International Symposium on Multimedia Information Processing

Dec. 1997

Asymmetric Binary Search of Mobile Agents §

Yao-Nan Lien*, Fuhan Liu*, and Wen-Shyen Chen**

* Department of Computer Science, National Chengchi University
** Institute of Computer Science, National Chung-Hsing University

Abstract

In a mobile computing environment that supports mobile agents, a client can send an agent to visit a sequence of servers in the network. Tracking the location of agents becomes a critical problem in managing a mobile agent service network. In our previous papers, we had developed a few strategies to search an agent in a sequential search list. The Extended Binary Search algorithm recursively probes the servers in the middle of the search list until the target agent is located. In this algorithm, the forward and backward search probes are treated quite differently. Therefore, the best range to probe is not necessarily at the middle of the search list. In this paper, we try to find the best range to probe. In our simulation experiments, we found that the best range to probe is between 0.29 and 0.4 of the search list starting from the head.

KEYWORD: *Mobile Computing, Mobile Agent.*

1. Introduction

1.1 Agent and Agent Mobility

A ubiquitous information service network provides information to the users any time anywhere [1,9]. Client users can access to the network through a wireless mobile communication network. In addition, the service network must provide some mechanisms allowing client users access to various information resources conveniently, which is referred to as *mobile computing* [1,2,3].

Because the distributed computing technology is not sufficiently mature to support mobile computing in such a scale, client users have to access network resources in a prescriptive fashion by interacting with individual servers one by one to accomplish a complicated task. However, in most mobile computing environments, the nature of communications is intermittent and the battery energy is limited so that it is very difficult and expensive to accomplish a complicated task. The *mobile agent* paradigm, which allows clients to interact with multiple servers in a dynamic fashion has been brought up to cope with this problem [1,2,3,5,8,10].

Simply speaking, a *mobile agent* is an electronic message that carries a computer program, whether procedural or declarative, which can be executed by the receiving servers on behalf of the originating client. The program in the message can also instruct a receiving server to forward automatically the message itself to another server, on which the program is executed continuously in a pipeline fashion. Good examples can be found in [3].

Since an agent may move continuely in a service network, the originating client may not be able to trace or control its operation directly. A service network must provide some mechanisms allowing its clients to trace and control these agents. This problem is referred to as the *agent mobility management*.

1.2 Open Service Network Architecture

To establish a large scale service network that can support mobile agents, we proposed an open service network architecture in [3], which separates service networks from transport networks. It also allows services of any scale and any quality to be introduced into the network easily. Readers are referred to [2,3,5] for details.

On the top of the open architecture mentioned above, we also proposed a hybrid operation mode in [2] that allows a service provider to offer both centralized and distributed operation modes to its subscribers. Subscribers can choose to use their own Internet facility, called *Home Base Node* (HBN), to share the OA&M (Operation, Administration, and Maintenance) duties. (Typical OA&M functionalities are billing, client location register, agent tracking and control, agent status holder, etc.) At their own expense, subscribers can also designate some OA&M duties to the centralized facility

§ This work is supported by NSC Grant (87-2213-E-004-002).

managed by the service providers. In this paper, we assume that a service network that supports mobile agents is established based on the proposed open architecture and managed using the proposed operation infrastructure.

1.3 Search of a Mobile Agent

After an agent is dispatched into a service network, the client or the network manager may need to know its current location in order to inquire its status, or to control its execution, etc. A simple way is to send another agent, called *search agent*, to search the original agent along the original path, or to broadcast a message to every server where the agent might have visited. There are some problems associated with these straightforward solutions:

1. Sending many messages over a wireless network may be too expensive.
2. A sequential search may take too much time.

Therefore, better ways to locate an agent are in need. In [4], we proposed and analyzed several search strategies, which will be described in the next section.

Since forward and backward probes are treated differently in various search algorithms such as Extended Binary Search algorithm, we found taht the best point (server) to probe is not right at the middle of the serach list. Probing a visited server can exclude some servers out of the search list, while probing an unvisited server can't. This paper proposes to adopt a probing range shorter than 0.5 of the search list.

The rest of this paper is organized as follows: Section 2 will review the search strategies proposed in [4,6]; Section 3 will discuss the proposed Asymmetric Binary Search strategies; Section 4 will illustrate our experiments; finally, Section 5 will discuss some complicated issues and future research directions.

2. Previous Work

The strategies proposed in [4] can be categorized into two types: blind and intelligent searches. The intelligent search strategies make use of prior knowledge about the execution of all tasks, while blind search strategies don't.

