第四屆行動計算研討會
The 4th Mobile Computing Worksh
National Chiao Tung University, Taiwan, R.O.
March 25-26, 1998

Using Cut Vertices Search to Search Mobile Agents on a Non-Deterministic Path

Jang, Hung-Chin (jang@cs.nccu.edu.tw)
Lien, Yao-Nan (lien@cs.nccu.edu.tw)
Dept. of Computer Science, National Cheng-Chi University, Taipei, Taiwan, R.O.C.

Huang, Jyh-Shyan (frank@cs.nccu.edu.tw)
Dept. of Applied Mathematics, National Cheng-Chi University, Taipei, Taiwan, R.O.C.

Abstract

In a mobile computing environment that supports mobile agents, a client is able to send an agent to visit a sequence of servers in the network. Tracking the locations of agents becomes a critical problem in managing a mobile agent service network. In [8,9], Lien proposed several blind and intelligent search methods and studied their performances. But these methods are under an assumption that the path of the agent is deterministic. In [2], Jang proposed a search strategy called Non-Deterministic Binary Search (NDBS) which releases this assumption and allows locating an agent through a non-deterministic path. Using NDBS, we should know not only all the servers that possible to be visited by the agents but also the deterministic servers which have deterministic destinations; and the non-deterministisitic servers whose destinations are non-deterministic. In this paper, we propose a search strategy, Cut Vertices Search (CVS), when locating an agent, all we need to know are all those servers that possible to be visited by the agent and the server that sends the agent out.

Keywords : mobile agent, intelligent search, non-deterministic search, cut vertices set search

1. Introduction

1.1 Agent and Agent Mobility

The goal of an ubiquitous information service network is to provide information to the users anytime and anywhere [11]. Hence, a service network must be provided with a wireless communication network and be able to easily access to various information resources [12].

Due to the immaturity of distributed computing technology, clients have to access network resources in a prescriptive fashion by interacting with individual servers. However, in most mobile computing environments, the nature of communications is intermittent and the battery energy is limited. Thus, it becomes more difficult to accomplish a complicated task that requires intensive interactions among its client and multiple servers. A non-traditional computing paradigm, intelligent messaging, allows clients to interact with multiple servers in a dynamic fashion has been brought up to cope with this problem [1,4,5,10].

Simply speaking, an intelligent message is an electronic message carrying a computer program, either 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 automatically forward the message to another server, upon which the program is executed continuously in a pipeline fashion. We would use the term, mobile agent, as a substitution for intelligent message through the paper. Good examples are referred to [5].

Since an agent moves along a service network, the originating client may not be able to track or control its operation directly. A service network must provide some mechanisms allowing its clients to track and control these messages. This problem is referred to as agent mobility management.

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 OA&M (Operation, Administration, and Maintenance). It will be impractical to demand the comparable resources to support OA&M functionalities in many prospective information services. Thus, the computing community have to develop and deploy demanded functionalities by themselves. All infrastructures and solutions must not require any change to the existing telecommunication network. To achieve this, we employ an open service

network architecture [5] to separate service networks from transport networks. Under this architecture, it allows services of varied scales and qualities to be introduced into the network easily. Readers are referred to [5] for details.

On top of this open architecture, Lien proposed a hybrid operation model [4] that allows a service provider to offer both centralized and distributed operation models to its subscribers. Subscribers can choose to use their own Internet facility, e.g., Home Base Node (HBN), to share the OA&M functionalities. At their own expense, subscribers have alternatives to 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 based on the proposed open architecture and managed using that operation infrastructure.

1.2.2 Search a Mobile Agent

After an agent is submitted into a service network, the client or the network manager may want to know its current location for either inquiring its status or controlling its execution, etc. Obvious solutions are to send another agent, called search agent, to track the original agent along the original path, or to broadcast a message to all the servers that the agent may choose to go to. A couple of problems might arise with these solutions:

1. The path that an agent moves along might be non-deterministic and this is hard to track.

2. The cost of sending many messages over a wireless network is high.

3. A sequential search consumes lots of time.

Lien [8,9] proposed several search strategies to cope with these problems. However, all these strategies assume that the path of the target agent is deterministic. In [2], Jang introduced a Non-Deterministic Binary Search (NDBS) method to release this assumption and make it non-deterministic. Using NDBS, we should know not only all the servers that possible to be visited by the agents but also the deterministic servers which have deterministic destinations; and the non-deterministic servers whose destinations are non-deterministic. In this paper, we propose a search strategy, Cut Vertices Search (CVS), when locating an agent, all we need to know are all those servers that possible to be visited by the agents and the server that sends the agent out.

