Towards Bidirectional Distributed Matchmaking (Short Paper)

Victor Shafran*
Bar Ilan University
Department of Computer Science
Ramat Gan, Israel
shafrav@cs.biu.ac.il

Gal Kaminka, Sarit Kraus
Bar Ilan University
Department of Computer Science
Ramat Gan, Israel
{galk,sarit}@cs.biu.ac.il

Claudia V. Goldman
Samsung Telecom Research Israel
Yakum, Israel
c.goldman@samsung.com

ABSTRACT

Matchmaking is the process of introducing two or more agents to each other. Current matchmaking techniques are unidirectional and fail to address large-scale and highly dynamic systems with time constraints. We propose a new distributed technique which scales well, and still maintains relatively low matchmaking time and communication overhead. Our technique introduces very low storage and computational overhead to the agents. We suggest using a matching cache which can take advantage of the multidirectional nature of the matchmaking problem. We empirically evaluate the proposed technique on bilateral matchmaking and show that it outperforms the existing techniques.

Categories and Subject Descriptors

I.2.1 [**Distributed Artificial Intelligence**]: Multiagent systems

General Terms

Algorithms, Performance, Experimentation

Keywords

Matchmaking; Peer to Peer; Distributed; Evaluation

1. INTRODUCTION

Matchmaking is the process of introducing two or more entities to each other. In the context of multi agent systems (MAS), this process can be used in order to obtain service providers, create groups of shared interest, or form coalitions. Matchmaking is a prerequisite function for those and many other tasks in systems where entities do not have full information about the overall system configuration in advance, i.e., in open multi-agent systems that are usually very large in size.

The importance of the agent matchmaking problem rises from the growing popularity of open MAS. Open MAS can contain a large number of agents, possibly developed by different vendors. Moreover, agents can join and leave the system dynamically, which means the system configuration cannot be preprogrammed in advance. All this makes it necessary to provide mechanisms for online discovery of resources and service providers that are currently available in open MAS.

*This work was supported in part by ISF under grant #1685/07

Cite as: Towards Bidirectional Distributed Matchmaking (Short Paper), Victor Shafran, Gal Kaminka, Sarit Kraus and Claudia V. Goldman, *Proc. of 7th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2008)*, Padgham, Parkes, Müller and Parsons (eds.), May, 12-16., 2008, Estoril, Portugal, pp. XXX-XXX.

We are interested in developing protocols for matchmaking under time constraints, in environments where the number of agents is large, and each agent has limited computational (runtime and storage) capabilities. We assume a highly dynamic environment, agents continuously join and leave the system. We aim to develop matchmaking algorithms that will be fast and scalable, and in particular will minimize the amount of traffic generated in the system.

When solving the matchmaking problem, we focus on a distributed approach. In this case, each agent is capable of searching and announcing the activities it is looking for. There is no central point or special kind of agent that assists in finding matching agents; instead, agents share information among themselves and help each other resolve their matching requests. Such a distributed solution is very robust although the number of messages that are transferred among agents and the time it takes to find matching partners can be extremely large [3].

Previous works on distributed matchmaking [5, 6] used techniques that are unidirectional in nature: One agent searches, while the other passively waits to be contacted. This type of approach requires each passive agent to be continuously on-line. Moreover, it may increase the search time, because only one of the agents is searching for a match.

We are interested in highly dynamic environments where agents can join or leave the environment at any time. In addition, we are interested in systems where agents look for short-lived interactions, therefore we need a matching service that can find partners as quick as possible. For such types of applications, however, the passive search is not appropriate. Examples of interactions relevant to this paper include pickup, chess games or multi-party card games, given that a number of partners can be found quickly. Here, all matching agents are active in the search, simply because the user is not open to game proposals at all times. Thus the matchmaking process is not unidirectional but multi-directional.

To carry out efficient multi-directional matchmaking, we propose to use a *matching cache* stored with each participating agent. A matching cache is a structure, maintained by each one of the agents, which enables the agents to collect information about queries and perform matching on behalf of other agents. We aim to show that this makes matchmaking more efficient.