The following notations will be used in the following sections:

- $S = \{S_1, S_2, \ldots, S_n\}$: the set of distinct servers visited by an agent.
- T: the elapsed time since the target agent was originated.
- T_k: the time duration that the target agent stayed at server S_k. It is called the *service time* at server S_k.

In this paper, we assume that the target agent visits $\{S_1, S_2, \ldots, S_n\}$ sequentially and nonrecursively. Further, the time for the target agent to move from one server to another is considered nominal and is neglected.

Note that a recursive execution sequence can be easily expanded to a nonrecursive one by treating each visit (i.e. the target agent visits a server) a new server rather than a revisit. Further, if the time for an agent to move from one server to another is too long to be ignored, the agent migration can be modeled as a server so that our algorithms and analytical results can be applied without any modification.

2.1 Chase-from-holder Algorithms

In the hybrid operational infrastructure we proposed in [2], users are encouraged to use their own HBNs to participate in the management of their agents. One possible usage of HBNs is to store the current status of agents, including agents' locations. Thus, the current reported location of an agent can be obtained right in the status holder (i.g. HBN) of its originating user. If the accurate current location is needed, the search agent can visit the reported location first and proceed with a sequential search right from there if the target agent had passed that server. This type of algorithms is called the *Chase-from-holder algorithms*. In fact, it is a combination of tracking and searching methods.

Obviously, one major problem with the Chase-from-holder algorithms is the extra expense to update status periodically. The update cost has at least two major components: the induced network traffic and the resource consumption in the status holder. The system resources could be very expensive if the status holder is designated to the network management center. Thus, the trade-off between update cost and status availability must be carefully balanced. Depending on its status inquiry frequency, a client may choose to command an agent report its status either completely or selectively. The following search strategies are useful when the current location of an agent is not available in its status holder.

The *Basic Binary Search* algorithm (BBS) is similar to the binary search in searching a data object in a sorted list. The search agent probes the middle server in the search list and excludes half of the servers out of the search list at a time. The search is performed recursively until the target agent is found or the list is exhausted. In average, the number of probes required to locate the target agent is in the order of $log(n)$, where n is the number of servers in the search list. BBS might be a very good search strategy for blind searches. However, BBS may fail to locate the target agent if the target agent continues to move during the search. During the course of search, some unvisited servers may be excluded out of the search list after a server is probed. Unfortunately, the target agent may *slip through* the search window so that the servers it visits afterward are not included in the search list. As a result, the search agent will fail to locate the target agent. Although BBS is naive and faulty, it provides a basis to mutate into other search strategies as well as a baseline for performance comparison.

The *Extended Binary Search* (EBS) algorithm corrects the slip-through problem by not excluding any unvisited server out of the search list at the cost of demanding more search probes. Fortunately, the average number of probes required to locate the target agent is still in the order of $log(n)$, only with a larger coefficiency.

Above search strategies are classified as *blind search* because they do not use prior knowledge about the status of the target agent and servers.

2.3 Intelligent Search

If a client has a good estimation on the service time in each server, he/she may be able to guess the approximate location of an agent. By using this information in a search, we will be able to reduce the search time significantly. The term *intelligent search* here reflects the fact that these algorithms make use of service time statistics and thus, is non-blind.

We assume that an agent visits a set of servers, $S_1, S_2, \ldots, S_n$, in sequence and it stays at each server, says S_k, for a time duration of T_k. At any elapsed time T, the current location of the agent, S_c, can be determined by the following formula:

$$\sum_{i=1}^{c-1} T_i \le T < \sum_{i=1}^{c} T_i.$$

However, it is impractical to know in advance exactly how long the target agent will stay at each server. The service time in each server is most likely probabilistic and can be pre-estimated through either sample collection or experiments. Furthermore, for security and privacy reasons, it may be more practical to obtain statistics rather than detailed execution time records from servers. As a result, the location of the agent is probabilistic. We can calculate the location of the target agent with the highest probability, next highest, etc., and then search the target agent according to the order of probability. Assuming that the service time of each task is uniformly distributed, we derived a formula to calculate for each server the probability that a target agent might reside [4]. For simplicity, this probability is called the *residing probability*. In our proposed Intelligent Binary Search algorithm (IBS), the search list is presorted according to the calculated probabilities. Then, the search agent probes the servers in the search list sequentially.

