A Naming Service Architecture and Optimal Periodical Update Scheme for a Multi Mobile Agent System

Suphithat Songsiri
Dept. of Communication Systems, FernUniversität Hagen
Universitätstr.11, D-58084 Hagen, Germany
suphithat.songsiri@fernuni-hagen.de

Abstract

An agent's mobility presents a challenge to the communication framework in the mobile agent paradigm. A mobile agent communication, e.g. messaging, can only occur when the actual location of the mobile agent is known and a reliable message delivery scheme is utilized. This paper concentrates on designing a globally distributed naming service architecture, which aids in locating a mobile agent and finding the optimal time threshold. The optimal time threshold aims to minimize the sum of the cost of periodical updates and the cost of maintaining forwarding pointers.

1. Introduction

Communication, by means of exchanging information and knowledge sharing, is paramount for mobile agents to collaborate with one another. Dale et al. [1] demonstrate two forms of communication namely; synchronous (i.e. synchronous dialogue) and asynchronous (i.e. sending messages). To send a message to a mobile agent, its location must be known. To locate a peer mobile agent, the following three processes must eventuate: name resolution, location update and searching or tracking scheme. The name resolution process comprises of an entity sending the name of a mobile agent to a name server, and a name server returns the mobile agent's latest update location. In the location update process, a mobile agent sends its current position to a name server. Searching is a procedure, where an entity, after retrieving a mobile agent's location from a name server, performs a broadcast scheme to find the target mobile agent. The tracking scheme is a method where each visited host maintains some information, e.g. forwarding pointer, leading to the actual location of a mobile agent. The well-known hurdle of designing a positioning system is the mobility of a mobile agent. A location update algorithm alone is inadequate to guarantee the accuracy of the actual location of a mobile agent. In order to escalate the accuracy of a target mobile agent's location, a coalescence of location update algorithm with either the tracking or searching algorithm should be effectuated. This paper concentrates solely on the periodical location update algorithm and tracking (forwarding pointer) algorithm consortium. The reason underlying the combination of time based location update and forwarding pointer is to eliminate the weaknesses of the forwarding pointer [15, 16, 17], which are node's failure and lengthy forwarding pointer, as well as to identify the status of a mobile agent. This paper does not discuss node failure handling, instead proposes to reduce the size of the forwarding pointer so as to provide an efficient naming service architecture. Finally, this paper derives the optimal threshold for periodical location update, which yields the lowest total cost of location update and forwarding pointer maintenance. This paper commences with section 2 where definitions and related work are explained. Section 3 presents the background and problem statements. Section 4 demonstrates a naming service architecture, server selection and load balancing schemes. Section 5 describes optimal threshold calculation and location update algorithm. Section 6 presents the summary.

2. Definitions and Related work

The definitions of the frequently used terms in this paper are given as follows:

- **Mobile agent**: An active object acting on behalf of its owner and autonomously deciding on which location it will visit next.
- **Current vs. Actual Location:** Current location refers to the location where location update is performed, whereas actual location indicates the agent's actual residence.

- **The Geographically Distributed Naming Server Clusters** (GDNSC): Sets of servers, responsible for providing the latest location of the target mobile agent in the system in response to a request. All the servers in a cluster are connected by a local area network. Each server in a cluster has the same hardware specification.
- **Server Cluster Manager** (SCM): A front-end node that receives all of the inbound requests sent to a cluster and redistributes incoming requests to the servers in the cluster. It acts as the centralized dispatcher of a cluster with fine-grained control on client request delegation.

There are numerous studies concerning mobile agent messaging protocol and locating mobile agent algorithm. For example, Feng [24] uses the mailbox concept. The mailbox acts as a message buffer, which stores incoming messages. The mailbox can however be detached from its owner. Baik et al. [8] present a message-transferring model in multi-region computing environment by broadcasting a "message received" notification. Tolman [25] summarizes the strategies of locating a mobile agent. From these mentioned studies, we conclude that efficient naming service and mobile agent location method are essential to every message transfer model.

