The Seventh IEEE International Symposium on
Personal, Indoor and Mobile Radio Communications
PIMRC'96
Taipei, Taiwan, ROC
October 15-18, 1996

On the Search of Mobile Agents *

Yao-Nan Lien and Chun-Wu R. Leng
National Chengchi University
Department of Computer Science
Taipei, Taiwan, R.O.C.

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. To track the location of agents becomes a critical problem in managing a mobile agent service network. This paper studies the agent search problem based on an open architecture proposed by Lien [3] and is currently being implemented in the National Chengchi University [4].

1. Introduction

1.1 Agent and Agent Mobility

The goal of a ubiquitous information service network is to provide information to the users anytime anywhere [6]. To accomplish this goal, a service network must be supported by a wireless communication network and be able to conveniently access to various information resources [8].

Due to the immaturity of distributed computing technology, clients have to access network resources in a prescriptive fashion by interacting with individual servers probe by probe to accomplish a complicated task. However, in most mobile computing environments, the nature of communications is interrnittent and the battery energy is limited. Thus, it is very difficult to accomplish a complicated task that requires its client to interact with multiple servers intensively. A non-traditional computing paradigm, the *intelligent messaging*, that allows clients to interact with multiple servers in a dynamic fashion has been brought up to cope with this problem [1,2,3,5,7].

Simply speaking, an *intelligent message* is an electronic message that carries a computer program, whether procedural or declarative, that 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. For simplicity, it is called a *mobile agent*, or *agent* in the rest of this paper. Good examples can be found in [3].

Since an agent may be traveling 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 messages. This problem is referred to as the *agent mobility management*.

* This work is partially supported by ITRI/CCL grant under MOEA Distributed Information Processing Program (86-EC-2-A-17-0204).

1.2 Mobile Agent Service Networks

1.2.1 Open Service Network Architecture

Traditional telecommunication networks such as PSTN and 800 Toll-Free service used to take considerable resources and long deployment duration to establish. One major resource drain in such networks is the OA&M (Operation, Administration, and Maintenance). It will be impractical to demand the comparable resources to support the OA&M functionalities in many perspective information services. The computing community will have to develop and deploy the demanded functionalities by themselves. All infrastructures and solutions must not require any change to the existing telecommunication network. To achieve this, we employ the open service network architecture proposed 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 [3] for details.

On top of the open architecture mentioned above, Lien proposed a hybrid operation mode in [2] that allows a service proivder to offer both centralized and distributed operation modes to its subscribers. Subscribers can choose to use their own Internet facility to share the OA&M functionalities. (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 functionalities to the centralized facility 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. After an agent is submitted 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 send 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. The path that an agent took may be non-deterministic such that it is difficult to trace.
3. A sequential search may take too much time.

Therefore, better ways to locate an agent are in need. This paper studies the agent search problem with several algorithms proposed and analyzed. The rest of this paper is organized as follows: Section 2 will study blind search where no prior knowledge about execution time in each server is available; while Section 3 will study non-blind search, or *intelligent search*. Section 4 discusses some complicated issues and future research directions.

2. Blind Search

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_0: the point of time when the target agent arrives the first server.
- 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 $\{S_1, S_2, \ldots, S_n\}$ in sequence and nonrecursively. Further, the time for the target agent to move from one server to another is considered nominal and is neglected.

2.1 Chase-from-holder Algorithms

The concept of *status holder* is especially useful in the search of agents as well as other OA&M functions. When using a Chase-from-holder algorithm, a search agent will visit the status holder of that client first to inquire the registered location of the target agent and then route to that server. If the target agent is found in that server, the search agent sends the result back to the client and terminates. Otherwise, it uses a trace algorithm to chase the original agent along the original route.

Although the use of status holders can greatly reduce the effort in searching an agent, it is not free. To maintain the most current agent status in its status holder requires the agent to report its status in each stage it moves. These messages might be very expensive, especially when the status holder is designated to the network management center. Thus, the trade-off between message cost and status availability must be carefully balanced. A client may choose to command an agent to report its status either completely or selectively. As a consequence, more search strategies that are not dependent on status holders are needed.

2.2 Status Holder Independent Search Algorithms

When a status holder is not available for search, or when the agent being searched loses contact with its status holder, a blind search will be inevitable.

2.2.1 Binary Search Algorithms

A search algorithm similar to the binary search in searching a data object in a sorted list may be good for blind search. It can only work in deterministic routing where the expected visiting servers as well as the visiting sequence are known in advance. Several algorithms will be shown in this section.