Intelligent search algorithms are much better than blind search algorithms because they make use of prior knowledge about the service time of each task in each server. However, it requires the service time to be predictable. In other words, the algorithm assumes the target agent and the service network work normally without any exception. Thus, these algorithms have better be used under healthy operation conditions. They may not be appropriate to be used in handling exceptional cases.

Another problem associated with the intelligent search strategy is its quadratic computation time to calculate each individual residing probability. Even if the computation overhead can be compensated by using faster processors, the collection of the entire service time distribution from every server will still consume significant system and network resources. To reduce computation time, we proposed to use aggregated statistical properties to predict the location of a mobile agent [6]. Our analysis and experiments showed that, under a variety of circumstances, the mean service time is a reliable index to predict the location of a mobile agent. Readers are referred to [6] for details.

Nevertheless, both blind and intelligent searches have their own best circumstances to apply. It is immature to exclude any search strategy without further study. In this paper, we improve the binary searches by reducing the range to probe a server in the search list. In other words, each search probe does not probe the middle point of the search list as we have seen in all kinds of "binary searches". This will be illustrated in the following sections.

3. Asymmetric Binary Search

As described elsewhere [6], the Extended Binary Search Algorithm corrects the so called "slip-through" problem by keeping all unvisited servers in the search list. Only those servers that are known having been visited can be excluded out of the search list. In other words, after each probe, the search list will be kept unchanged if the target agent hasn't reach the currently probed server yet. Assume that the target agent may locate at any server with equal probability. By intuition, this algorithm prefers probing a visited server to a unvisited server (if the currently probed server doesn't have the target agent.)

As a consequence, the best server to probe in each attempt (a search probe) will not be right at the middle point. Instead, it will be at somewhere between the head and the middle point of the search list.

Another factor to make a search probe prefer a visited server to an unvisit server is the ability to use more up-to-date information in predicting the current location of the target agent. (The intelligent search algorithms presented in [6] make use of execution time statistics to predict the "current location" of the target agent, while other search strategies are "blind".) After probing a server that the target agent had visited recently, the search agent can make use the elapsed time since the target agent left the current server to predict its current location. Because this information is more up-to-date than the historical statistics, the prediction can be much more accurate. Therefore, it is more beneficial to probe a visited server than an unvisited server. This technique can be used in both blind and intelligent searches. However, it is more practical to be used in an intelligent search because of its capability of predicting the current location of the target agent based on the execution time statistics. This will be studied elsewhere [7]. In this paper, we will focus on the study of the best point to probe in a blind search, namely the Extended Binary Search.

3.1 The ABS Algorithm

The following is a basic asymmetric binary search algorithm:

Asymmetric Binary Search (ABS) Algorithm

```
Main {
  SrhSet = {S1,S2, . . . , Sn}
  c = n × θ
  ABSrh(target, SrhSet, Sc )    }
Procedure ABSrh (target, SrhSet, Sc )
 (1) if (SrhSet = empty) return (NOT_FOUND)
 (2) if (target in Sc) return (Sc);
 (3) elseif (target had visited Sc) {
 (4)  SrhSet = {Sc+1, . . . , Sn}
 (5)  c = c+(n−c+1) × θ    }
 (6) else  {
 (7)  h = index of head of SrhSet
 (8)  c = h + θ×(c−h+1) }
 (9) return ( ABSrh(target, SrhSet, Sc ))
```

In the above ABS algorithm, the magic number θ is a parameter between 0 to 1 denoting the ratio of the probing range to the entire search list. It will be 0.5 if the middle point of the list is to be probed. This algorithm is a mutation of the Extended Binary Search [4] by modifying the range of search probes.

3.2 The ABSS Algorithm

As we have mentioned before, if the target agent just left the currently probed server, the search agent can use the elapsed time information to predict the current location of the target agent. However, in this paper we assume the search agent does not have such a prediction capability in blind searches. Instead, the search agent simply switches to a simple search strategy, the sequential search. Major concerns are then (1) how to determine the proper condition to switch to the sequential search? and (2) how to minimize the penalty caused by estimation errors? These two issues will be discussed later. The following is a basic algorithm to realize this idea.

