CONCURRENCY AND COMPUTATION: PRACTICE AND EXPERIENCE
Concurrency Computat.: Pract. Exper. 2014; **26**:447–467
Published online 1 February 2013 in Wiley Online Library (wileyonlinelibrary.com). DOI: 10.1002/cpe.3006

Performance analysis of a mobile agent prototype system based on VIRGO P2P protocols‡

Yunliang Jiang[5], Yong Liu[1,2,*,†], Wenliang Huang[3] and Lican Huang[4]

[1]*Institute of Cyber-Systems and Control, Zhejiang University, Hangzhou, China*
[2]*State Key Laboratory of Industrial Control Technology, Zhejiang University, Hangzhou, China*
[3]*China Unicom Labs, Beijing, China*
[4]*Zhejiang Sci-Tech University, Hangzhou, China*
[5]*School of Information & Engineering, Huzhou Teachers College, Huzhou, China*

SUMMARY

The mobile agent technique has been broadly used in next generation distributed systems. The system performance measurement and simulation are required before the system can be deployed on a large scale. In this paper, we address performance analysis on a finite state mobile agent prototype on the basis of *V*irtual H*i*erarchical T*ree G*rid *O*rganizations (VIRGO). The finite states refer to the migration, execution, and searching of the mobile agent. We introduce a novel evaluation model for the finite state mobile agent. The experimental results based on this evaluation model show that the finite mobile agents can perform well under multiple agent conditions and are superior to the traditional client/server approach. Copyright © 2013 John Wiley & Sons, Ltd.

Received 27 June 2011; Revised 4 January 2013; Accepted 4 January 2013

KEY WORDS: mobile agent simulation; VIRGO; performance analysis

1. INTRODUCTION

In future computing environments, the cost of moving data will be much larger than the cost of moving codes. Although the bandwidth of a network has been improved significantly, the most challenging problem is still the large cost of transferring the data to perform computation. Thus, the mobile agent (MA) technique may be a desirable solution to address this challenge, especially when the computation logic is complex, for example, when users want to process both their data in the storage of a public cloud and their local private data alternately, or the scenario in Section 2.3. In addition, the MA will also be useful to address the challenge of the automatic provision of services and will effectively manage workload segmentation and portability (in other words, the seamless movement of workloads across many platforms and clouds) [1].

Cloud computing [2, 3] and virtual organization (VO) techniques [4–6] are new network architectures that enable self-organization and self-management. Currently, the ecosystems of cloud services require tight coupling with clients deployed on each of the users (or peers) to improve the user experience. Those clients may be powerful platforms for MA, for example, iTunes from Apple Inc. Chome from google, or eMule, which can be implemented as a type of cloud service that is constructed with numerous peers and can be designed as a peer platform§ to execute the agents for

*Correspondence to: Yong Liu, Institute of Cyber-Systems and Control, Zhejiang University, Hangzhou, 310027, China.
†E-mail: yongliu@iipc.zju.edu.cn
‡Here, a platform such as iTunes or eMule installed on PCs could correspond to the agent home that is mentioned in the following section of this paper.
§Here, a platform such as iTunes or eMule installed on PCs could correspond to the agent home that is mentioned in the following section of this paper.

services such as music sharing, software dispatch, and safety. The VO technology can help decrease the complexity of MA substantially by hiding the operating system and communication differences among the network nodes. The VO-based MA applications could be one of the major forms of next generation network applications. On the one hand, the internet software ecosystem has evolved into a cloud architecture that is constructed with numerous peers. For example, the 360 safe guider[¶] is installed on three hundred million personal computers (PCs) in China, which is resident on a background of computers and provides antivirus, software update management and system optimization services. It can now share the malicious software information via all of the client peers that are installed in the user's PC and provide an antivirus cloud service. The old Internet Protocol (IP)-based management protocols could be difficult to organize for the large numbers of peers and to provide convenient cloud services; thus, the virtual group-based architecture and MA techniques are desired. On the other hand, current MA-based applications are limited because of their safety and supporting environments; the next generation cloud architecture with tiny peers [7, 8] could provide a perfect solution for the MA-based applications. The cloud client software installed on peers could provide communication supports for those agents and could also restrict the access contents of the agents (this strategy could refuse malicious agents) such that the agents can focus on their tasks and simplify the services that are designed. Thus, the performance of a mobile agent system with large peers and organized with VO should be evaluated before being deployed in a real case; however, there are few studies that focus on this field.

Mobile agent is an intense research area [9–13], and many implementations have taken place in real-world applications [12, 14–16], which were developed based on prototype systems. The MA applications can autosearch the resources (including computation, data, and specific devices) in the networks and move to the proper network node to achieve their tasks [11, 13]. These applications could provide an easy way to execute comprehensive tasks; however, to evaluate the real system performance of the MA under networks is a challenge because the network conditions are quite complex and the deployments of resources are dynamic.

As measuring the metrics in real MA systems is intractable and time-consuming, the simulation of MA systems is still valuable to study. The simulation of an MA can obtain metrics rapidly and easily and can evaluate the performance easily and quickly under varied network conditions and system settings, developing further MA simulation techniques is useful.

1.1. Related works on performance evaluation of MA

There have been considerable works on MA simulation and performance evaluation [14, 15, 17–28]. The essential of the simulation on MA's performance evaluation is that the simulation should approach to the real system as far as possible. Thus, there are two main approaches in performance evaluation of MA: one is using some probability distributions to approximate the executing process and concurrency of MAs [26, 27]. Although those mathematical approaches could provide concise representation of the performance evaluation and simulation, it is hard to say those evaluations could approach to the real system accurately. For the mutation, concurrency of MAs may not only be described with several probability functions simply, let alone complex tasks with MAs on cloud computing environments. In the other hand, once the task of MA changed, the probability functions used in evaluation may be varied; the other approach is building real min-system to execute the MAs directly [28]. The drawback of these approaches is obviously, it could not provide references for real large multiple-MA-system especially in complex network environments. In additional, both two approaches mentioned earlier did not provide flexible evaluation on performance of MA, which means those evaluation approaches are most aimed to the same stale task of MA and cannot provide scalable evaluation for different tasks on MA system.