2. RELATED WORK

Many researchers e.g. [4] point out that open MAS and peer-to-peer (P2P) research areas are very similar to each other. As a result, the problem we study can be defined using P2P terminology, i.e., the resource location problem of some peer that is trying to search for a resource or service located somewhere in a P2P network.

Distributed techniques like Distributed Hash Tables (DHT) [7]

impose structure on the system based on the query key agent provides. The system is restructured each time an agent joins or leaves the network, and also each time an agent places or removes a request for a partner. Rather then pay this cost we focus on the unstructured systems,given the dynamic nature of the environment.

Shehory [6] introduced a distributed agent location mechanism in which each agent stores information about some predefined number of other agents in the network. While looking for some resource or service, an agent queries the agents it knows of for the required resources. The query propagates recursively through the system, until it is resolved. Under the assumptions of Shehory's work, it has been shown that the agent may know only a small portion of the MAS and still be able to resole queries efficiently in terms of time and communication costs. However, these results only suit MAS that can be modeled as lattice graphs. Also, it is assumed that the MAS changes adequately slow, to make the information which is sent over the network sufficiently reliable. The latter assumption does not hold in our environment.

Banaei-Kashaniy and Shahabi proposed modeling P2P resource-discovery using methods from statistical physics and percolation theory [3]. Since MAS or P2P systems can be very complex, these methods are important for analytical studies of the matchmaking problem. Using criticality-based analysis Banaei-Kashaniy and Shahabi reduced communication overhead introduced by the distributed search query which is sent over the network. They proposed a technique called probabilistic flooding, in which a message is sent with some predefined probability to each one of the sender's neighbors. The exact value of the probability for sending the message is determined analytically. However, they did not model matchmaking, in particular multidirectional matchmaking.

Additional research in matchmaking was performed by Ogston and Vassiliadis [5]. Their environment included simple agents with limited resources and they studied distributed techniques that would be suitable to solve consumer-provider problems. They proposed to use local search and investigated the system behavior with different numbers of agents and different numbers of tasks. Only local communication was allowed.

In contrast to these investigations, in this paper we propose a *multi-directional active* search process, in which all partners take active searching actions to accelerate the search process. This is done through the use of a matching-cache located at intermediate nodes, which allow agents to find each other's "trails" in the network, and thus discover matches. To the best of our knowledge such solution has not been studied previously.

3. THE APPROACH

Due to the size of the environment and its high dynamics, none of the agents has full knowledge of all other agents in the system. Instead, agents have an address book of a limited size where they store connection information to a small number of other agents. This situation can be modeled as a directed graph, where nodes are agents and edges correspond to links stored in the address book. In graph-theoretic terms, the graph is *not* fully-connected; actually, it is fairly sparse (though connected)[1].

To locate (i.e., to match) an agent that is not in the seeker's address book, the seeker can send a query to one or more of the agents in its address-book, and ask them to forward the query to their own peers. The cost of such a query is proportional to the number of messages the query creates. Once a match is found, the connection information is obtained, and the cost of communication is constant.

We want the agents to query the MAS for available matches, but the distributed nature of the environment makes it difficult to reduce the required matchmaking time together with a reduced number of messages. Previous works have proposed two basic techniques to reduce the number of messages in uni-directional search: The first one is called *teeming* [2]. Instead of forwarding a matchmaking query message to all the agents in the address book, teeming proposes to send messages to the neighboring agents with some predefined probability. Given the random nature of the address book graph, the proper probability value will ensure the message delivery to a bounded number of peers, and thus some bounds on the message number can be guaranteed. The second technique, standard in networking, limits the number of times each message is sent. This limitation is referred to as *Time To Live (TTL)*[3]. Both techniques can keep matchmaking time low. We assume that our environment supports both techniques and that proper values for both TTL and teeming parameters are given.

The key to our techniques is the use of a *matching cache* in intermediate agents. The matching cache stores incoming queries until a match is found (e.g., a matching query arrives at the same node), or until the time-limit expires (in which case the matchmaking is no longer relevant).

3.1 Bilateral Matchmaking

In bilateral matching, both agents actively query the MAS for a possible match. These two queries travel through the network, in essence executing a bi-directional distributed search process. In a simple case, a match is successfully resolved if one of the queries reaches a second matching agent.