Basic Binary Search (BBS) Algorithm

```
Main {
  SrhSet = {S1,S2, . . . , Sn}
  BinSrh(Aid, SrhSet )
  }
Procedure BinSrh (target,SrhSet
 (1) if (SrhSet = empty)
 (2) then return (NOT_FOUND)

 (3) Sc = Middle Server in the SrhSet
 (4) if    (target in server Sc)
 (5) then  return (Sc);
 (6) elseif (target had visited Sc)
 (7) then  SrhSet = 2nd half of SrhSet
 (8) else  SrhSet = 1st half of SrhSet
 (9) return (BinSrh(target,SrhSet))
```

Line (7) commands the agent to search forward, called *forward probe*, and line (8) commands the agent to search backward, called *backward probe*. The Basic Binary Search Algorithm is directly derived from the standard binary search algorithm that is used to search a data object in a sorted list. It has a shortcoming in applying to the mobile agent search. Since a mobile agent may move during the search, it may slip through the search window such that the search agent fails to find it. For example, in the course of a search, a server S_k is probed before it is visited by the target agent, thus the search agent proceeds with a backward probe while the target agent continues to move; then the target agent visits S_k before it is found by the search agent. This is referred to as the *slip-through problem* in this paper. The following Extended Binary Search Algorithm can correct this problem.

Extended Binary Search (EBS) Algorithm

```
Main {
  SrhSet = {S1,S2, . . ,Sn}
  ExBinSrh(target, SrhSet

Procedure ExBinSrh (target, SrhSet)
 (1) if (SrhSet = empty)
 (2) then return (NOT_FOUND)

 (3) Sc = Middle Server in the SrhSet
 (4) if (target in Sc)
 (5) then  return (Sc);
 (6) elseif (target had visited Sc)
 (7) then  SrhSet = {Sc+1, . . . , Sn}
 (8) else  SrhSet = 1st half of SrhSet
 (9) return ( ExBinSrh(target, SrhSet))
```

The EBS differs from the BBS algorithm in the forward probe: when the search agent finds that the target agent had visited a server, it will search the entire search space succeeding the current probed server. Since the algorithm only excludes the known visited servers out of the search space, it can correct the slip-through problem in the BBS

algorithm.

2.2.2 Performance of Binary Search Algorithms

Traveling in a network, a mobile agent not only consumes network resources, it also consumes a significant overhead on each server it visits, such as authentication, request processing, agent migration, as well as bookkeeping overhead. Therefore, it is a nature objective to minimize the number of probes in searching an agent.

It is not difficult to figure out that it takes $O(\log | S |)$ probes for an agent using the BBS algorithm to find the target agent in the worst case, where $| S |$ is the size of the search space. The performance of the EBS algorithm will be analyzed in this section.

There are three different possible results when a search agent probes a server to see whether the target agent is in that server:

1. The target agent is right in the server: stop the search.
2. The target agent had already visited the server and been forwarded to another server: proceed with a forward probe.
3. The target agent hasn't visited the server yet: proceed with a backward probe.

Since the EBS excludes fewer candidates out of its search space than the BBS in forward probes, it requires more probes than BBS. In fact, the worse case happens when the target agent moves from server to server against EBS's search window. In other words, it always slips through the backward probe. In that unusual case, its performance is worse than a linear search. More complicated algorithms are needed to dear with such situation if it happens more frequently.

It is essential to assume that the target agent does not move during the search period for a meaningful comparison between BBS and EBS. Given a search space S, the EBS algorithm can exclude one quarter of the search space in every pair of (forward, backward) probes. After kth (forward, backward) probe pairs, the size of the search space is reduced to $\left\lfloor \frac{3}{4} \right\rfloor^k | S |$. Thus we can conclude that when the target agent does not move during the search, the EBS can find the target agent in $O(\log S |)$ probes (but, with a larger coefficient than BBS). Although it requires more probes than BBS algorithm, it is in the same order and does not have slip-through problem as BBS does.

Without a prior knowledge about the status of the target agent and servers, by intuition, the binary search is probably the best way (or close to the best) to search an agent. However, when there is a prior knowledge, other algorithms that can take this advantage will be better than binary search algorithms. It will be discussed in the following section

3. Intelligent Search

If a client has a good estimation on the service time in each probe, he/she may have a good guess on the current location of an agent. By using this information in a search, it may be able to reduce the search time significantly. The term *intelligent search* here reflects the fact that the class of search algorithms that makes use of execution time and thus, is non-blind.

Considering an agent visits a set of servers $S_1, S_2, \ldots, S_n$ in sequence starting from time T_0 and it stays at each server, says S_k, for a time duration of T_k. At any point of time t, the current location of the agent, S_c, can be determined by the following formula:

$$\Delta T_{c-1} > 0 > \Delta T_c,$$

where ΔT_k denotes $t - T_0 - \sum^{k} T_i$.