3. Background and Problem Statements

This paper considers a free roaming mobile agent scenario where the owner dispatches its mobile agent to carry out some tasks during its lifetime. Each mobile agent has a unique name defined using the naming function, and lifetime T_{Life}. Once its lifetime has expired the mobile agent migrates back to the owner. Besides performing some tasks on the visited host, the mobile agent must periodically update its current location to a name server. The visited hosts must maintain pointers to the host to be visited next by the mobile agent until the consecutive update is done. Once a new location update has been done, the mobile agent sends messages informing each visited host to erase their forwarding pointers. Before proceeding any further, this paper will present some problems posed by naming function, location update and naming service architecture.

In this paper, an agent's name is used to track its location. It is not impossible that several agents belonging to different owner possess the same name. To avoid the latter, the naming function algorithm is designed to produce a unique name for each mobile agent. The location of a mobile agent is an important issue for location-aware applications. Due to the mobile agent's mobility, an efficacious location update scheme needs to be developed. To design an effective location update algorithm, there are two important issues to be considered namely: where and when should a mobile agent update its location. The first issue can be determined by using the server selection method discussed in section 4.2. The second can be distinguished by the following three methods:

- Simple Method: Whenever a mobile agent migrates to the next host, it updates its location to an authorized entity.
- Movement-Based Method: Whenever a mobile agent completes *d* migrations between hosts, it updates its location to a name server [5, 16]. The critical weakness of this method occurs during the period between the first migration and the d^{th} migration, as the status of a mobile agent cannot be easily determined. During this period a mobile agent could have been killed or prevented from migrating.
- Time-Based Method: A periodical location information update after a certain interval of time τ has elapsed. The effectiveness of this method is not determined by the behaviour of a mobile agent. This method allows mobile agent's status identification. It is assumed that a mobile agent has been killed if a location update to the authorized entity fails to ensue.

Wright [10] demonstrates that the basic properties of naming service architecture are scalability, security and fault-tolerance. A naming service architecture can be categorized into two paradigms:

- Centralized Scheme [6, 7, 8, 9]: Only one name server acts as a central database. Since everything relies on only one entity, it is obvious that it will become a single point of failure.
- Distributed Scheme [10]: This can be roughly categorized into three types. The Cluster type is a group of servers connected to each other by local area network. The front-end node of a cluster connects to the outside world and distributes loads equally to each server. Even though this scheme provides load balancing and easy management, the front-end node undoubtedly becomes a single point of failure. The Geographically Distributed Servers type consists of servers, which are physically installed on different locations. Unfortunately, this scheme poses the danger of dispensing stale location information. For instance, a mobile agent "*A*" updates its current location to a server "S_1". At that same time, an entity requesting server "S_2" for *A*'s location will be provided with *A*'s latest location, and not the current location or S_2 does not have *A*'s location. The Geographically Distributed Cluster of Servers (GDCS) type comprises of physically dispersed clusters of servers. According to Shehory et al. [14], the distributed scheme proves to be more efficient in comparison to the centralized scheme.

4. The Naming Service Architecture

This section attempts to explain the naming service architecture design chosen to support global and secure name resolution proposed in this paper. From the problems indicated in section 3, the GDNSC architecture has been depicted in this paper due to its ability to expunge the bottleneck and single point of failure problems incurred from a single name server. Having a mobile agent's name in hand, a host chooses the best server cluster by using the server selection algorithm, and sends a location request to the chosen cluster. One of the servers in the cluster will then return the location of the mobile agent to the host. The SCM assigns the received request to one of the servers by performing load distribution. If none of the servers in the cluster is available due to excessive load, the SCM will relay all incoming requests to another SCM. Each cluster provides a single virtual interface to the outside world. The only address visible to the client is the virtual IP address corresponding to one device, i.e. SCM. The mechanism of how to designate the responding server will be further elucidated in the sub-section 4.2. As previously mentioned in the preceding section, the availability of a mobile agent's name is quintessential for retrieving the actual mobile agent's location. The next sub-section expounds the generation of a unique name by employing naming function algorithm.