With the evolution of network environments, the performance evaluation of MAs have undergone four stages, which is MA's performance evaluation on wired Transmission Control Protocol/IP network [23], Remote Method Invocation (RMI) [18], Wireless network [19], 3G network [27] and so forth. And the researchers have also compared the performance of MA with Remote Procedure

[¶]www.360.cn

DOI: 10.1002/cpe

Call [23], client/server (C/S) model [29–32] under varied network environments. To the best of our knowledge, there is not any work on the performance evaluation of MAs under the VO environment. Thus, proposing proper performance evaluation models for MAs under new network architecture, for example, VO networks or cloud architectures, is needed.

We address this problem and propose a quantitative performance evaluation model for that architecture with a large number of finite state MAs (FS-MA) [33] under a virtual hierarchical tree Grid organizations (VIRGO) platform [5, 6] in the following section.

1.2. Contributions of our approach

The MA-based performance evaluation approach provided in this paper is addressed on the following technical challenges:

(i) Current performance evaluation approaches could not provide accurate simulation and evaluation for multiple MAs' performances in large-scale network environments. The mathematical evaluation approaches [26, 27] can only use the probability distribution to roughly approximate the MA's actions; once users want to change the evaluation tasks of MA, those probability functions may be invalid. Although using the real evaluation system with min-network condition, [28] could not provide accurate performance estimation for large-scale networks, such as the cloud or VO environments. Besides, if users want to evaluate different tasks for MAs, then they normally need to rewrite the codes of MA. This may be very inconvenience for the pre-deployment simulations in complex services, which often need to adjust the tasks configurations. So, the first challenge may be the accurate performance evaluation for MAs with varied tasks in large-scale networks.
(ii) As mentioned earlier, sometimes, the evaluation tasks are varied, whereas both current approaches need to either rewrite the codes of MAs or readjust the mathematical models. How to provide a reusable and flexible performance evaluation for MA when processing complex evaluation tasks may be the second challenge.
(iii) When concerning the concurrency of multiple MAs, there should be more reality than modeling with assuming probability function that means the evaluation system should execute those multiple MAs in real; otherwise, the concurrency could not be truly evaluated only with mathematical functions. This is the third challenge.
(iv) The fourth challenge is how to evaluate the MA's performance under the large-scale VOs. Although there are several works on the performance evaluation of MA under wired networks and wireless network, the evaluation works of MA on VOs are still in the blank stage.

On the basis of the aforementioned implementation challenges in performance evaluation of MAs, we present our FS-based MA performance evaluation approach, which may be novelty and useful in the future cloud computing architecture and will be able to solve the challenges.

Our contributions on the performance evaluation of MA are presented as follows:

(i) We present a scalable performance evaluation approach for MA (Section 2.2), which summarizes the whole life cycle of MA as an automotioning FS machine (FS-agent) driven by resources, and then complex tasks of MA can be abstracted as content-FS sequence defined by the users freely. With this approach, users can focus on the design of content-oriented tasks without concerning the detailed executing processing of MA, which is concentrated with the FS-agent. This approach is also network independent and can be implemented in varied network environments.
(ii) We present a real executing policy in the performance evaluation of MA, which means all the MAs defined by the content-FS are really executed in the simulation platform; the workload of MA's tasks and service time for MA's data size transferring are calculated from the true environments. This could provide more accurate estimation for the performance of multiple MAs in large network environment. And we also use multiple-thread method to execute multiple MAs to simulate the real concurrency among multiple MAs.

DOI: 10.1002/cpe

(iii) We present several metrics for MA system, such as *average service availability* and *average aggregate bandwidth*, to evaluate the performance of MA system under VIRGO architecture.
(iv) We also compare the performances of MA-based approach and service-oriented architecture (SOA)-based approach on VIRGO architecture and provide a reference for application design with those two approaches.

1.3. Organization of sections

The remainder of this paper is organized as follows. In Section 2, we introduce the VO architecture-VIRGO and the finite state MA system. Section 3 provides details about the performance evaluation model and the corresponding simulation environment. The experimental results investigating the C/S model and the MA model are discussed in Section 4, and conclusions are drawn in Section 5.

2. MOBILE AGENT BASED ON VIRGO VIRTUAL ORGANIZATION ARCHITECTURE

2.1. Virtual hierarchical tree Grid organizations

Virtual hierarchical tree Grid organization [6] is an open-source project that is self-organizing and decentralized, on the basis of unstructural and structural peer-to-peer; it merges an n-tuple replicated virtual tree-structured network and a random cached unstructured network. It can provide a series of lookup protocols to manage its virtual groups and resource searching similar to virtual and dynamic hierarchical architecture [4].

Users can access VIRGO from the access-point node. Once the new user node is added into VIRGO, it will be managed by its owner nodes. VIRGO uses gateway nodes to route the different groups (or levels). A typical two-tuple virtual hierarchical tree is shown in Figure 1; the real nodes are mapped into virtual groups (see the dashed lines) and are overlapped. Each virtual group includes two nodes from the upper layer . In our MA model, we use VIRGO [6] to support the mobile computation. The login and logout of the nodes use a VIRGO protocol [6]. An important part of the MA-based computation model is to perform a resource/service search before actual migration/computation can take place. With an appropriate search policy in place, a tree-structured VO can decrease the search time and cost significantly. In the following FS-MA performance evaluation, we adopt the full tree search policy [4] as the default search policy. In the future, we may use the VIRGO protocol [5, 34] to avoid the possible load bottleneck of the nodes in the root layer.

2.2. Finite state mobile agent

The FS-MA system consists of two important components, namely the FS-agent and the content-FS. The FS-agent is an FS machine that represents the state transition diagram of an MA. The content-FS defines how a complex task can be executed and deployed with the composition of subfinite states. It plays a similar role as Web Services Flow Language in a Web Service and can also be viewed as a workflow description protocol for the execution and migration of the MA [33].