Unfortunately, in reality, it is impractical to know 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. As a result, the location of the agent is also probabilistic. To minimize the number of search probes, it is essential to calculate the location of the target agent with the highest probability, next highest, etc. so that a search agent can locate the target agent with the minimum number of probes. The calculation will be shown in next section.

3.1 Optimal Location Estimation

For simplicity, we assume $T_0 = 0$ and the service time in each server is uniformly distributed over a period of $[t_1 \cdots t_m]$. Variant distribution will be studied in the future. Further notations are defined as follows:

- $f_j(t_i)$: probability that the service time for the target agent at server S_j is exactly t_i, where $t_1 \le t_i \le t_m$, and $1 \le j \le n$.
- P_j^i: probability that an agent is currently running at server S_j after t time units from the beginning.

The probability function $f_j(x)$ indicates the time distribution that an agent spends in the server S_j. Therefore, $\sum^{t_m} f_j(x) = 1$, for each server S_j.

According to the probability calculation, the optimal searching strategy can be obtained by visiting servers according to the decreasing order of P_j^i values. (That means larger the value of P_j^i is, higher the priority of server S_j should be visited.) In other words, the expected number of probes after t time units from the beginning can be minimized as long as the search follows the rule. In that case, the problem remains to be solved is to calculate P_j^i.

The probability value of P_j^i will be formulated first by the following recursive equation:

$$P_j^i = f_j(t_1) P_{j-t_1}^{i-1} + f_j(t_2) P_{j-t_2}^{i-1} + \cdots + f_j(t_m) P_{j-t_m}^{i-1} \quad (1)$$

or

$$P_t^j = \sum_{\ell_1=1}^{m} f_j(t_{\ell_1}) P_{t-t_{\ell_1}}^{j-1} = \sum_{\ell_1=1}^{m} f_j(t_{\ell_1}) P_{(t-t_1+1)-\ell_1}^{j-1} \quad (2)$$

That means the probability of an agent currently running in server S_j is the summation of the agent spending t_{ℓ_1} time units in server S_j and all the previous servers totally contributing $t - t_{\ell_1}$ time units while $t_1 \le t_{\ell_1} \le t_m$. Noted that the distance between time t_{ℓ_1} and the time t_1 is $(\ell_1 - 1)$ time units. By continuing the same iteration, the probability P_t^j will be expressed as

$$P_t^j = \sum_{\ell_1=1}^{m} f_j(t_{\ell_1}) \sum_{\ell_2=1}^{m} f_j(t_{\ell_2} \quad \sum_{\ell_{j-1}=1}^{m} f_j(t_{\ell_{j-1}} \cdot Q \quad (3)$$

where $Q = P_{[t-(j-1)t_1+(j-1)]-(\ell_1+\ell_2+\cdots+\ell_{j-1})}^{1}$.

If the distribution function $f_j(t_\ell)$ is assumed uniform for every server S_j within the time period $[t_1, t_m]$, then Eq. (3) can be reorganized as

$$P_t^j = \frac{1}{m} \Bigg]^{j} \cdot \sum_{\ell_1=1}^{m} \sum_{\ell_2=}^{m} \quad \sum_{\ell_{j-1}=1}^{m} Q \quad (4)$$

while the function $f_j(t_\ell) = 1/(t_m - t_1) = 1/(m-1)$. According to the definition P_t^1 and the uniformity assumption, we may conclude that the value of P_t^1 is either $1/(m-1)$ for every time t between t_1 and t_m or zero otherwise. This conclusion translates Eq. (4) into a new problem that is to calculate how many tuples $(\ell_1, \ell_2, \ldots, \ell_{j-1})$ satisfy the inequality Eq. (5) when every ℓ_i is in the range of 1 to m:

$$t_1 \le [t-(j-1)t_1+(j-1)] - (\ell_1+\ell_2+ \quad +\ell_{j-1}) \le t_m \quad (5)$$

or

$$(t-m) - (t_1-1)j \le (\ell_1+\ell_2+\cdots+\ell_{j-1}) \le (t-1)-(t_1-1)j,$$

where $1 \le \ell_i \le m$, for all $\ell_i \in \{\ell_1, \ell_2, \ldots, \ell_{j-1}\}$

This new problem now can be easily solved by a nested *for* loop algorithm to pre-calculate the P_t^j value of every server S_j.

3.2 Intelligent Binary Search Algorithm

Intuitively, it seems a good strategy to search the target agent by visiting all servers according to the decreasing priority order. Unfortunately, this is not the case. Assuming S_c has the highest probability among the seach set S and we first visit S_c. If the target is not there and S_c had already been visited, we can ignore all preceeding servers regardless their probabilities. On the other hand, if S_c has not been visited yet, we can proceed with a backward probe regardless the probabilities of succeeding servers.