4.1. Naming Function

The naming function is a function that generates a message containing a mobile agent's readable name and secure parameters for identification purposes. A *Name* is a syntactic entity consisting of symbol alphabet(s) as well as numerics, which denotes an object. Roth [18] demonstrates that a globally unique name could be used to defend against impersonation; i.e. the chance to create another agent that maps to the same globally unique name is negligible. Recently, Wright [10] employs a hierarchical DNS-styled agent naming format, which provides a readability characteristic, but is vulnerable to impersonation attack. Roth's algorithm protects against impersonation attack, however lacks in readability. Even though a unique name is always assumed in many literatures, they are devoid of an explicit description of a unique name generation. Hence, this paper endeavours to determine an appropriate unique name generation method. As a conclusion, a name should have the following properties:

- Singularity: One name belongs to only one mobile agent.
- Unpredictability: No one can generate a mobile agent, which has the same name as an existing mobile agent.

In this paper, the public key infrastructure is assumed to be available. $ENC_S(m)$ denotes an encrypted message m with host S's public key. $Sig_S(m)$ indicates host S's signature on a message. The unique name generation is accomplished by registering a message (R) to the name server. A readable name (R_N) should be exclusively different from other agents' names of the same owner. The mobile agent's kernel comprises of initial value and codes. Mobile agent's information refers to which tasks are to be accomplished, the agent's ability, optimal time threshold, total number of updates, and its lifetime. The last three values are further elaborated in section 5.1. This encrypted message is sent by the owner and then registered at the name server. The registration message is illustrated in the following equation:

$$R = ENC_{Cluster}\left[Sig_{owner}\left[R_N, kernel, Agent's\ information\right]\right] \quad (1)$$

After having received (1), the request is decrypted and the signature attained is broadcasted to the other clusters in GDNSC. They will send an acknowledgement back to the broadcaster at the acquirement of the signature. At the last entry of acknowledgement, the selected cluster will send a "registration completed" notification back to the host. This indicates that the name is now registered in the GDNSC. The possibility that another host could have generated the same readable name as an existing agent's name belonging to another host is not nonexistent. This will not, however lead to a name conflict, since the registration at the name server enforces the signing of the readable name, agent's kernel and agent's information with the private key of the owner host. Rendering thereby the uniqueness of each name registered at the GDNSC.

4.2. Server Selection and Load Balancing

This sub-section deals with the server selection algorithm for the choice of the responding name server cluster. The SCM of a cluster of servers in GDNSC appears to be the server that responds to incoming requests, even though in actual fact it only relays the requests to one of the servers in the cluster. The name server selection algorithm in this paper is not to be misinterpreted to be the selection of a responding server. The incoming requests allotment performed by a SCM is accomplished by administering the load-balancing algorithm. The load-balancing algorithm is discussed later in this

sub-section. Due to the disseminated location of each cluster, we necessitate to have a method to choose one of these clusters to be the service point. This paper considers server-side and client-side [11] selection method. The client-side selection method is most appropriate for the widely scattered clusters in the GDNSC architecture. The cluster appointment is done by client, based on selection methods like client-server proximity [19, 20], random selection, server's load and so on. In this paper, a cluster selection is attained by using client-server proximity selection method. Favourably, the cluster with the smallest proximity to the client will be depicted. This server selection method contributes; to a certain extent, to the performance improvement of the system. Generally, to quantify the proximity between client and server, three possible consideration issues which are number of hops (or routers), round trip time (RTT) and number of administrative system (AS) hops, can be viewed. This paper considers only the number of hops and round trip time. The main reason for this decision is to avoid the time-consuming computation in acquiring the number of AS hops [19]. Apart from that, having computed the number of AS hops does not help determine the best server to respond to the request. The *Traceroute* command is utilized to determine the number of hops and RTT between a client and a server. The *ping* utility is used to measure RTT between a client and a server. Silva et al. [19] point out that RTT should be used when trying to reduce client's perceived latency, while the number of hops is a good indicator of network resource usage. The cost of performing *ping* is obviously much less than *Traceroute*, but *Traceroute* provides more information. As a conclusion, the best client-server proximity can be computed by incorporating the RTT and number of hops (N) as follows:

$$I_C = \min_{i=1}^{N}\{(\alpha * RTT_{CS_i} + \beta * N_{CS_i})\} \qquad \text{with } \alpha+\beta=1 \quad (2)$$

The selected cluster is CS_{I_C}. The values of α,β refer to respectively perceived latency and network usage. They may be adjusted according to the user's requirements. This algorithm can be performed as frequently as necessary.

After a mobile agent obtains a selected name server cluster, it may now send a request to that cluster. We now turn to the problem of how each SCM effectively decides which server of the cluster should serve the client's request. This is where the server-side selection method is employed in this paper. The server-side selection method focuses on server clusters in which a dispatcher; in this case the SCM, equally distributes requests to one of the servers contained in the cluster or redirects the requests to another cluster of servers in case the cluster's load exceeds a predefined value. The SCM consigns the requests based on dispatching policy. Cardellini et al. [13] describe two algorithms for dispatching policy: static or dynamic algorithms. Since the static algorithm does not take into consideration any state information while making assignment decision, it proves to be the best solution in preventing the SCM from becoming the primary bottleneck of the cluster. The dynamic algorithm, on the other hand, takes into account a variety of system state information, and thereby has the potential to outperform the static algorithm. It however, introduces high computational complexity, which could result in a bottleneck problem. Hence, we choose the static algorithm to be our dispatching policy implemented in server cluster. Typically, static algorithm can be achieved by Random and Round-Robin (RR) schemes. A random scheme conveys the incoming requests uniformly through the servers with equal probability. The danger in using this scheme is that a certain server could receive multiple requests while another server receives nothing. Delay of service could arise once the overloaded server has reached its limits in processing the requests. To avoid the latter, we decided to employ the RR scheme at the SCM. To make a dispatching decision, RR uses a circular list and a pointer to the last selected server. SCM executes as follows:

- Request Distribution: each server in the cluster must send an online status message periodically to the SCM. Assume that S_i^c was the last chosen host, the new request will be assigned to S_j^c as follows:

$$j = (i+1) \bmod K \quad \text{where K = number of hosts} \qquad (3)$$

- Load Monitoring: SCM observes the overall cluster's load, so as not to exceed the predefined threshold; otherwise any new incoming request must be redirected to another cluster.

4.3. Current Location Information Retrieval

Assume that a host wants to request the location of a mobile agent. The first step that it has to accomplish is to find the best cluster of servers in the vicinity to respond to its request. Once the cluster has been selected, a host may now request for the agent's location by sending the location request (LR) message to the cluster. The LR is shown as follows:

$$LR = ENC_{cluster}\left[Sig_{RH}\left(R_N, \text{owner's Public Key}, \text{RH's ID}\right)\right] \qquad (4)$$

RH is a requesting host. Viewing the scenario described above from the cluster's aspect, a LR arrives at a server, and thus an agent's location is to be returned. The server then compares the received name and public key of the owner to its database, and returns the location. The current location message used in returning the agent's location is described as follows:

$$CL = ENC_{RH}[Sig_{Cluster}(R_N, \text{owner's ID}, \text{kernel}, \text{location}, \text{remaining\_time\_to\_update})] \quad (5)$$

Note that all the servers in a cluster share the same pair of public and private keys. The initial current location of a mobile agent is set to be the owner's location. Due to the complexity and interdependence of CL's components, the location update algorithm will be explicated in detail in the next section.