FS-agent in finite state mobile agent

The life cycles of the entire actual MA can be defined as an automotioning FS machine driven by resources. Here, resources represent the general designation of data, devices, and software in remote nodes that will potentially affect the state transition of mobile agents. That is why, we call those MA as FS-agent. As shown in Figure 2, the FS machine includes five states and one transition relationship between those states:

◇ *Request state* - each agent begins its life in this state and tries to request the resources and execution time slot, which are required for the task.
◇ *Blocked state* - once an agent obtains necessary service time, it transits into this state. All the service mobile agents who obtain enough service time while without enough service resources at certain nodes will be put into the blocked state queue.
◇ *Suspended state* - an agent will transit into this status once it obtains all the necessary resources. The system maintains a queue for the agents in this state that have enough resources while without enough service time at certain nodes.

DOI: 10.1002/cpe

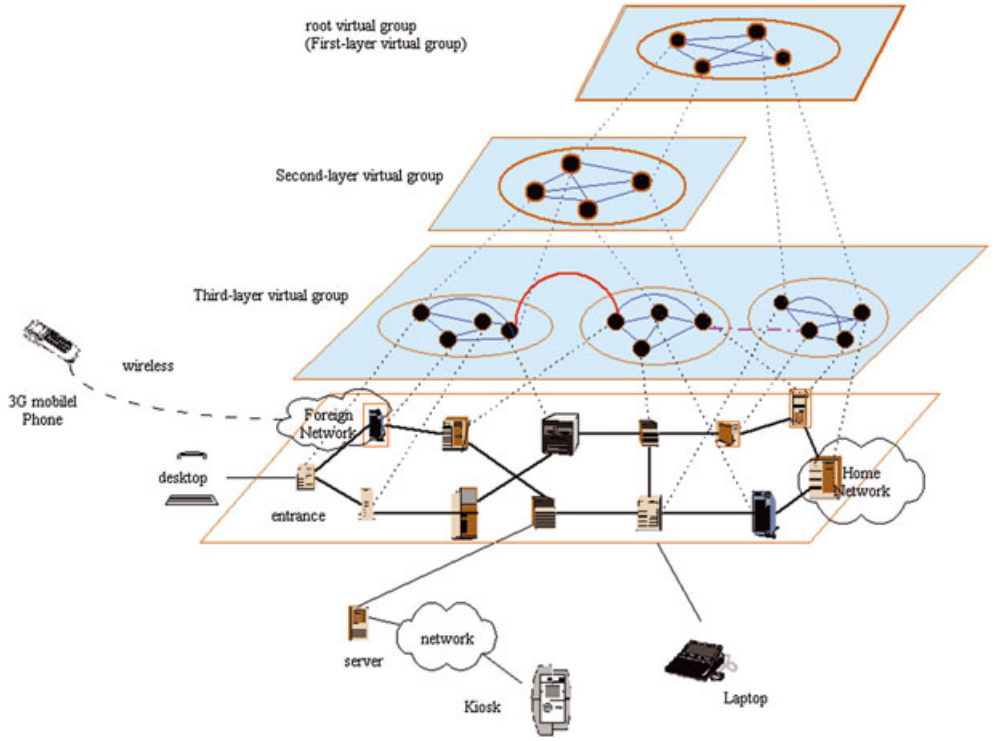

Figure 1. Virtual hierarchical tree topology [6].

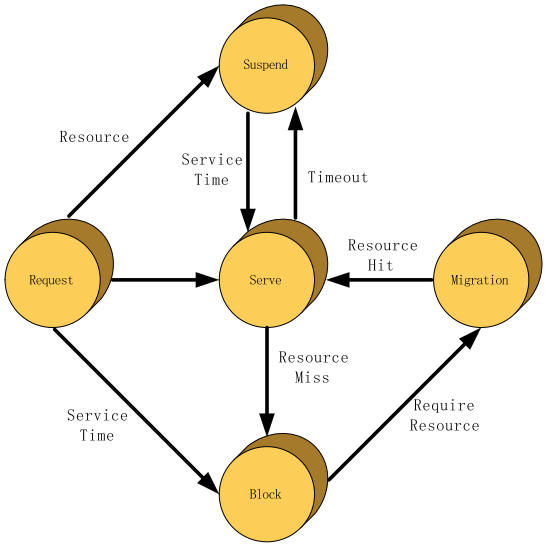

Figure 2. Transition relationship of the finite state agent.

- ◇ *Migration state* - this state represents that an agent tries to obtain necessary resources from a remote node after it fails to acquire those resources from the current node.
- ◇ *Serving state* - if an agent obtains enough resource and service time, then it will execute its service.

In a real FS-MAs system, any FS-agent is actually executed in a container called agent home. The agent home is deployed at all the nodes and essentially provides following services to all the agents:

DOI: 10.1002/cpe

```
<?xml version="1.0"?>
<ContentFS>
   <STATE stateid="0" call="S0" nstatenum="1">
      <NextS stateid="1" >
         <rdf:RDF>
         <rdf:Description
          rdf:about="http://icad.8866.org/Mining/localhost">
            < facilities:LogDB>available</ facilities:LogDB >
            < facilities:location>localhost</ facilities:location >

         </rdf:Description>
         </rdf:RDF>
      </NextS>
   </STATE>
   <STATE stateid="1" call="S1" nstatenum="1">
      <NextS stateid="2">
         <rdf:RDF>
         <rdf:Description
          rdf:about="http://icad.8866.org/Mining/Remote">
            < facilities:ComputS>available</ facilities:ComputS >
            < facilities:location>localhost</ facilities:location >

         </rdf:Description>
</rdf:RDF>
      </NextS>
   </STATE>
  ..........
  ..........
</ ContentFS >
```

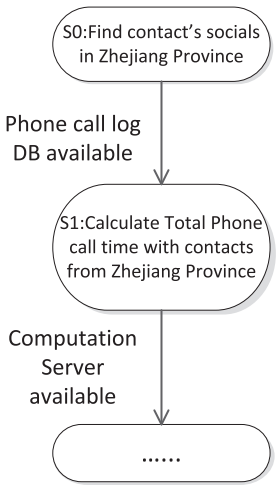

Figure 3. A simple content-FS for the social data mining task in telecomm.

- ◇ All the functions of an agent are invoked by the agent home indirectly.
- ◇ Queue management and agent execution scheduling - It maintains several different queues (e.g., blocked queue and suspended queue) for the agents at the current node and is responsible for allocating service time for them if necessary.
- ◇ Resource searching - It will proactively search resources on the behalf of blocked agents.