Based on the analysis shown above, we propose the following intelligent binary search that recalculate the probabilities in every forward probe.

Intelligent Binary Search (IBS) Algorithm

```
Global OriSrhSet
Main {
  OriSrhSet = PrioSort({S1,S2,     Sn}
  InBinSrh(target, OriSrhSet)

Procedure PriorSort(SrhSet) {
Sort SrhSet in decreasing order of Pt^j


Procedure InBinSrh (target, SrhSet
 (1) if (SrhSet = empty)
 (2) then return (NOT_FOUND)

 (3) Sc = 1st server in the SrhSet
 (4) if (target in Sc)
 (5) then  return (Sc);
 (6) elseif (target had visited Sc)
 (7) then SrhSet  = OriSrhSet - {S1, . . . , Sc}
 (8) else SrhSet -= {Sc,Sc+1, . . . , Sn}
 (9) return ( InBinSrh(target, SrhSet))
```

As we can see from the above algorithm, IBS is a simple extension of the EBS algorithm. In fact, the blind search can be viewed as a special case of intelligent search.

3.3 Comparison of Blind Search and Intelligent Search

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 exceptional case handling. For example, if the target agent is found far behind the schedule, the client may send out a search agent to check the status of the first agent. Apparently, the intelligent search is not appropriate to deal with this situation since all prior knowledge is known to be wrong already. In this case, blind search strategies may be a better choice. To determine when to use appropriate strategies requires further studies.

4. Issues and Future Research

4.1 Exact Search vs. Approximate Search

Because the execution of an agent continues to progress, the exact current location of an agent might have already changed when the client receives the acknowledge. To make the location of an agent remain unchanged after it is found, a freeze agent has to be submitted together with the search agent as explained in [2]. Otherwise, only an approximate location will be obtained.

4.2 Non-deterministic Execution

The search algorithms presented so far all assume that the agent to be searched traveled along a predetermined path. It may not be the case when the path it took depends on real time conditions and thus, is non-deterministic. In this case, the chase-from-holder algorithms are more appropriate. It is yet to be researched if the status holder is not available.

4.3 Lost Agent

An agent may get lost due to a failure in the agent itself, the server it resides, or the network it travels such that no search agent can find its status. Sometimes it may have no harm to just ignore the lost agent. However, it may be necessary to recover the lost agent in some critical situations such as business transactions. This problem is further complicated when concurrent execution is allowed because it is difficult to know whether all child agents are alive and all finish their assignments. Further research on this issue is needed.

4.4 Race Conditions

Since the target agent may be moving while a search agent is looking for it, a non-sequential search may fail to locate an agent. Therefore, it is essential for a search algorithm to avoid all possible race conditions. One of the main objectives of the prototyping experiment undergoing in the National Chengchi University is to analyze all possible race conditions [4].

4.5 Concurrent Search

Sequential search may take much longer time than what a client can tolerate. A client may submit more than one search agent into a network to reduce the search time. A search agent may also create child search agents by itself to cover all possible paths in non-deterministic situations such as an if-then-else decision. (The actual route an agent took may depend on real time conditions or on the information it obtained in the course of its execution.)

We are expecting many open issues under this situation. Typical issues are:

- How to converge forked agents?
- What to do if some child agents, or even the parent search agent itself gets lost?
- How to terminate all child agents?

5. Summary

A ubiquitous information service environment needs to offer various mobility allowing its client to access network through wireless communication networks [2]. To overcome the intermittent connection problem inherent in mobile environments, the intelligent messaging paradigm allows a user to request services by sending a mobile agent to cruise the network. Locating a mobile agent is one of the most important OA&M functions that are needed by a service network to maintain the desired level of QoS.

This paper proposes and analyzes several agent search strategies. All search algorithms can locate an agent in $O(log \mid S \mid)$ probes. Intelligent search algorithms reduce the number of search probes by using the knowledge of execution time of indiviual tasks; while blind search algorithms don't.

Although this paper lays down the fundamental agent search strategies, there are some exceptional cases that may make these strategies inefficient or even failed. Further researches are in need.

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, et. al., "FlyingCloud: A Mobile Agent Service Network", *Summitted to the International Conference on Distributed Systems, Software Engineering, and Database Systems.*
5. P. Maes, "Agents that reduce work and information overload", *CACM*, July 1994, pp. 30-41.
6. M. Weiser, "The computer for the 21st century", *Scientific America*, 1992, pp. 94-104.
7. James E. White, "Telescript Technology: The Foundation for the Electronic Marketplace", General Magic, Inc.
8. TIA/EIA IS-41, "Cellular Radio Telecommunications Intersystem Operations", *Telecommunications Industry Association*, Dec. 1991.