To increase the likelihood (and to reduce the time) of a successful match, we introduce the matching cache data-structure. Each agent stores a FIFO cache of incoming queries of a predefined size and matches new queries against this cache. The cache stores agent connection information, together with the information necessary for matching (i.e., the activity type sought). Due to the distributed nature of the system, it is possible that information stored in the cache may be outdated. But if the cache size is chosen properly, and the data flow is fast, the portion of the outdated information will be relatively small.

To limit the number of messages in a network, and to prevent an infinite cycling of messages we use network hop counter with each message, called TTL (Time To Live). The TTL defines the number of hops (edge traversals) that a message can travel in a network until it expires. Every node that receives a message forwards it only if the TTL is greater than 0; when it carries out such forwarding, it reduces the TTL by one.

We assume that there are different kinds of activities in our environment. For example, there are agents looking for chess game players and agents looking for checkers players. Those activities are referred to as having different activity types. Agents that are willing to participate in an activity are referred to as partners. The agents pass simple query messages to each other, composed of the following data items: the initiator of the query,the activity deadline, activity type and TTL. Each agent A in the system performs Algorithm 1, running forever.

The cache stores received queries, including their associated activity types, partners sought, partners found, and deadlines. A separate process is assumed to maintain the cache, in terms of deadlines: When a message query in the cache reaches its deadline (i.e., how long the user is willing to wait), the process discards the query. In addition, the process is in charge of throwing out the oldest queries (even if still valid) when the cache is full, and new queries need to be stored.

3.2 Cache size

The cache stores only unmatched requests; when a matching request is satisfied it is removed from the cache. As a result, if an

Algorithm 1 Bilateral Matchmaking Algorithm.

1. For each incoming query from agent B do:
   (a) If agent A is interested in the query (i.e., it is a match), try to contact B to start the activity.
   (b) Otherwise, try to match the query to queries in A's cache.
      i. If a match is found, inform the matching agents.
      ii. Otherwise, store the query in the cache and forward the query using teeming.
2. For each new activity seeking a match for A do:
   (a) Create a query q.
   (b) If q is satisfied by a query from B on the local matching cache, then try to contact B to start the activity.
   (c) Otherwise, forward the query using teeming.

activity requires k participants, at most $k-1$ requests are stored in the cache. Assume that the number of different activities in the system is m. This provides us with an upper bound for the matching cache size: Matching Cache Size $= m\,(k-1)$

The above formula provides only an upper bound. Moreover it should be noticed that the cache can also reduce the performance of the system. This can happen since the cache stores knowledge about the whole system. According to our model, the system is distributed and highly dynamic, and, as a result the information stored in the cache can be incorrect. Assume that some, already invalid request is stored in the cache. When, a new valid request arrives it is matched with an invalid request and is not forwarded. We should minimize the effect of such an event.

4. EXPERIMENTS

We have conducted two comprehensive sets of experiments to evaluate the techniques presented in this report. The first set examines the effects of different network characteristics (e.g., connectivity, TTL levels, teeming probability) on performance, in order to establish baselines. The detailed results of these experiments are omitted from this article due to lack of space. The general trends that are derived from these experiments are summarized in Section 4.1 after a brief definition of all network parameters. The second set of experiments (summarized in Section 4.2) focuses on matchmaking performance when applying the matching cache for a given set of parameters in the context of bilateral matchmaking.

In all the experiments, we examine matchmaking performance along four independent measures:

- **Matching Success Rate.** This measures the percentage of matching attempts ending in a successful match. A higher value indicates improved performance.
- **Number of messages.** This measures the total number of query messages sent during an experiment. The lower the value the better.
- **Matchmaking Time.** This measures the time (in units of network hops) that took the agents to establish a successful match. Again, the lower the value the better. Note that this measure *only applies to successful matches*, and is thus biased toward such successes.

4.1 Experiment Setup

As stated previously, the network of agents is modeled as a directed graph. In order to evaluate the algorithms proposed earlier, different graphs (corresponding to different networks) were generated using the following procedure: First, a complete simple cycle, encompassing all N agents in the network, was created (essentially establishing an N-size ring topology) to ensure that the graph is connected. Then, we created each possible edge in the graph (i.e., an edge may connect any two agents), with a probability of p. For the purpose of the experiments in this section, we generated graphs with $N = 1200$, and we experimented with different p values.