Content-FS in mobile agents

Content-FS is a service description protocol for FS-MAs; and content-FS is similar to Web Services Description Language and Web Services Flow Language, which provide coarse-granularity service description for web service.

There are three elements in content-FS, that is, subservice states, resources, and transition relationship. Subservice states represent subtasks in a service task and are basic building blocks in the service workflow design process. The resources refer to required resource and service time for each subtask, and only the resources arrives at the requirements; the content-FS can switch from one substate to another substate directly. Transition relationships in content-FS describe the functions and logic workflows of the services. A service typically begins with a start state and stops at an end state.

Figure 3 shows a simple content-FS for the social data analyzing task in telecomm, which try to calculate one user's total phone call time in Zhejiang Province. In this example, 'S0' and 'S1' are the binding name for the corresponding subservice states. The 'NextS' tag describes the next state of current state, and the corresponding required resources are described by the 'rdf' tag.

2.3. Background and implementation scenario of finite state mobile agent

China Unicom is the second telecom operator in China and consists of multiple subcompanies, which are divided by province, and those subcompanies are allowed to carry out their own information systems and operate account independently. The architecture of China Unicom is similar with the VO, which may be autonomy in each level of organization. This architecture is favorable for each subcompany because of its flexibility in making changes. However, this makes it not easy to integrate the information systems and data from all the subcompanies. Thus, China Unicom tries to

DOI: 10.1002/cpe

carry out a project to integrate all the information systems and data of all the subcompanies, aiming at providing a low cost, scalable approach that can easily share all the information systems and data. The MA is an optimal approach for its low cost and highly scalability.

Why is MA, why not SOA?

Firstly, it is highly complex and costly for China Unicom to redesign all the database and information system with the SOA architecture; secondly, it is also impossible to require each subcompany to upgrade all their information systems and data to the service architecture. Even if parts of their information systems and data are switched to SOA, it is very hard to design a perfect granular of services to be able to handle new tasks. Besides, the SOA needs the resource owner to build and deploy such services to others. This is not practical for in many user scenarios because (i) those services are normally required by other subcompanies, the service provider may not fully understand the specific requirements of the service from each individual customer, and both the provider and users need to spend much cost on communication and designing for services; and (ii) in particular for the China Unicom organization architecture, the service usually cannot bring revenue to the service provider (the subcompany), and as a consequence, they do not have enough motivation to provide high quality service.

In the MA-based approach, all the tasks are specified and developed by the users, and thus, it will avoid the risks mentioned earlier. Besides, in this case, the MA-based approach will also decrease the cost compared with SOA; the MA only need the subcompanies to provide interfaces for their information systems and data instead of wrapping all those systems and data as services which is a much heavier and complicated task.

Although MA is a superior approach for the integration project, it still needs to evaluate the performance of MA-based approach before its deployment. This is the main original intention of this article.

Two metrics, that is, the response time and the service availability, are used to compare the C/S-based approach and the MA-based approach. The response time can also be regarded as the agent executing time, which can be represented as the waiting time of the end users. The service availability will reflect the available time for the task executions. We will first model the computation of these two metrics and then build simulation experiments for comparing agents with clients in the condition of large-scale nodes.

3. EVALUATION OF FINITE STATE MOBILE AGENT

3.1. Finite state mobile agent work scenarios and process

The most important thing in the FS-MA computing process is finding one node with a series of resources, which can drive the FS-MA to the next state. In this process, the FS-MA only needs to communicate with a local node in the VIRGO platform, and the node in the VIRGO platform will implement all of the remote communications for the current FS-MA.

The typical agent can be modeled as the states' switching process in Figure 4. *State 0* and *state 1* represent the operations of obtaining the mining services and retrieving the phone call log database, respectively. R_0 and R_1 represent the corresponding resources. Figure 4 shows the FS-MA executing process in detail and that the FS-MA can be viewed as a batch of executing states, which will be driven by the request resource.

(i) A two-state agent 'exam' in Figure 5 is deployed on node X in the VIRGO platform, initialized with *state 0* (all FS-MAs beginning with *state 0*).
(ii) Agent 'exam' needs resource R_0 to implement its *state 0*, so it sends an R_0 request message to the VIRGO platform. We can view the VIRGO platform as a black box and need not address the details of resource searching in the FS-MAS executing process.
(iii) The VIRGO platform will find a node with R_0 and will inform the local node X.
(iv) After the exam receives the message from the local host indicating that node Y has the resource it needs, it will transfer to node Y.
(v) Agent 'exam' executes *state 0*.

DOI: 10.1002/cpe

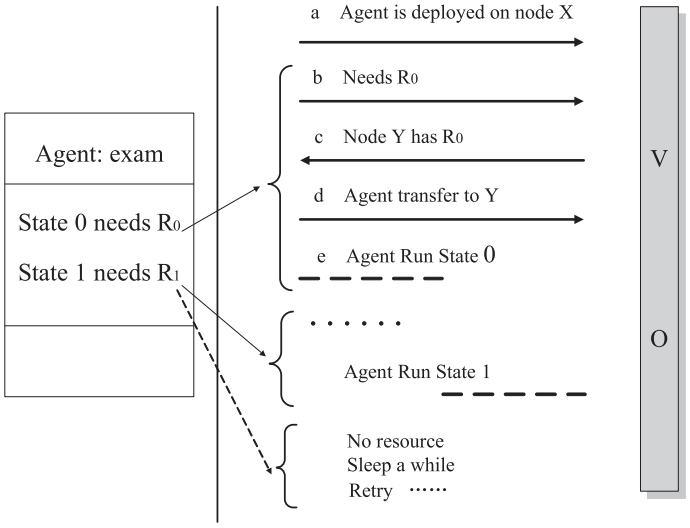

Figure 4. Finite state mobile agent executing process.