The rest of the paper is organized as follows:

In section 2, we review those search strategies given in [8,9] and NDBS. We proposed our strategy Cut Vertices Search in section 3. At last, we have conclusion and future research in section 4 and 5, respectively.

2. Previous Work

2.1 Deterministic Path vs. Non-Deterministic Path

2.1.1 Deterministic Path

A server is a deterministic server means that the next stop of the agent is deterministic when it visits the server. Similarly, a server is called a non-deterministic server if the agent's next stop is non-deterministic. If all the servers that the agent will visit are deterministic servers, then the possible path of it looks like a list.

2.1.2 Non-deterministic Path

If the possible path of the agent is non-deterministic then it looks like a spanning tree or a simple graph. If some of the servers that the agent will visit are non-deterministic servers and their visited sequence are determinable then the possible path of the agent looks like a spanning tree. Moreover, if the visited sequence of these severs are unknown, then the possible path looks like a simple graph.

2.1.2.1 Spanning Tree

If some of the servers that the agent will visit are non-deterministic servers then the possible path of the agent looks like a spanning tree.

For example, when the professor arrives the airport in New York, he needs a car to drive to the hotel. Hence, the agent rents a car and books a room for him according to his arriving time. Since, the name of hotel is not given. So, after the agent goes to the airport to confirm the arriving time and rents a car, it communicates with the information service agent in the airport to get information about the hotels. After interacting with all hotels, it chooses one of the hotels and books a room. Here, the path of the agent is non-deterministic since all hotels are possible candidates. This path is thus no more a sequential list of servers but a spanning tree of all alternative servers.

2.1.2.2 Simple Graph

If some of the servers will be visited are non-deterministic server and the visited sequence is unknown,

then the possible path can not be represented by a spanning tree but a simple graph.

For example, the agent is given a keyword and asks it to find all the papers that contain the keyword. If the addresses of digital libraries are given, the possible path of the agent looks like the simple graph. Because, there is not any absolute visited sequence that the agent has to follow to visit these servers.

2.2 Search Strategies of Deterministic Path

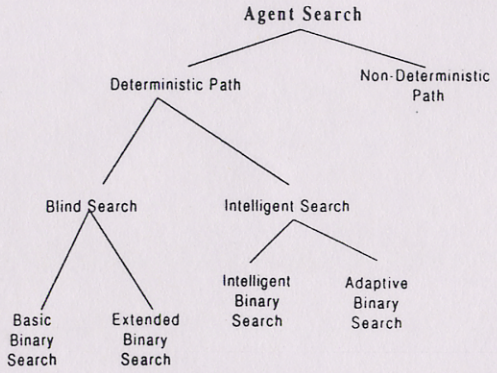

Fig. 1 Classification of agent search methods of previous work.

The searching strategies proposed in [8,9] can be classified into two groups: blind searches and intelligent searches. The intelligent search strategy makes use of prior knowledge about the execution of all tasks, while blind search strategy doesn't. Fig. 1 shows all the agent search methods have been done in our previous work.

2.3 NDBS

Jang [2] proposed Non-Deterministic Binary Search (NDBS) to search target agents moving along non-deterministic paths. It takes two steps to complete this search.

Step1: Search agent tracks the first non-deterministic vertex (*FN* vertex) from the root of the spanning tree and comes to the following two alternatives:

Case 1 : If the target agent has visited the *FN* vertex, we will know which child vertex is the next to be visited from the information recorded in server. Then, we remove all the ancestor vertices of this visited vertex and all branches other than the descendant branch that contains the next stop of this vertex. Go to step1.

Case2 : If the target agent hasn't visited the *FN* vertex then we know that the agent stays in one of the vertices between the root and the *FN* vertex. We then remove all child vertices of the *FN* vertex. Since all the vertices between the root and the *FN* vertex are deterministic, the tree becomes a list of vertices.

Step2 : Using either one of BBS, EBS or ABS to find the target agent. To prevent the target agent from passing through the end vertex of the list before the search agent finds it, we leave the search agent at the end vertex waiting for target agent and fork a child agent using either IBS or ABS to continue the search. This induces two alternatives:

Case 1 : The child agent finds the target agent and reports the location of the target agent to the search agent and terminates the search. The search agent then reports the result and terminates the whole search process.