Asymmetric Binary Sequential Search (ABSS) Algorithm

```
Main {
  SrhSet = {S1,S2,     , Sn
```

```
Procedure ABSSrh (target, SrhSet, S_c )
 (1) if (SrhSet = empty) return (NOT_FOUND)
 (2) if (target in S_c) return (S_c);
 (3) elseif (target had visited S_c) {
 (4)  SrhSet = {S_{c+1}, . . . , S_n}
 (5)  if (target is near) SeqSrh(target,SrhSet)
 (6)  else {   c = c+(n−c+1) × θ )}
 (7) else  {
 (8)  h = index of head of SrhSet
 (9)  c = h + θ×(c−h+1) }
(11) return (ABSSrh(target, SrhSet, S_c))
```

Similar to ABS, the parameter θ in ABSS is a value to be researched. The procedure *SeqSrh* is a sequential search algorithm. (Details are ignored). The most critical decision is line (5) which is to decide the proper condition to switch to the sequential search.

It is not difficult to figure out that switching to the sequential search is more beneficial if the target agent is within the range of $\log_2$(length of the search list). The problem is that the search agent might not know exactly whether the target agent is within that range or not. One possible way is to make a prediction and accept the possible penalty caused by the estimation errors. Thus, it is a challenge to find a practical method that can minimize this penalty. Since the theoretical analysis is not trivial, we proceeded with a simulation study for the initial observation.

4. Experiments

We had proceeded with a series of experiments to verify our hypothesis. The number of servers we simulated are from 10 to 100 stepped by 10. For each simulation, we obtained the number of serach probes by varying probing ranges form 0.1 to 1 the size of the search list. (For abbreviation, we will simply use probing range for the ratio of probing range to the size of the search list.) As we can see from Fig. 1(a), Fig. 1(b), and Fig. 2, ABS is indeed better than a symmetric one, namely Extended Binary Search. Fig. 1(a) and Fig. 1(b) show the average number of serach probes when the length of the server list is 30 and 100 respectively. When the probing range is close to 0 or 1, the ABS degenerates into a simple forward or backward sequential search. Their best optimal probing ranges are clearly skewed away from 0.5. From Fig. 2 we can see that the best probing ranges are between 0.29 and 0.4 the size of the search list.

Because the best probing range varies with the size of the search list, we proceeded with another simulation to change the probing range based on the size of the current list during the course of each search. This is referred to as *dynamic probing range*. Intuitively, this may be closer to the upper bound of the search performance. However, it is very difficult to know exactly the optimal probing ranges as a function of the list size. Therefore, we chose to use the best probing ranges obtained from the first set of simulations. From Fig. 3, we can clearly see that the performance of the second set of simulations (dynamic probing range) is better than the first set (fixed probing range). Nevertheless, both of them are better than the conventional EBS.

5. Summary

In a mobile computing environment that supports mobile agents, a client can send an agent to visit a sequence of servers in the network. To track the location of agents becomes a critical problem in managing a mobile agent service network. In this research, we found that in a blind search, the best point to probe may not be the middle point of a search list. Due to the difference in treating forward and backward probes, the best range to probe is between 0.29 and 0.4 of the search list starting from the head. Further, the best probing range varies with the size of the search list. An asymmetric search algorithm that uses dynamic probing range according to the size of the current list will be better than a fixed one.

References

T. Imielinski and B. R. Badrinath, "Mobile Wireless Computing: Challenges in Data Management," *Communication of ACM*, August 1994.

2. Yao-Nan Lien, "Client and Agent Mobility Management," *Proc. of the Second Workshop on Mobile Computing*, Hsing-Chu, Taiwan, March 1996, pp. 141-152.

3. Yao-Nan Lien, "An Open Intelligent Messaging Network Infrastructure for Ubiquitous Information Service," *Proc. of the First Workshop on Mobile Computing*, Hsing-Chu, Taiwan, April 1995, pp. 2-9.

4. Yao-Nan Lien and Chun-Wu Leng, "On the Search of Mobile Agents," *Proc. of the IEEE Personal, Indoor, and Mobile Radio Conference*, Taiwan, Oct. 1996, pp. 703-707.

5. Yao-Nan Lien, et. al., "FlyingCloud: A Mobile Agent Service Network", *Proceedings of the International Conference on Distributed Systems, Software Engineering, and Database Systems*, Dec. 1996, pp. 177-183.

6. Yao-Nan Lien, Fuhan Liu, Chun-Wu Leng and Wen-Shyen Chen, "Intelligent Search of Mobile Agents", *Proceedings of the 1997 International Conference on Computer System Technology for Industrial Applications*, April, 1997, pp. 110-116.

7. Yao-Nan Lien, Fuhan Liu, Wen-Shyen Chen and Chun-Wu Leng, "Adaptive Search of Mobile Agents ", *Accepted to appear in the 1997 National Computer Symposium* .

