both correctness and coverage see a sudden drop, the success rate also drops suddenly.

3.2.5 *Interest Similarity*

Interest similarity measures how similar the interests of two agents are. Again, we use a metric that operates on the interest vectors of the agents.

$$I_A \oplus I_B = \frac{e^{-\|I_A - I_B\|^2} - e^{-n}}{1 - e^{-n}} \qquad (2)$$

In Equation 2, I_A and I_B denote the interest vectors of two agents A and B, respectively. Here n is the length of the interest vectors. The metric captures the Euclidean distance between the two vectors, and normalizes it to return a value between 0 and 1.

For each agent, its similarity to its neighbors is calculated. Then, the average over all the agents is taken. Figure 6 plots the variation of interest similarity for different levels of caching in a system with 20 experts. (Our results for 40 experts are similar.) We observe that interest similarity increases with every neighbor change. This means that the agents tend to form neighbors with those agents who have similar interests. The intuitive explanation of this is that when two agents A and B have similar interests, they tend to generate similar queries. If A has cached an answer to a query that is later also generated by B, B can locate A as a good provider. Hence, A's cache provides an incentive for B to choose A as a neighbor.

We also observe that the values of interest similarity in the cases with caching are higher than the case without caching. We conclude that that caching has resulted in the agents with similar interests to group together. This situation could lead to some interesting observations such as the formation of communities in the network of agents.

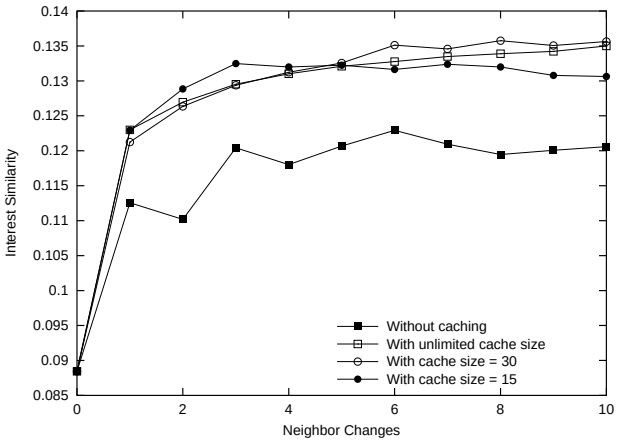

Figure 6: The effect of caching on interest similarity

3.2.6 *Local Quality*

The *local quality* viewed by an agent reflects the usefulness of the neighbors of the agent, given its interest and their expertise. That is, we estimate the likelihood of the neighbors themselves giving good answers to the questions and ignoring the other agents. We calculate local quality as obtained by an agent and then average it over all agents.

Figure 7 plots local quality for different levels of caching when the population has 20 experts. We see that there is a gradual decrease in the local quality of the network. When agents choose neighbors they take into account that good answers were received from an agent and cannot differentiate whether the answer was given from a cache or was generated afresh. Hence, an agent that serves items from its cache may be pointed by many even though it is not an expert itself, but has answers to the queries that the other poses.

4. CONCLUSION

Our referrals-based caching model that takes an adaptive, agent-based stance on peer-to-peer information systems. Referrals are a natural way for people to go about seeking information. One reason to believe that referral systems would be useful is that referrals

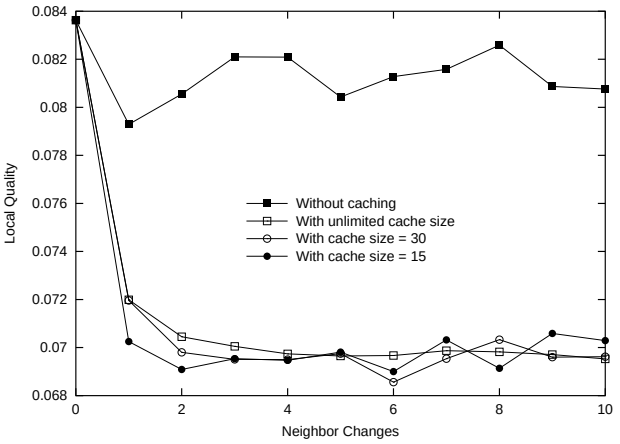

Figure 7: The effect of caching on local quality

capture the manner in which people normally help each other find trustworthy authorities. MINDS, based on the documents used by each user, was an early agent-based referral system [Bonnell et al., 1984]. Kautz *et al.* model social networks statically as graphs [1997]. They study properties of these graphs such as how the accuracy of a referral to a specified individual relates to the distance of the referrer from that individual. The above approaches do not incorporate caching.