Case 2 : The search agent finds the target agent. It means that the target agent arrives at the end vertex before the child agent finds it. The search agent then reports the location of the target agent, informs the child agent of "end of search" and terminates the whole search process.

3. Cut Vertices Search (CVS)

The path that an agent moves along may be either deterministic or non-deterministic. If we know all the servers to be visited by the agents are non-deterministic or deterministic exactly, the path can be represented by a tree. Just like the tree spanned as a non-deterministic path in NDBS. If we could not make sure whether the servers to be visited are deterministic or non-deterministic, we can use a graph to represent the whole paths of mobile agents.

A graph $G = (V,E)$ consists of a set of *vertices*, $V = \{v_1,v_2,v_3,\ldots\}$, and a set of *edges*, $E = \{e_1,e_2,e_3,\ldots\}$. Each edge element, e_k, is represented by an unordered pair (v_i, v_j) of vertices, and v_i and v_j are called the *end vertices* of e_k. The number of vertices G has is called the *order* of G, denoted by $|G|$. Vertex v_i represents the server with index i, possible to be visited by the target agent, and edge (v_i,v_j) represents the path between server i and server j, where 1

$\leqq i, j \leqq |G|$.

3.1 Introduction to Cut Vertices

In a connected graph G, we define *cut vertices set*, K, as a set of vertices whose removal from G leaves the graph disconnected. The *vertex connectivity*, $\kappa(G)$, is defined as the minimum number of vertices whose removal from G leaves the graph disconnected. For example, let $K = \{k_1,k_2,...,k_{\kappa(G)}\}$, the removals of $k_1,k_2,...,k_{\kappa(G)}$ leaves the graph, G-K, disconnected. The *order* of a graph G , $|G|$, is defined to be the number of vertices in G . The *degree* of a vertex, $d(v)$, is defined to be the number of edges incident on a vertex, v. The *minimum degree* is denoted as $\delta(G)$ and it is known that $\delta(G) \geqq \kappa(G)$.

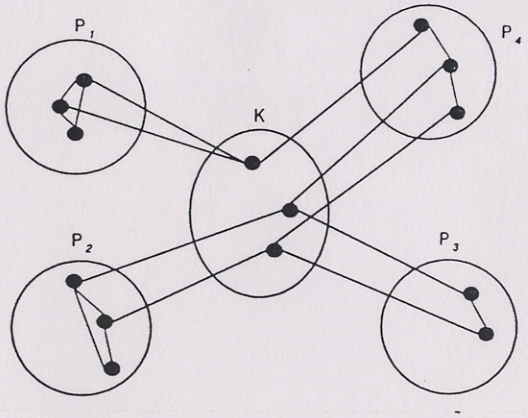

Fig. 2 Graph $G = P_1+P_2+P_3+P_4+K$

Assuming that the graph, G-$K = P_1+P_2+\ldots+P_t$, where $P_1, P_2,\ldots, P_t$ are maximal connected subgraphs of G-K and there exist no paths between any two of the subgraphs. As shown in Fig. 2, $G = P_1+P_2+P_3+P_4+K$, if we remove K from G, and the components of the graph G-K, P_1, P_2, P_3, P_4, are connected subgraphs of G-K and there exist no paths between any two of P_1, P_2, P_3, P_4.

Corollary : Given a graph $G(V,E)$, K is the cut vertices set of G and G-$K = P_1+P_2+\ldots+P_t$ where $P_1, P_2,\ldots, P_t$ are maximal connected subgraphs of G-K. Let $v_i \in P_i$, $v_j \in P_j$, and $i \neq j$, if there exist paths between v_i and v_j then every one of these paths should contain at least one of the vertices of K.

If there exists a path between v_i and v_j and this path does not contain any vertex of K. If we remove K from G, we have $P_i \cup P_j$ belongs to G-K and is connected. This contradicts the definition that P_i and P_j are maximal connected subgraphs of G-K. Hence, the corollary always holds.

3.2 CVS vs. NDBS

When a user issues an agent, all information he has is which servers are to be visited and all the paths to be taken by the agent. Only with these information, NDBS is not applicable. All the possible paths can't be enumerated exhaustively by a spanning tree. Our solution is to use a graph to cover all those possible paths and then employ a Cut Vertices Search (CVS) to track the agent from the graph. We use a set of vertices, $V = \{v_1,v_2,v_3,...\}$, to represent these servers and the server to be probed first is called the start vertex. The set of edges, $E = \{e_1,e_2,e_3,...\}$, represents those paths between any two servers that the agent may travel. Hence, all possible paths of the target agent are included in $G(V,E)$.