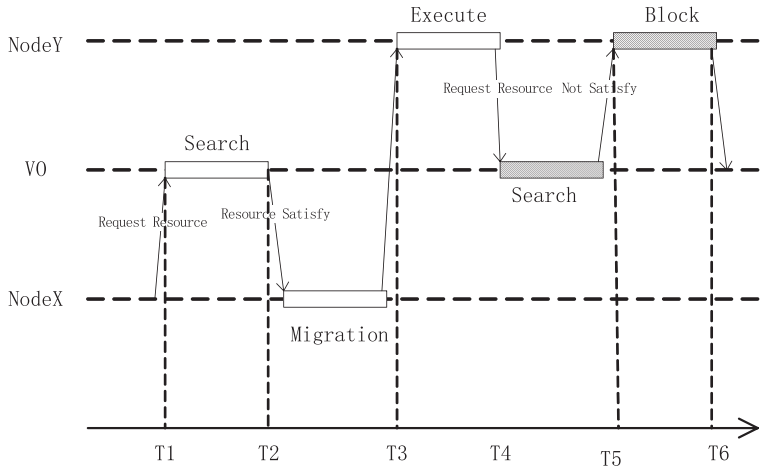

Figure 5. Time sequences of the agent executing process.

State 1 will repeat the process that is described earlier. When all of the nodes with the required resources that exam needs are unavailable, the VIRGO platform will send back the *UNFIND* signal, and thus, the exam will switch to the block array and be frozen until the required resources are available.

The time sequences of the agent executing are shown in Figure 5. According to the figure, the executing process can be divided into several subtime intervals. These are the time for the agent searching the destination nodes, T_{search}; the time for the agent migration, $T_{\text{migration}}$; the time for the agent executing in remote nodes, T_{execute}; and the time for agent blocking, T_{block}. As Figure 5 shows, we have $T_1 < T_2 < T_3 < T_4 < T_5 < T_6$, and in our evaluation model, we ignore the time beacon delay between those subtime intervals. Then, we have the following formulas:

$$T_{\text{search}} \approx T_2 - T_1 \tag{1}$$

$$T_{\text{migration}} \approx T_3 - T_2 \tag{2}$$

$$T_{\text{execute}} \approx T_4 - T_3 \tag{3}$$

DOI: 10.1002/cpe

$$T_{\text{block}} \approx T_6 - T_5 \tag{4}$$

3.2. Resources search algorithm

We use the full tree search query policy [4] in VIRGO for resources searching. First, we find the root virtual group, and then the coordinate of virtual group members of this group forwards the query message to all of their son members. All of these members execute in parallel, sending the message down to the members of their low-layer groups until the query message reaches the leaf nodes. The nodes that have the needed resources will answer the search query after receiving the message. Finally, VIRGO will select one node to be the resource target, according to the response time, and return it to the searcher. The full tree search query algorithm is given as follows [4]:

Algorithm 1: Full tree resource query

Input: node P, which queries the specific resource
Output: node Q, which contains the queried resource

```
1 P.send(qmessage, gate_node(P));
2 C = rcvg_node(gate_node(P));
3 gate_node(P).send(qmessage, C);
4 Q = C.Route(qmessage);
5 return Q;
```

The *gate_node(P)* function will return the gateway node of *P*, and the *rcvg_node(P)* will return the gateway node in the root layer of VIRGO. The function of route is given as follows:

Algorithm 2: *cvg.Route(qmessage)*

```
if Type(VGroup(cvg)) == leaf then
    for each q ∈ VGroup(cvg) do
        if q.compare_resource(qmessage) == true then
            q.send(rmessage,gate_node(q)), return q;
        end
    end
end
else
    for each p ∈ VGroup(cvg) do
        p.Route(qmessage);
    end
end
```

Here, *Type* will return the type of the virtual group (if it is a leaf group of the VO group), and *VGroup(p)* will return the group of *p*, *q.compare_resource(m)* and will compare the resources contained in node *q* and queried by message *m*. The node will return true when matched.

The search algorithm is based on the message passing approach. The messages are forwarded by the nodes in the virtual group. Because the communications in the virtual group are not based on the request-acknowledge method, the message dropping condition may occur. The message dropping condition refers to the condition whereby messages are lost at a certain part of the virtual group. In our search algorithm, another condition, namely resources missing, may occur when the resources requested by the agent are not found in the virtual group. Because of the messages dropping and the resources missing, there should be maximal search time latency for the finite state agent execution. In other words, once the agent requests the resources and exceeds the maximal search time latency,

DOI: 10.1002/cpe

it will be treated as message dropping, and the agent will be switched to the block queue. When an agent switches to the block queue, it will start a timer, and a block queue query time latency will be given. Then, the local node will resend the resources request for the agent in the block queue once its timer count exceeds the query time latency.

The maximal search time latency and the query time latency normally are given by the virtual group, and their values are affected by the network conditions, such as the size and scale of the whole virtual group. This scenario hops between the top root and the bottom leafs. In our FS-MA system, agents or local nodes can also specify these two values individually.

3.3. Parameters and evaluation model

Later, we define all of the parameters used in our model for easy reference. The parameters are listed as follows:

- ◇ S_A is the average size of the agent (bits). The parameter S_A is the average size of the agents, and includes the agent codes, the executing states of the agents, and the temporal data when executing.
- ◇ C_n is the number of substates in the agent.
- ◇ N_A is the number of agents deployed in the VO. The parameter N_A is the total number of agents running in the VO. In our simulation experiments, we use the same agent to execute in the system. Then, the substates number of each agent, C_n, is equal in our experiments.
- ◇ i is the index of the substates of the agent, $1 \leqslant i \leqslant C_n$.
- ◇ j is the index of the agent deployed in the VO.
- ◇ S_M is the search message data size (bits).
- ◇ $T_{\max}$ is the maximal searching time latency.
- ◇ B is the bandwidth available in the VO (bits/s). Here, B is the raw bandwidth of the VO network, but the practice bandwidth can never reach the raw bandwidth. We assume that the real message transfer bandwidth would achieve B_M and that the agent data transfer bandwidth would achieve B_A after the influence of the network protocols and hardware devices.
- ◇ β_A is the communication overhead factor for agents' migration ($\beta_A < 1$).
- ◇ β_M is the communication overhead factor for messages in VO ($\beta_M < 1$).
- ◇ B_A is the effective bandwidth available for agent migration (bits/s), $B_A = \beta_A B$.
- ◇ B_M is the effective bandwidth available for messages (bits/s), $B_M = \beta_M B$.
- ◇ L_{xy} is the minimal network hops between node x and node y in VO.
- ◇ T_b is the minimal query time latency for the blocked agent (s).
- ◇ count_i is the block count for the ith substate executing of the agent. Because the agent requesting resources may fail multiple times, we use count_i to represent the block times of the ith substate executing.
- ◇ T_{S_i} is the ith substate executing time, $1 \leqslant i \leqslant C_n$ (s).
- ◇ N_y is the number of the unexecuted agent in queue of the node y.
- ◇ P_x^E is the performance of the substate executing on node x, (substates/s).
- ◇ α_x^E is the performance efficiency of the software and hardware platform on node x ($\alpha_x^E < 1$).