5. Optimal Time Threshold Calculation

The optimal time threshold, the total number of updates, and the agent's lifetime values are also in agent's name registration in (1). Hence, all these information must have been computed before the agent's name registration. The owner predestines the agent's lifetime. The number of updates and optimal time threshold (τ_{opt}) are essential information for the agent's location update. τ_{opt} is the value of τ which provides the minimum total cost of location update and maintaining forwarding pointers. τ is the period of time at which the agent is to update its current location. To derive the costs of location update and maintaining forwarding pointers, we model a free roaming mobile agent. The agent migrates to hosts H_1, H_2, and so on, and resided in the hosts for certain duration described by T_{H_I} which indicates the host residence time (HRT). The agent will perform a location update at every τ during its lifetime. It is not uncommon that an agent still resides in a host after updating its location. Thus, the time remaining for the agent to stay in the host is indicated as T_{R_I}. Every time the agent migrates to another host, the previous host maintains a forwarding pointer, which contains the next host's address. The duration for a host to maintain the forwarding pointer is indicated as T_{M_I}. The forwarding pointer is maintained until the next update has effectuated. Figure 1. illustrates the above-described scenario. The cost of performing a location update (U for U>0) accounts for bandwidth utilization and the computation requirement, which includes the expense of performing server selection as well as updating the database in GDNSC. Let C_{up} be the total cost of location update per mobile agent's lifetime. Thus, it is the cost per location update multiplied by the number of updates given by

$$C_{UP} = U * \left\lceil \frac{T_{Life}}{\tau} \right\rceil \quad (6)$$

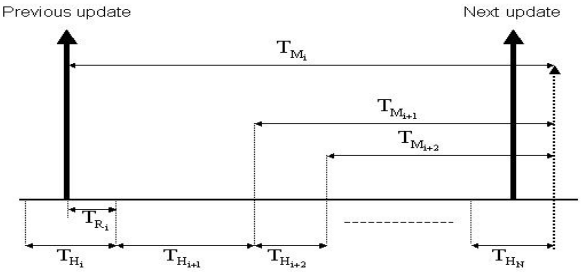

Figure. 1 Time Diagram

From (6), the number of updates can be easily derived from the dividing agent's lifetime by the time threshold, rounding the result to the closest integer. The range of time threshold is given as $0 < \tau \le T_{Life}$. C_{up} decreases for increasing τ. The cost of maintaining forwarding pointer per unit time M (M>0) can be obtained by considering the memory usage of each visited host. Let T_{H_i} be independent identically distributed random variable with a general distribution $F_H(T_{H_i})$, the probability density $f_H(T_{H_i})$ and expectation $E[T_{H_i}] = 1/\lambda_{H_i}$. Let $r_R(T_{R_i})$ be the probability density function of T_{R_i} and $R_R(T_{R_i})$ be the distribution. The probability that there are *i* visited hosts between two location updates us denoted by v(i).

Let C_M be the expected cost of maintaining forwarding pointers per lifetime. It is the cost of maintaining forwarding pointers per unit time multiplied with number of location updates and the average time to maintain forwarding pointer, given as:

$$C_M = M * \left\lceil \frac{T_{Life}}{\tau} \right\rceil * \left[\sum_{i=1}^{\infty} v(i) * \left[E[T_{R_1}] + \sum_{j=0}^{i-1} j * E[T_{H_{j+1}}] \right] \right] \quad (7)$$

The derivation of v(i) and $r_R(T_{R_i})$ is achieved as follows:

- Calculation of $r_R(T_{R_i})$: To derive this probability density, we apply the renewal theory by using the concept of renewal reward process to calculate residual life (i.e. in this case it is T_{R_i}). The detailed derivation can be found in [21]. We have

$$r_R(T_{R_i}) = \left[1 - F_H[T_{H_i}]\right] \Big/ E[T_{H_i}].$$

- Calculation of v (i): Consider fig 1. , v(1) indicates that there is one visited host between two location updates.