We assume that the target agent is suspended when it is located by a search agent. Since graph G includes all possible paths and the agent resides in one of the vertices of the graph. To locate the residing vertex of the target agent, we have the following search scenario:

First, we find out the cut vertices set, $K = \{k_1,k_2,..,k_m\}$ of G and then probe each of these vertices sequentially until the target agent is found. If the agent can't be found in K, we remove K from G. In this case, now we have a disconnected graph, G-K. Checking with the log files of vertices of K, we would know the component, say P_a, that the agent resides in. We reset G to be P_a and continue the same process as above until $|G|=1$ or when the agent is found.

3.3 CVS Strategy

The detail of CVS strategy is as follows.

Step0 : if $|G|=1$, probe the vertex in the graph to see whether the agent is there or not and report the result. Terminate the search process.

Step1 : Determine the cut vertices set, $K = \{k_1,k_2,..,k_m\}$, where $m \geqq \kappa(G)$.

Step2 : Probe those vertices of K sequentially. If the agent resides in K then it would be found at last. Report agent location and terminate search process.

Step3: Else, remove K from G and the graph, G-K, becomes a disconnected graph, say G-$K = P_1+P_2+..+P_t$, where $P_1, P_2,.., P_t$ are maximal connected subgraphs of G-K and there exist no paths between any two of them. From the log files of vertices of K, we have the following

two alternatives:

Case1: None of the vertices of K has been visited, that is the target agent hasn't passed through the vertices set, K. Hence, the agent still stays in one of the components, say P_a, that contains the start vertex.

Case2: At least one of the vertices of K has been visited. Find the latest visited vertex by comparing the elapsed times in the log files. Find the next stop of that vertex, say v_n. Then, we know that the agent resides in some component, say P_a, that contains v_n. Note here, vertex v_n is the start vertex of component P_a.

At the end of Step3, we would find the component, P_a, that the target agent resides in.

Step4 : Set G to be P_a. Go to step0.

3.4 Search Example

Assuming that we are given a graph $G(V,E)$ as shown in Fig. 3. G is a graph with order 15, v_1 is the start vertex and bold edges represent the path the target agent moves along. We see that the agent visits v_1, v_2, v_5, v_4, v_9, v_8, v_{12} and v_{11} sequentially and resides in v_{11} now.

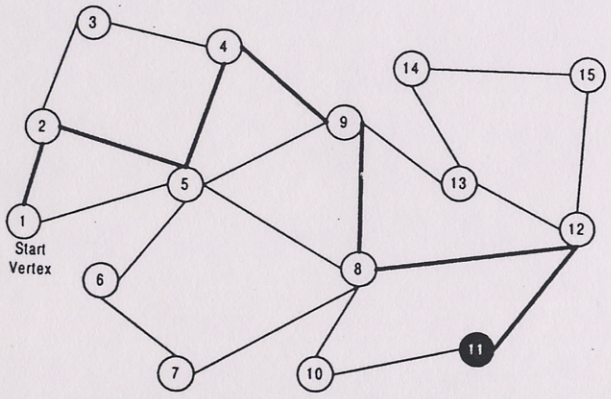

Fig. 3 The path the target agent moves along.

Loop1:

Step0 : $|G| \geqq 1$ and the agent isn't found.

Step1 : Determine cut vertices set, $K=\{v_8, v_9\}$.

Step2 : Probe v_8 and v_9 sequentially. Since the agent does not reside in K, it isn't found. Go to step 3.

Step3 : Remove K from G, then the graph G-K becomes the one in Fig. 4. Here, we have $G-K = P_1+P_2$ and $P_1=\{v_1, v_2, v_3, v_4, v_5, v_6, v_7\}$, $P_2=\{v_{10}, v_{11}, v_{12}, v_{13}, v_{14}, v_{15}\}$.

Case2 : v_8 and v_9 are visited. From the log files of them, we find that v_8 is the latest visited vertex of K, v_{12} is the next stop of v_8, and is contained in P_2.

Step4 : Set G to be P_2 and the start vertex is set to v_{12}. Go to step0.