The agent's substate executing time, T_{S_i} , in different nodes is varied and relies on the hardware speed of the nodes and the software performance in the current nodes. We use P_x^E to represent node x's performance and α_x^E to represent the overhead of that node x, including the hardware processing speed, the software language, the compiler, the running time environment, and the system resources.

The substate executing time also depends on the unexecuted queue in the current node. Here, we apply the first-come-first-serve policy[‖] to this queue such that the agent should wait until there are no other agents before it in that queue.

Performance measurements for the mobile agent system. After giving the parameters used in the evaluation model, the following measurements are presented as the computed results of other previous parameters:

[‖]Here, we may add other policies for performance evaluation.

DOI: 10.1002/cpe

- ◇ $T^i_{\text{migration}}$ is the agent migration time for its ith substate.
- ◇ T^i_{block} is the agent block time for its ith substate.
- ◇ T^i_{search} is the agent searching time for its ith substate.
- ◇ T^i_{execute} is the agent executing time for its ith substate.
- ◇ T_{total} is the total agent consuming time, including the transferring time via network, the processing time in remote or local node, the resources searching time, and the blocking time. The total agent consuming time equates to the sum of the agent consuming time in each substate executing, migration, searching, and blocking. The total agent consuming can also be viewed as the response time for each agent as follows:

$$T_{\text{total}} = T_{\text{migration}} + T_{\text{block}} + T_{\text{search}} + T_{\text{execute}} \tag{5}$$

- ◇ R_A is the agent service availability. This parameter contains the ratio of the total agent executing time with the total consuming time, and we use this measurement to evaluate the performance of the agent system:

$$R_A = \frac{T_{\text{execute}}}{T_{\text{total}}} \tag{6}$$

- ◇ B_N is the agents' aggregate bandwidths. This parameter contains the used bandwidth in the agent's whole life cycle and defined as the total transferring data size versus the total agent consuming time.
- ◇ $\overline{B_N}$ is the average agents aggregate bandwidths. This parameter is the average value of all the agent aggregate bandwidths and can be used to estimate the system performance:

$$\overline{B_N} = \frac{\sum_{j=1}^{N_A} B_N^j}{N_A} \tag{7}$$

Computing the parameters. We deduce the performance measurements from the detail substate to the whole agent executing process. Assume the agent start to execute its ith substate ($1 \leqslant i \leqslant C_n$), and then we have the following computing formulas. The agent finds that the current node cannot provide sufficient resources for its ith substate executing, and then it will send out the search messages and request the resources. The searching time for the ith substate executing is the following:

$$T^i_{\text{search}} = \begin{cases} 2L_{x_i y_i} \cdot \frac{S_M}{B \cdot \beta_M} & \text{if resources are found} \\ T_{\max} & \text{if resources are not found} \end{cases} \tag{8}$$

Here, the x_i and y_i are the source node and destination node, respectively. The resources search time that is consumed should be null if the resources lie on the local node. The minimal network hops is the following:

$$L_{xy} \begin{cases} = 0, & \text{if} \quad x = y \\ \neq 0, & \text{if} \quad x \neq y \end{cases} \tag{9}$$

Then, the searching time that is consumed in the agent life cycle can be calculated as follows:

$$T_{\text{search}} = \sum T^i_{\text{search}} \tag{10}$$

which is the following:

$$T_{\text{search}} = \sum_{i=1}^{C_n} 2L_{x_i y_i} \cdot \frac{S_M}{B \cdot \beta_M} \tag{11}$$

Here, the condition of without resources is included into the block state, and the time consuming, $T_{\max}$, adds to the block time consumption.

DOI: 10.1002/cpe

If the agent finds the desired resources, then the agent will transfer to the remote node, and the time calculates as follows:

$$T^i_{\text{migration}} = L_{x_i y_i} \cdot \varepsilon \cdot \frac{S_A}{B \cdot \beta_A} \tag{12}$$

Here, the data transferring of the agent is based on the reliable connection. The ε is the coefficient factor of the latency in the network in our evaluation model, and thus, we have the following:

$$\varepsilon = \begin{cases} 0, & \text{if} L_{x_i y_i} = 0 \\ \frac{1}{L_{x_i y_i}} \leqslant \varepsilon \leqslant 1, & \text{if} L_{x_i y_i} \neq 0 \end{cases} \tag{13}$$

Because the total migration time consists of each migration time in the substate executing, we have the following:

$$T_{\text{migration}} = \sum T^i_{\text{migration}} \tag{14}$$

Then, the total consumed migration time can be calculated as follows:

$$T_{\text{migration}} = \sum_{i=1}^{C_n} L_{x_i y_i} \cdot \varepsilon \cdot \frac{S_A}{B \cdot \beta_A} \tag{15}$$

If the agent cannot find the desired resources, then in this condition, a block occurs. The agent home**, which deploys on the node, will block the agent for a while and will resend the searching messages for the agent after a certain latency period. Thus, the block time is the following:

$$T^i_{\text{block}} = T_{\max} \cdot \text{count}_i + T_b \cdot \text{count}_i \tag{16}$$

Then, the total block time in the agent's life cycles has the following:

$$T_{\text{block}} = \sum T^i_{\text{block}} \tag{17}$$

and

$$T_{\text{block}} = \sum_{i=1}^{C_n} (T_{\max} \cdot \text{count}_i + T_b \cdot \text{count}_i) \tag{18}$$