$$\begin{aligned} v(1) &= P[T_{R_i} > \tau] \\ &= 1 - P[T_{R_i} \le \tau] \\ &= 1 - R_{R_i}(\tau) \end{aligned} \quad (8)$$

For $i > 1$, we have:

$$\begin{aligned} v(2) &= P[T_{R_1} < \tau] * P[T_{R_1} + T_{H_2} > \tau] \\ &= P[T_{R_1} \le \tau] * [1 - P[T_{R_1} + T_{H_2} \le \tau]] \end{aligned}$$

$$v(i) = \left[\prod_{n=1}^{i-1} P\left[\sum_{j=1}^{n} T_{H_j} \le \tau\right]\right] * \left[1 - P\left[\left(\sum_{j=1}^{i} T_{H_j}\right) \le \tau\right]\right] \quad (9)$$

Please note that $T_{H_1} = T_{R_1}$.

- The term $\left[E[T_{R_1}] + \sum_{j=0}^{i-1} j * E[T_{H_{j+1}}]\right]$ from (7) describes the total amount of forwarding pointers maintaining time which is gathered from i visited hosts given v(i). For example, given v(4) indicating the four visited hosts H_1, H_2, H_3 and H_4 and two location updates. The first location update occurred at H_1 and the second update is to occur at H_4. The total amount of forwarding pointers maintaining time is shown as follows:

$$\begin{aligned} T_{M_1} &= E[T_{R_1} + T_{H_2} + T_{H_3} + T_{H_4}] \\ T_{M_2} &= E[T_{H_3} + T_{H_4}] \\ T_{M_3} &= E[T_{H_4}] \end{aligned} \quad (10)$$

The total cost is obtained by $C_{up}(\tau)+C_M(\tau)$. For the calculation of the optimal threshold parameters; the probability density of HRT, U and M must be carefully specified. The $f_H(T_{H_i})$ in this paper has been assumed to be exponentially distributed. In [22], Ross explains that the lifetime of an instrument e.g. a random variable can be assumed to be exponentially distributed, because such a distribution does not deteriorate with time, is easy to work with and often a good approximation to the actual distribution. This distribution has only one parameter, which is the failure rate λ or mean. λ in this paper refers to the average HRT. Due to the complexity of (6) and (7), the optimal time threshold value has been procured by means of simulations conducted using Matlab. At the commencement of the simulation, some difficulties in estimating the values of the constants T_{Life}, U, M, and λ were encountered. These values should never be assigned solely upon assumption. Instead, proper experiments must be conducted so as to acquire the most appropriate values. For example, the best way to determine the most appropriate λ will be to assign one task for the mobile agent to perform on different hosts and use the algorithm proposed in [23] (i.e. the failure rate for exponential distribution can be estimated either graphically on probability plotting paper, or analytically using either least squares or maximum likelihood). The value of T_{Life} could be easily estimated when the owner knows the exact number of hosts (i.e. predefined itinerary) a mobile agent will visit. The same applies to the estimation for the values of U and M. The acquirement of the U and M values are yet to be investigated. The following values were assigned for demonstration purposes only: $T_{Life} = 200$, $\lambda = 0.1$, U = 2 and M=1. The maximum number of visited hosts (i) is 15.