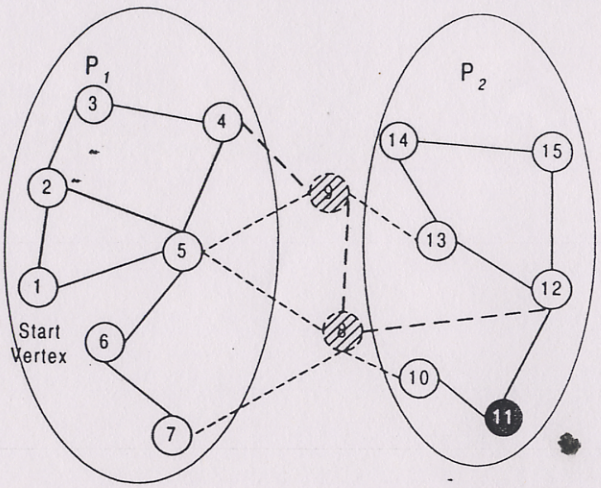

Fig. 4 Vertices v_8 and v_9 are removed from G and the graph G-K is partitioned into P_1 and P_2.

Loop2 : At the end of loop1, the graph G is reduced to the one in Fig. 5. The start vertex becomes v_{12} and the agent still resides in v_{11}.

Step0 : $|G| \geqq 1$ and the agent isn't found.

Step1 : Determine cut vertices set, $K=\{v_{12}\}$.

Step2 : Probe v_{12}. Since the agent doesn't reside in K, it's not found. Go to step 3.

Step3 : Remove K from G, then the graph G-K becomes the one in Fig. 5. Here, we have $G-K = P'_1+ P'_2$ and $P'_1=\{v_{13}, v_{14}, v_{15}\}$, $P'_2=\{v_{10}, v_{11}\}$.

Case2 : v_{12} is visited. From the log file of it, we find that v_{11} is the next stop and is contained in P'_2.

Step4 : Set G to be P'_2 and set the start vertex to *be* v_{11}. Go to step0.

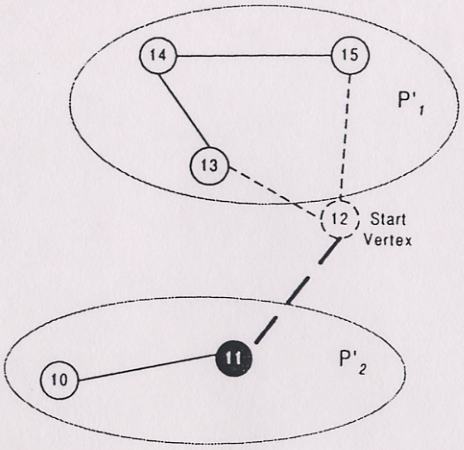

Fig. 5 Vertex v_{12} is removed from G and the graph G-K is partitioned into P'_1 and P'_2.

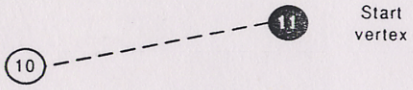

Fig. 6 The target agent is found at v_{11}.

Loop3 : At the end of loop2, we have $V=\{v_{10},v_{11}\}$ in graph $G(V,E)$. The start vertex becomes v_{11} and the agent still resides in v_{11}.

Step0 : $|G| \geqq 1$ and the agent isn't found.

Step1 : Determine cut vertices set, $K=\{v_{11}\}$.

Step2 : Probe v_{11}. Since the agent resides in K, it is found (Fig. 6). Report the agent location and terminate the search process.

3.5 Complexity Analysis

Given a graph $G(V,E)$ of order n. To determine the complexity, we should analyze how many loops the search process goes through and how many vertices the search agent probes in a loop.

In graph G, the removal of cut vertices set will partition G into at least two components. In step2 and 3, the residing component, P_a, is found after all cut vertices are probed. In step4, the cut vertices set and all the components other than P_a are excluded. The graph G is then reset to be P_a. The order of the graph is reduced from $|G|$ (= n) to $|P_a|$. Since P_a is one of the components of G and G is composed of at least two components. Hence, $|P_a| \leqq \frac{n}{2}$ on average.