Recently, several peer-to-peer network architectures have been proposed [Stoica et al., 2001; Ratnasamy et al., 2001] and using these as underlying layers, file-sharing application have been built [Dabrek et al., 2001; Rowstron and Druschel, 2001]. Rather than allowing peers to autonomously decide what to cache or index, these systems model the network as a distributed hash table that maps keys to peers. Each data item is replicated an equal number of times and the assignment is not decided by the peer. Further, as long as the peers in the system do not change, the search route is always the same (i.e., the agent has to contact the same agents).

This contrasts with Marmara, which allows agents to cache items that are of interest to them. The items that are of interest to more are cached at more peers. Further, based on the previous interactions with others, each agent chooses the agents it wants to interact with. The agents that provide better services, either by generating an answer or by serving an answer from their caches, are preferred over others. Hence, each agent adaptively decides on its neighbors.

Aberer and Despotovic develop a reputation-based trust model to manage the trustworthiness of agents in a peer-to-peer system [2001]. The agents that have been cheated file a complaint about the other party. The complaints are stored in a P-Grid structure [Aberer, 2001], a distributed structure that is maintained by multiple parties. An agent who is looking for a trustworthy agent pulls the complaint entries from the P-Grid and aggregates them. In our approach, the agents that provide evidence are rated. Hence, agents adapt by choosing neighbors that are most useful to them. Contrary to aggregating evidence from all possible agents, the agents that have not been useful in previous interactions are not considered in future interactions.

Marmara allows richer queries to be formulated, allowing trade-offs between search costs and item quality. In future work, we will perform experiments that exploit these trade-offs. We will also study other aspects that affect the performance of the system, such as cache replacement policies and local policies of the agents. A problem of particular interest is about revoking answers or letting cached answers expire. If the essential updates or revocations can propagate through the system, it would produce better responses without compromising the overall quality.

5. ACKNOWLEDGMENTS

This research was supported by the National Science Foundation under grant ITR-0081742. We thank the anonymous reviewers for helpful comments.

References

Karl Aberer. P-Grid: A self-organizing access structure for P2P information systems. In *Proceedings of Cooperative Information Systems (CoopIS)*, pages 179–194, 2001.

Karl Aberer and Zoran Despotovic. Managing trust in a peer-2-peer information system. In *Proceedings of the 10th International Conference on Information and Knowledge Management (CIKM)*, pages 310–317, 2001.

Ronald Bonnell, Michael Huhns, Larry Stephens, and Uttam Mukhopadhyay. MINDS: Multiple intelligent node document servers. In *Proceedings of the 1st IEEE International Conference on Office Automation*, pages 125–136, 1984.

Frank Dabrek, M. Frans Kaashoek, David Karger, Robert Morris, and Ion Stoica. Wide-area cooperative storage with CFS. In *Proceedings of the ACM Symposium on Operating System Principles (SOSP)*, pages 202–215, 2001.

Henry Kautz, Bart Selman, and Mehul Shah. ReferralWeb: Combining social networks and collaborative filtering. *Communications of the ACM*, 40(3):63–65, March 1997.

Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Shenker. A scalable content-addressable network. In *Proceedings of the ACM SIGCOMM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, pages 161–172, 2001.

Antony Rowstron and Peter Druschel. Storage management and caching in PAST, a large-scale, persistent peer-to-peer storage utility. In *Proceedings of the ACM Symposium on Operating System Principles (SOSP)*, pages 188–201, Banff, Canada, October 2001.

Gerard Salton and Michael J. McGill. *An Introduction to Modern Information Retrieval*. McGraw-Hill, New York, 1983.

Munindar P. Singh, Bin Yu, and Mahadevan Venkatraman. Community-based service location. *Communications of the ACM*, 44(4):49–54, April 2001.

Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for Internet applications. In *Proceedings of the ACM SIGCOMM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, pages 149–160. ACM, 2001.

Pınar Yolum and Munindar P. Singh. Flexible caching in peer-to-peer information systems. In *Proceedings of the Fourth International Bi-Conference Workshop on Agent-Oriented Information Systems (AOIS)*, pages 72–83, 2002.

Bin Yu and Munindar P. Singh. Searching social networks. In *Proceedings of the 2nd International Joint Conference on Autonomous Agents and MultiAgent Systems (AAMAS)*. ACM Press, July 2003. To appear.