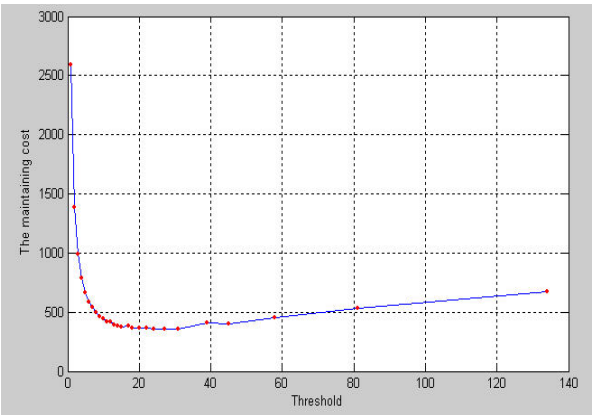

Figure. 2 Optimal Threshold

From Figure 2., τ_{opt} has been determined to be 24. However, since the costs of τ = 27 and 31 are not significantly different, these values can also be adopted to be τ_{opt}. Thereby, it can be concluded that the threshold value can take up values 24, 27 or 31.

5.1. Location Update and Stale Location Information

With the availability of the optimal threshold value, we can now proceed with mobile agent location update. In precedence to the location update, the mobile agent has to ask the current host for the best name server in the vicinity according to the algorithm described in (2). At the attainment of the server, the host produces a signature on the agent's kernel and its own IP address. After obtaining a

cluster server's address and the signature from the host, the mobile agent generates UL and transmits it to the designated cluster.

$$UL = ENC_{Cluster}\left[\left[R_N, Remaining\_updates\right], \left[Sig_{host}(kernel, host\_addr\right]\right] \quad (11)$$

Once the SCM has received UL, it then broadcasts the decrypted UL to the other clusters in GDNSC and awaits acknowledgements. At the last entry of acknowledgement, the location update is completed. If a certain host requests for the mobile agent's location, the server will return the location, annexing the agent's remaining time for the next update. This hinders any host from obtaining stale location information. The signature generated by the host obviates the repudiation of the fact that the agent has resided in this host. In return, this ensures smooth tracking of the mobile agent.

Let us consider a scenario where a host retrieves an agent's location just shortly before the agent performs its following update. The address obtained by the host at that instance will be the address of the first visited host. As previously mentioned, the hosts visited by the agent maintain forwarding pointers to be visited next host in the agent's path only as long as the agent has not updated its next location. Hence, as the host contacts the first visited host to track the agent's next location, the pointer to the next host might have well been erased, since the agent has performed its next update. Undoubtedly, the tracking of the agent breaks down at this point. Thereby, the attached agent's remaining time for the next update in the returned current location allows the requesting host to consider between tracking the agent immediately or to request the location at a later time.

6. Summary

The naming service architecture presented in this paper provides secure name resolution for a large-scale mobile agent system. Since the GDNSC architecture encompasses globally distributed clusters of name servers, a server selection algorithm is proposed in this paper for the closest cluster selection. The Server Cluster Manager performs load balancing to distribute cluster's load to the members. In case of overloading, some incoming requests may be relayed to other clusters in GDNSC. The naming function described in this paper produces mobile agent's unique name using the conjunction of public key infrastructure and digital signature. This paper also proposes on how to obtain the actual location of the mobile agent by using the combination of periodical update (i.e. to determine the mobile agent's latest location) and the forwarding pointer (i.e. the pointers which lead to the actual location of the mobile agent). The optimal time threshold has been procured to determine the best period for location update, which gives the minimum cost of location update and maintaining forwarding pointers.

7. Acknowledgement

The author is currently a doctoral candidate under the supervision of Prof. Dr.-Ing Firoz Kaderali at the department of Communication Systems, Fern Universität Hagen Germany. The author wishes to thank Prof. Dr. rer. nat. habil. Werner Poguntke and Prof. Dr.-Ing Firoz Kaderali for the encouraging discussions that led to the development of this model and reviewing this paper. The author is fully funded by the project " Mathematic and Engineering Science Method for Secure Data Transmission and Information Transfer" from DFG (German Research Community).

8. References

[1] F.G. McCabe , J. Dale."*Asynchronous Messaging "*. FIPA 98 Draft Specification part1.