In general, we can exclude $\frac{n}{2}$ number of vertices, and the target agent won't be found until $|G| = 1$ in the worst case. So, the target agent will be found after *log n* loops of execution on average. Furthermore, the number of vertices probed in each loop is $|K|$. If we always choose the minimum cut vertices set in every loop, we have $|K|=\kappa(G)$, where $\delta(G) \geqq \kappa(G)$. So, the expected number of vertices to be probed is $\kappa \log n$. Where $\kappa = \max \{\kappa_i(G) | \kappa_i(G)$ is the number of vertices to be probed in $loop_i$, where i is a nature number.$\}$. The complexity of CVS is thus in the order of $O(\kappa \log n)$.

3.6 Refined work of CVS

Though, CVS releases the assumption that all the possible paths should be included in a spanning tree. However, CVS is under an assumption that the target agent is suspended when it is located by a search agent. In this section, we use listening agent and adaptive CVS to further remove this assumption.

3.6.1 Listening Agent

Listening agent is a special agent that is assigned to each server. When a search agent probes a server to find a target agent, if it is not found, the search agent sends a search record to the listening agent associated with the server. A search record contains search agent ID, target agent ID and the server name that the target agent can send its location to. A listening agent is used to check whether the new coming agent is the target agent. At the time the target agent comes, the listening agent reports the location of the target agent. Fig. 7 illustrates the use of listening agent when the target agent is not suspended.

The scenario of using listening agents is as follows. Let the cut vertices sets determined in $loop_1$, $loop_2$,...,$loop_n$ be $K_1,K_2,\ldots,K_n$ and the current residing component be P_a. In
, if it goes without using listening agents, when the target agent leaves P_a and passes K_n after we probed K_n. The search agent will not know the pass through and expect that P_a is the residing component of the target agent. Obviously, the target agent won't be found eventually.

On the other side, if the search agent leaves search records to all the listening agents after probing $K_1,K_2,\_,K_n$. Once the target agent passes any vertex of K_n, the pass through will be aware by the listening agent and the

location will be reported.

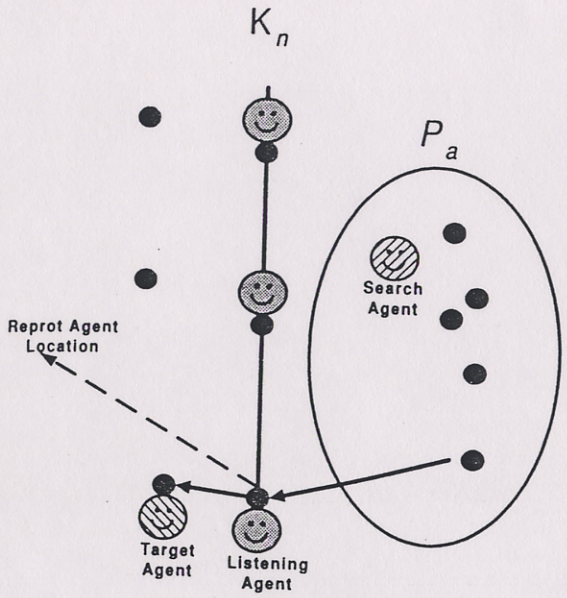

Fig. 7 Use of listening agent when the target agent is not suspended.

3.6.2 Adaptive CVS (ACVS)

Adaptive CVS (ACVS) is another solution to release the assumption of target agents suspension. At the beginning, the search goes through the same process as CVS does. That is, we determine cut vertices set and then probe each vertex sequentially. After probing all the vertices, we know the latest visited vertex of K, the departure time of the target agent, and its next stop. If the agent leaves very recently, we switch our search strategy from CVS to sequential search. Here, we use the departure time and statistics of the service time to determine the right time of switching search strategies.

For example, we check with the figures in Fig. 3 and 4. When the target agent resides in v_{11} and the search agent comes to v_{12} expecting that the target agent might reside in either v_{10} or v_{11}. Say that the search agent chooses v_{10} to probe. What happens if the target agent leaves v_{11} and passes through v_{10}, v_8 and comes to v_5. Only CVS couldn't help us to find the target agent. However, if we switch the search policy from CVS to sequential search at this time. The target agent will eventually be found no matter the target agent is suspended or not. The result is shown in Fig. 8.