When agent arrives at the destination node and waits for its executing at that node, there may be some other agents coming before the new agent. Thus, an unexecuted agent queue is established in that node, and the agent home deployed in that node will manage and schedule the unexecuted agent queue. Then, the calculation for the agent executing time is related with the process node's state, including its hardware speed, its software overhead, and the length of the queue. Thus, the time for agent executing can be calculated as follows:

$$T^i_{\text{execute}} = \frac{n^i_{y_i} + 1}{P^{E_i}_{y_i} \cdot \alpha^{E_i}_{y_i}} \tag{19}$$

If the unexecuting agent queue is empty, then the $n^i_{y_i} = 0$ ($n^i_{y_i}$ is the number of MA in the waiting queue ahead of the arriving MA); the agent will be executed without any queue. The total executing consumed time is as follows:

$$T_{\text{execute}} = \sum T^i_{\text{execute}} \tag{20}$$

which is the following:

$$T_{\text{execute}} = \sum_{i=1}^{C_n} \frac{n^i_{y_i} + 1}{P^{E_i}_{y_i} \cdot \alpha^{E_i}_{y_i}} \tag{21}$$

**It is the software installed in nodes, which enables MAs working under its monitor.

DOI: 10.1002/cpe

After presenting the calculated formulas for each suboperation in the agent's *i*th substates executing, and the total calculation time of each suboperation, we can obtain the total agent consuming time as follows:

$$T_{\text{total}} = \sum_{i=1}^{C_n} \left(T^i_{\text{execute}} + T^i_{\text{migration}} + T^i_{\text{block}} + T^i_{\text{search}} \right) \tag{22}$$

$$T_{\text{total}} = \sum_{i=1}^{C_n} \frac{n^i_{y_i} + 1}{P^{E_i}_{y_i} \cdot \alpha^{E_i}_{y_i}} + \sum_{i=1}^{C_n} (T_{\max} \cdot \text{count}_i + T_b \cdot \text{count}_i) + \sum_{i=1}^{C_n} L_{x_i y_i} \cdot \varepsilon \cdot \frac{S_A}{B \cdot \beta_A} + \sum_{i=1}^{C_n} 2L_{x_i y_i} \cdot \frac{S_M}{B \cdot \beta_M} \tag{23}$$

Then, the *agent service availability* can be calculated as follows:

$$R_A = \frac{\sum_{i=1}^{C_n} \frac{n^i_{y_i}+1}{P^{E_i}_{y_i} \cdot \alpha^{E_i}_{y_i}}}{\sum_{i=1}^{C_n} \left[\frac{n^i_{y_i}+1}{P^{E_i}_{y_i} \cdot \alpha^{E_i}_{y_i}} + (T_{\max} \cdot \text{count}_i + T_b \cdot \text{count}_i) + L_{x_i y_i} \cdot \varepsilon \cdot \frac{S_A}{B \cdot \beta_A} + 2L_{x_i y_i} \cdot \frac{S_M}{B \cdot \beta_M} \right]} \tag{24}$$

The agent service availability is a ratio between the total consumption time and the executing consumption time of agent. It can reflect the efficiency of the MA system, for the consumption time, except the executing consumption time is the total overhead in the agent executing process.

The bandwidth workload of the agent is also another important measurement. Here, we use the *agent aggregate bandwidth* as the workload measurement. The workload in agents' running time can be classified into two parts: one is the searching information bandwidth usage and another is the transfer for the agent itself. The bandwidth workload for single agent can be calculated by the following formula:

$$B_N = \frac{\sum_{k=1}^{C_m} S_I \cdot L_{x_k y_k} \cdot \varepsilon + \sum_{i=1}^{C_n} S_A \cdot L_{x_i y_i} \cdot \varepsilon}{T_{\text{total}}} \tag{25}$$

Here, C_m is the total message number in the agent execution life cycle, and S_I is the total searching information size.

The bandwidth workload of the agent can be used to evaluate that of the whole network. The average bandwidth workload of the multiple agents can be calculated by formula (26), and this measurement can also be used as a network performance monitor measurement.

We can give out the average bandwidth workload of the agents as follows:

$$\overline{B_N} = \frac{1}{N_A} \sum_{j=1}^{N_A} \left(\frac{\sum_{k=1}^{C_m} S^j_I \cdot L^j_{x_k y_k} \cdot \varepsilon + \sum_{i=1}^{C_n} S^j_A \cdot L^j_{x_i y_i} \cdot \varepsilon}{T^j_{\text{total}}} \right) \tag{26}$$

3.4. Network topology simulation and parameters measurement

To evaluate the performance of the agent system, we first need to estimate reasonable parameters for the agent performance model that was presented. Because our FS-MA system is deployed on the Internet environment, the networks are as varied as the networks where VIRGO existed, measuring time points, and other parameters. It is difficult to record the agent's executing logs and to analyze the whole agent system's performance when a large number of agents are executing in the network. Thus, we conduct a small-scale test environment to measure the model parameters. The parameters measurement experiments environment consists of two normal PCs: one is an Intel Pentium 4 1.6 G processor, 512 M RAM, 80 G Maxtor hard disk, running in Windows XP professional SP2, and the other is an AMD1600+(1400 MHZ) processor, 512 M RAM, 80 G Maxtor hard disk running in Debian 3.1. The FS-MA system is deployed on these two machines, and both machines are

DOI: 10.1002/cpe

connected with a 10 Mbps wired Ethernet hub. In this test experiment, the maximal searching time latency and the minimal query time latency for the blocked agent are set as $T_{\max} = 5$ s and $T_b = 5$ s, respectively. The message size, S_M, in the FS-MA is always set to 1024 bytes.