We focus on bi-directional searches for matches of different types of activities. We tested systems with 1200 agents and 10 different types of activities. We simulated different workloads on the system by creating 2 types of scenarios. In the first one, 120 randomly chosen agents were looking for matches. In the second one, the number of such active agents was set to 600. We randomly generated 9 graphs and 3 scenarios and ran each scenario on each graph, creating 27 samples for each simulation. In order to make our simulation more realistic, we activate the agents searching for a match one by one during the first 120 (or 600) time steps.

The next network characteristic is the teeming probability [2]. This value defines the probability that a message will be sent to a given neighboring node . In our experiments we varied the teeming parameter from 0.1 to 0.7.

The last network characteristics are the TTL. The purpose of TTL is to limit the message's life in the network. We varied the TTL parameter from 1 to 20.

As the values of the TTL and the teeming probabilities grow, the overall success rate of the system and the total number of messages increases as well. Later, we compare the performance obtained with our distributed matchmaking technique based on the matching cache with the search technique that is based only on teeming and also on the TTL parameter. As we show later, the matching cache technique is qualitatively and quantitatively distinct from the use of TTL and increased teeming probabilities.

In the following experiments, the graph size N was set to 1200, the TTL value was set to 5, the visitor's cache was set to 0, and the scenario was set to matchmaking 600 agents. The edge probability p was 0.005, and the teeming probability t was 0.3. These values were set as default (unless explicitly stated differently) after having tested a large range of values and having obtained similar trends.

4.2 Bilateral Matchmaking using a Matching Cache

The set of experiments, described here, evaluated the effect of the matching cache size on the performance of our matchmaking algorithm. In these experiments, we set the matching cache size to 0,1,5 and 10.

We compare our distributed bilateral algorithm to the passive matchmaking (unilateral search) algorithm in which only one of the agents is sending out queries, and the other is passively awaiting a message to reach it. We want to show that Algorithm 1 with a matching cache set to 0 provides an upper bound for the passive (unilateral) search. We assume that both the bilateral (Algorithm 1) and unilateral search algorithms use the same network parameters (TTL, Teeming, Edge probability).

PROPOSITION 1. *Let T_b, M_b and S_b be the time, number of messages and success rate of Algorithm 1 that finds a match between any 2 agents assuming that the size of the matching cache is zero. Let T_u, M_u and S_u be the time, number of messages and success rate of two unilateral searches run in parallel to find the same match between any 2 agents in the same network. Then, $T_u \geq T_b$, $M_u \geq 2 \cdot M_b$ and $S_u \leq S_b$.*

PROOF. First, we need to show that given any successfully re-

solved matching task with a unilateral search, our algorithm run with a matching cache of size zero will also perform the same task successfully. A match is considered successful if and only if both agents looking for a match have found each other.

Assume first that only pairs of agents look for a match. That is some agent B is looking for some agent A with certain characteristics (and similarly A is looking for B). This match can be resolved successfully with a simple unidirectional algorithm when the query of agent A reaches agent B or the query of agent B reaches A. The same result will be obtained when running our bilateral algorithm with a matching cache set to zero. A successful match can only be found when either one of the agent's message reaches the other agent directly (no intermediate node can help when the cache is kept of size zero). In such case, the bilateral algorithm cannot incur a larger cost in terms of time and number of messages. Also, the rate of the successes is equal in both cases. □

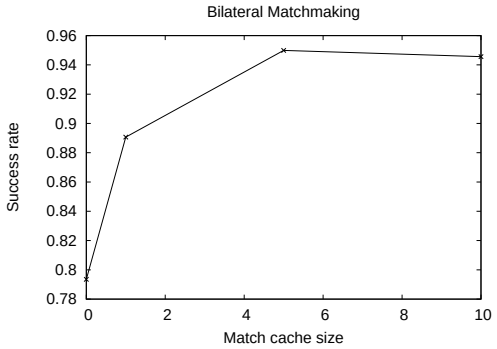

Figure 1: Success rate as a function of the matching cache size.