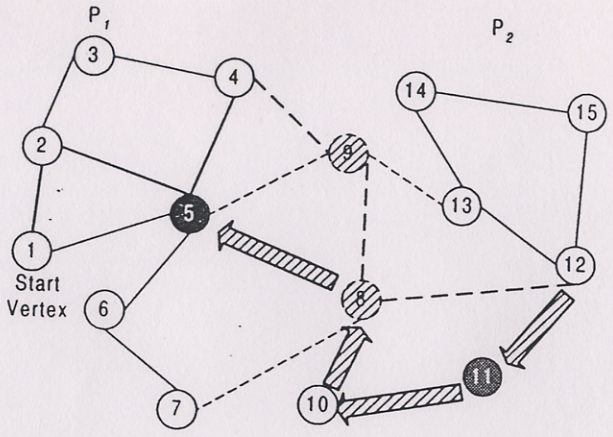

Fig. 8 Using adaptive CVS to search target agent by switching search policy.

4. Conclusion

In a mobile computing environment that supports mobile agents, a client is able to send an agent to visit a sequence of servers in a network. To track the locations of agents becomes a critical issue in managing a mobile agent service network.

Lien proposed a number of intelligent search strategies in [8]. However, all those strategies base on an assumption that the agent to be searched moving along a pre-deterministic path. However, in real world, the paths of an agent are varied according to distinct visited servers. In [2], Jang used a spanning tree to represent all possible paths of an agent and proposed a NDBS search method to release the assumption of pre-deterministic paths with time complexity in the order of $O(m + log())$.

However, NDBS runs under an assumption that we should know not only all the servers to be visited by the target agent but also all the deterministic and non-deterministic servers. In this paper, we propose Cut Vertices Search (CVS) to locate a target agent. All we need to know is the whole servers to be visited by the agent and the server that sends the agent out. The complexity of CVS is in the order of $O(\kappa \log n)$.

5. Future Research

The following is a list of a number of issues to be covered in the future research.

5.1 Lost Agent

An agent might be lost because of failures. The reasons may be the agent itself, the server it resides, or the network it moves along. Agent lost might be significant or not. In a real time control system or a business transaction system, agent lost may cause catastrophe. It becomes more complicated when concurrent executions are allowed.

5.2 Concurrent Search

Sequential search is simple but consumes time. If a client is allowed to submit more than one search agents to a network, search time may be saved to some degree. We may consider to have a search agent create child search agents to cover all possible paths in a non-deterministic situation such as in making an if-then-else decision. This therefore results in the following issues:

1. How to converge forked agents?
2. What to do if either child agents or parent search agent get lost?
3. How to terminate all child agents?
4. How to coordinate (share information and control) among sent out child agents.

5.3 Exact Search vs. Approximate Search

The execution of a target agent is kept going, the exact current location of an agent might be changed when the client receives the acknowledgement sent back by search agent. To ensure the location unchanged after it is found, a freeze agent is needed to be sent together with search agent. Otherwise, what we get is only an approximate location rather than an exact location.

Reference

1. Imielinski and B. R. Badrinath, "Mobile Wireless Computing: Challenges in Data Management," Communication of ACM, August 1994.
2. Hung-Chin Jang, Yao-Nan, Jyh-Shyan Huang, and Fu-Han Liu, "Non-Deterministic Binary Search of Mobile Agents", 1997 National Computer Symposium (NSC '97), Taiwan, R.O.C., Dec. 22-23, pp.89-94, 1997.
3. Hung-Chin Jang, Yao-Nan, Jyh-Shyan Huang, and Fu-Han Liu "New Intelligent Search of Mobile Agents", First Agent Technology Workshop, Taipei, Taiwan, Dec. 4, 1997.
4. Yao-Nan Lien, "Client and Agent Mobility Management," Proc. of the Second Workshop on Mobile Computing, Hsing-Chu, Taiwan, March 1996, pp. 141-152.
5. 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.
6. 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.
7. 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.
8. Yao-Nan Lien, Fuhan Liu, Chun-Wu Leng and Wen-Shyan Chen, ``Intelligent Search of Mobile Agents", 1997 International Conference on Computer System Technology for Industrial Applications, April, 1997, pp. 110-116.
9. Yao-Nan Lien, Fuhan Liu, Wen-Shyen Chen and Chun-Wu Leng, "Asymmetric Binary Search of Mobile Agents ", 1997 International Symposium on Multimedia Information Processing, Dec. 1997, pp. 294-299.
10. Maes, "Agents that reduce work and information overload", CACM, July 1994, pp. 30-41.
11. Weiser, "The computer for the 21st century", Scientific America, 1992, pp. 94-104.
12. TIA/EIA IS-41, "Cellular Radio Telecommunications Intersystem Operations", Telecommunications Industry Association, Dec. 1991.