[2] J. Li, H. Kameda, K. Li." *Optimal Dynamic Mobility Management for PCS Networks*" IEEE Transaction on Networking vol.8 June 2000.

[3] Y. Fang, I. Chlamtac, YB. Lin." *Portable Movement Modelling for PCS Networks*" IEEE Transactions on Vehicular Technology, Vol 49., July 2000.

[4] YB. Lin. " *Reducing Location Update Cost in A PCS Network*" IEEE/ACM Transaction on Networking, Vol.5, Feb 1997

[5] T.Y Li, J.F Zhang. "*An Optimal and Secure Tracking Scheme for Mobile Agent*". AAMAS 2002.

[6] P.T. Wojciechowski. "*Algorithms for location Independent Communication between Mobile Agents*", Technical Report DSC-2001/13, Departement Systemes de Communication, EPFL, 2001.

[7] D. Milojicic, W. LaFroge, D. Chauhan. " *Mobile objects and agents*" Proc 4th USENIX conf. On Object-Oriented Technologies and System,1998

[8] M. Baik. K.W. Yang, J. Shon, C. Hwang." *Message Transferring Model between Mobile Agents In Multi-region Mobile Agent Computing Environment.*" LNCS 2173, 2003

[9] X. Feng, J. Cao, J. Lü, H. Chan. " *An efficient mailbox-based algorithm for message delivery in mobile agent system*" Proc. 5th Intern. Conf. On Mobile agemt ,Atlanta, USA, LNSC 2240, 2001

[10] T. Wright. " *Naming Services in Multi-Agent System: A Design for Agent-based White Pages*", AAMAS 2004

[11] S.G. Dykes, K. A. Robbins, C.L. Jeffery. " *An Empirical Evaluation of Client-side Server Selection Algorithms*" IEEE INFOCOM, VOL. 3, MARCH 2000, pp.1361-1370.

[12] M. Sayal, Y. Breitbart,P. Scheuermann, R. Vingralek." *Selection algorithms for Replicated Web Server*", ACM SIGMETRICS, Volume 26, 1998

[13] V. Cardellini, E. Casalicchio, M. Colajani, P. Yu. " *IBM Research report: The State of the Art in Locally Distributed Web-server Systems*", 2001

[14] D. B. Ami, O. Shehory. " *Evaluation of distributed and Centralized Agent location Mechanism*", CIA 2002 , LNAI 2446, pp 264-278,2002

[15] L. Moreau. " *A Fault-Tolerant Directory Service for Mobile Agents based on Forwarding Pointers*." SAC 2002 Madrid Spain

[16] J. Ahn. " *Fault-Tolerant and Scalable Communication Mechanism for Mobile Agents*", ISCIS 2004, LNCS 3280, pp. 533-542, 2004

[17] J. Ahn. " *Decentralized Inter-agent Message Forwarding Protocols for Mobile Agent System*", ICCSA 2004, LNCS 3045,PP 376-385

[18] V. Roth, J. Peters. "*A Scalable and Secure Global Tracking Service for Mobile Agents*", MA'2001, LNCS 2240, pp. 169-181.

[19] K. Obraczka, F. Silva." *Network Latency Metrics for Server Proximity*", IEEE GLOBECOM (2000), pp. 421–427

[20] Z. Mao, C. Cranor, F. Douglis, M. Rabinovich. " *A Precise and Efficient Evaluation of The Proximity between Web Clients and their Local DND Servers*" USENIX 2002

[21] S. M. Ross. " *Stochastic Processes*" 2 nd ed New York, Wiley 1996

[22] S. M. Ross. " *Introduction to Probability Models*" 6th ed. 1997

[23] "*Calculating the parameter of the exponential Distribution*" www.weibull.com

[24] X. Feng. " *Design and Analysis of Mobile Agent Communication Protocols"*, Master thesis, 2002

[25] C. Tolman. " *A Fault-Tolerant Home-Based naming Service for Mobile Agents*", Master thesis, 2003