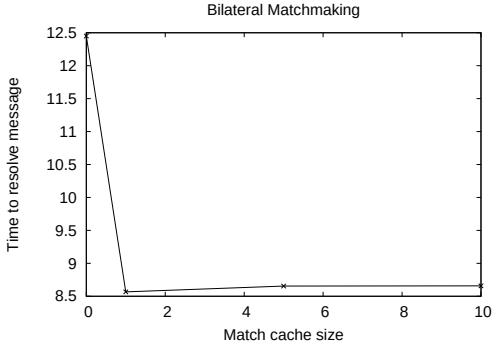

Figure 2: Time for matchmaking as a function of matching cache size.

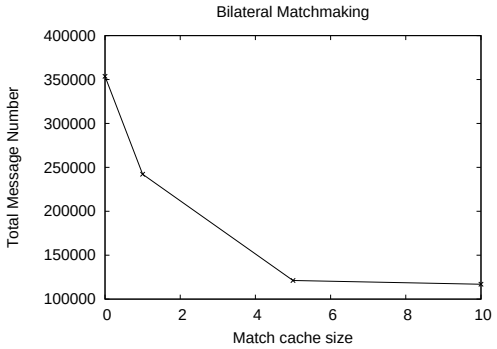

Figure 3: Number of messages as a function of the matching cache size.

Figures 1,2, and 3 show the matchmaking success rate, the matching time (for successful matches), and the number of messages, respectively, as a function of the matching cache size. The figures show that as the matching cache size increases, the success rate increases, and the number of messages and the time needed to find a match drops. An interesting observation is that even a small matching cache (size 1, in these experiments), is sufficient to provide a strong improvement in the matching time and success rate. Also, increasing the size of the matching cache further improves the other measures dramatically. Also note that the total number of messages that travels through the network decreases with larger caches. This happens because when agents receive a new query for which they have a match, such message is not forwarded. Following our approach, a match can be resolved by any of the agents in the network. Therefore, we also see a sharp decrease in successful matching time. Since we are interested in short-life interactions, this result is essential to finding matches as quickly as possible.

5. CONCLUSIONS

In this paper we presented an empirical study on the distributed matchmaking problem. In particular, we make use of the multi-directional nature of the matchmaking problem by introducing a matching cache which allows agents to find each other's "trail" in the network and thus discover matches more efficiently.

We have studied matchmaking in bilateral settings. We evaluated our techniques using a testbed which we developed and showed that we can solve the matchmaking process faster using our techniques. In addition, our algorithms reduce the total number of messages and improves the success rate of the overall system. We also demonstrate that achieving the same success rate using only the teeming or TTL techniques will require more time and more messages.

Our study is the first step towards implementing matching framework in highly dynamic networks comprised of agents looking for short-life interactions. In the future, we aim to extend our techniques by implementing incentives for cooperation and considering the topology of the agents' network and its effects on the cost of the matchmaking process.

6. REFERENCES

[1] T. T. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to algorithms*. MIT Press, 1990.

[2] V. V. Dimakopoulos and E. Pitoura. On the performance of flooding-based resource discovery. *IEEE Trans. Parallel Distrib. Syst.*, 17(11):1242–1252, 2006.

[3] F. B. Kashani and C. Shahabi. Criticality-based analysis and design of unstructured peer-to-peer networks as "complex systems". In *CCGRID*, pages 351–358. IEEE Computer Society, 2003.

[4] M. Koubarakis. Multi-agent systems and peer-to-peer computing: Methods, systems, and challenges. In M. Klusch, S. Ossowski, A. Omicini, and H. Laamanen, editors, *CIA*, volume 2782 of *Lecture Notes in Computer Science*, pages 46–61. Springer, 2003.

[5] E. Ogston and S. Vassiliadis. Matchmaking among minimal agents without a facilitator. In *Agents*, pages 608–615, 2001.

[6] O. Shehory. A scalable agent location mechanism. In N. R. Jennings and Y. Lespérance, editors, *ATAL*, volume 1757 of *Lecture Notes in Computer Science*, pages 162–172. Springer, 1999.

[7] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of the ACM SIGCOMM '01 Conference*, San Diego, California, August 2001.