8. P. Maes, "Agents that reduce work and information overload", *CACM*, July 1994, pp. 30-41.

9. M. Weiser, "The computer for the 21st century", *Scientific America*, 1992, pp. 94-104.

0. James E. White, "Telescript Technology: The Foundation for the Electronic Marketplace", General Magic, Inc.

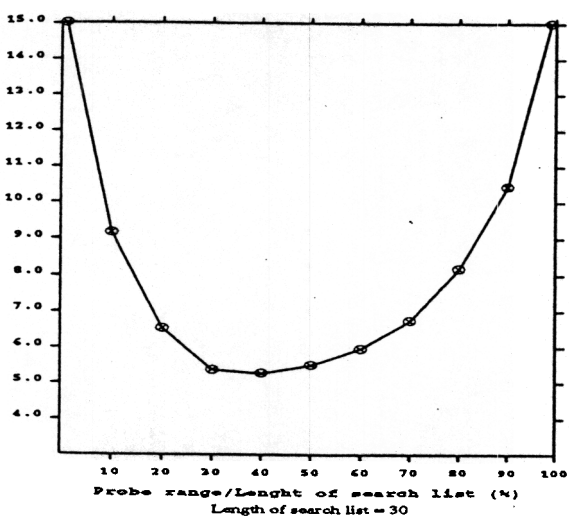

Figure 1(a). Simulation Results (I)

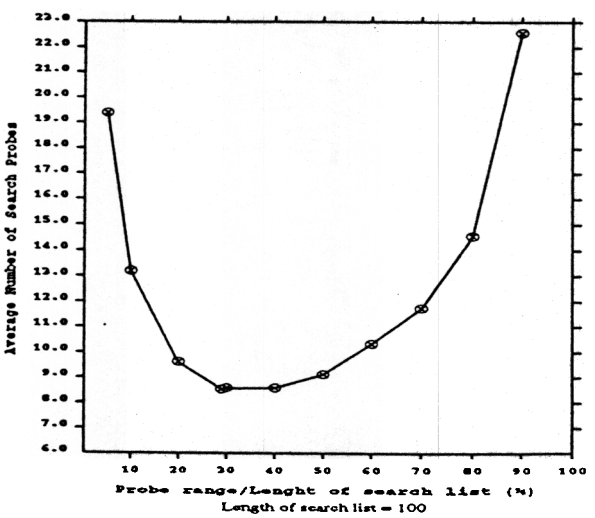

Figure 1(b). Simulation Results (II)

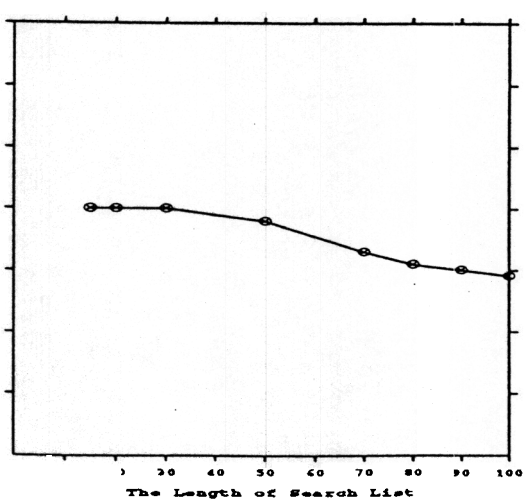

Figure 2. Optimal Probing Range

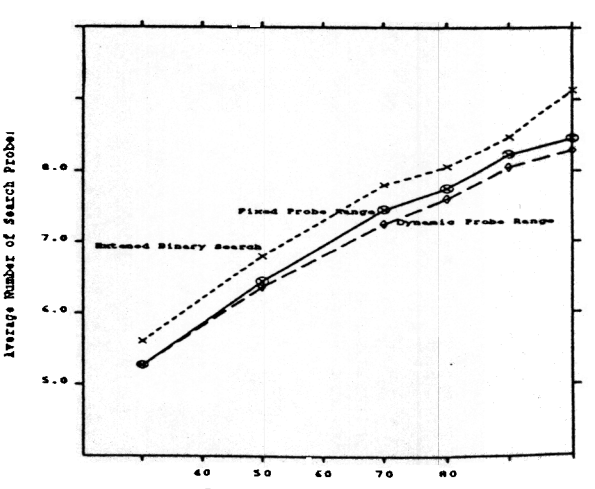

Figure 3. Comparison of Dynamic Probing Range to Fixed Probing Range