3.4.1. Measuring β_A. A test experiment is performed on the 10 Mbps (that is, 10,485,760 bps) Ethernet network. To measure the β_A, we conducted approximately 100 times the transferring experiments between the two PCs, which deploy the FS-MA, and obtained the average transferring time. In these experiments, we ran the agents on those two machines and logged the total agent migration transferring time in each experiment. The agent size is 5 Mb (which is approximately 41,943,040 bits). The test result of the average agent migration time between these two machines is 5562.4 ms. Then, the effective bandwidth of the FS-MA system can be computed as the total transferring data size divided by the time. In other words,

$$B_A = \frac{41,943,040 \text{ bits}}{5.5264 \text{ s}} = 7,589,577 \text{ bps} \tag{27}$$

and we can calculate the β_A as follows:

$$\beta_A = \frac{B_A}{B} = \frac{7,589,577 \text{ bps}}{10,485,760 \text{ bps}} = 0.7238 \tag{28}$$

3.4.2. Message transferring time. Because the maximal message data size in our FS agent system is 1024 bytes, less than the size of a Transmission Control Protocol segment on the Ethernet, which is approximately 1500 bytes, the transferring time in this condition will be affected by the network latency more than the transferring bandwidth. In addition, we assume that the messages are forwarded by the nodes without buffering and delaying in our FS agent system. Then, the transferring time between two near hops of the messages can approximate the network latency, and in our test environment, the latency is approximately 400 ms. Thus, we have the following:

$$\frac{S_M}{B \cdot \beta_M} \approx 400 \text{ ms} \tag{29}$$

3.4.3. Coefficient factor of the latency in network. As the measurement of network bandwidth of the Internet is a challenge work, the available bandwidth in practice is affected by many factors [35, 36]. We use the coefficient factor to indicate the total affections, and we constrain the network bandwidth between the best and worst conditions by that parameter. In our performance evaluation, we seek to discuss the worst condition, which is $\varepsilon = 1$.

3.4.4. Simulations of the network topology and the agent system. We build a VIRGO platform on a PC, and we simulate multiple nodes, which could be communication and search resources for agents. The network topology simulation system consists of two parts: the node-controlling module and the environment-controlling module. The node-controlling module is implemented by a multiple threads Java program, which uses a similar thread to simulate the controlling and scheduling functions of the agent home for each node. The environment-controlling module simulates the practice network conditions and interacts with the node-simulating threads.

We also write a multiple-thread program using Java to schedule and simulate the multiple agents that execute in the VO architecture. Each thread controls running one agent, including searching in the VIRGO, migration in the VIRGO, blocking in the current node, and executing in the nodes. The program can simulate and record all of the agents' running processes, and the experimental results are presented in the next section.

DOI: 10.1002/cpe

Table I. Parameters of virtual organization-based network condition.

Parameters	Symbol	Value
Grid size	G_S	60×60
Number of nodes in grid	N	1000
Bandwidth	B	10 Mbps
Coefficient of latency time	ε	1

4. RESULTS

4.1. Multiple agents

In practice, the performances of different machines are varied, and the performance efficiency α_x^E also varies as the time and machine state changing. Thus, it is quite difficult to measure the performance and performance efficiency α_x^E for each machine individually. In the simulation, we can obtain the practice agent executing time, which is calculated by formula (19), and then we define the average service state time as follows:

$$T_{\text{AST}} = \frac{T_{\text{execute}}}{C_n} = \frac{1}{Cn} \cdot \sum_{i=1}^{C_n} \frac{n_{y_i}^i + 1}{P_{y_i}^{E_i} \cdot \alpha_{y_i}^{E_i}} \tag{30}$$

In the agent-executing process, high-performance machines that the agents run will spend less time than the low performance machines, and the T_{AST} will also be less. On the other hand, the same agent substate executing in a different machine will cause different time consumption because of the performance overhead of the machines. Then, the T_{AST} also can be viewed as the characteristic to measure with respect to the machine performances.

The multiple agent experiments focus on two issues. One issue is the relationship between the average service availability and the number of FS-MA. The other issue is the influence of the number of FS-MAs on the network. The details of the parameter settings are specified in Table I. In multiple agent experiments, we load the same VIRGO platform at each time and use a three-state MA, with the average service state time of $T_{\text{AST}} = 25$ s. Although a huge number of agents are executing, the resources access competition cannot be void. The resource access competition is the condition under which two or more agents request the same resource node at the same time. Under this condition, the dispatching policy for agent is desired. In our multiple FS-MA system, we adopt a first-come-first-serve policy.

Figure 6 presents the relationship between the number of agents and the average service availability. The results show that the average service availability will not decrease significantly; however, the number of agents increases greatly. Even under the worst condition, where the agent number equates to the node number of the VIRGO platforms, the service availability is acceptable. Figure 6 also shows that the agent size will affect the service availability significantly.

Figure 7 shows the relationship between the average agent executing time and the number of agents with different agent sizes. As observed from the diagram, the average agent executing time grows slightly when the number of agents increases. According to Figures 6 and 7, we can see that the FS agent system will maintain stable performance when the number of agents grows. Thus, the FS-MA system can achieve a perfect performance in both the system's scalability and the executing efficiency when a large number of services are executing all together.

In Figure 8, we show the relationship between the number of agents and the average aggregate bandwidth. Because the average aggregate bandwidth can be viewed as the workload measurement of the whole network, we can conclude that the number of agents increasing greatly will not add to the workload of the whole network notably.

In the FS-MA system, the agent migration length (hops) is another important measurement for agent performance. Figure 9 shows that the agent migration length will remain stable when the

DOI: 10.1002/cpe

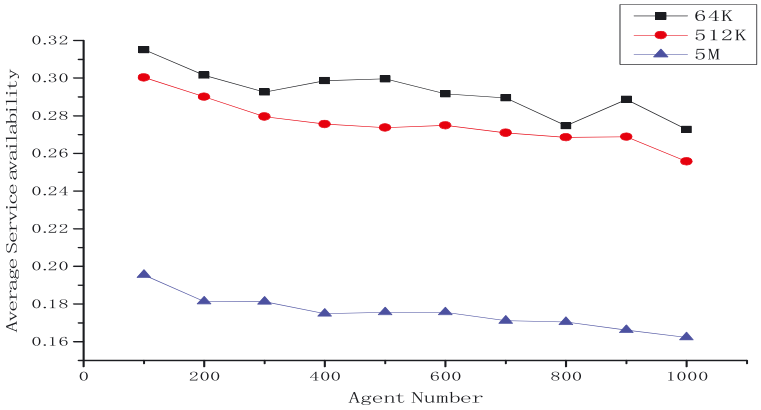

Figure 6. Agent number versus average service availability (agent size = 64 K, 512 K, 5 M).

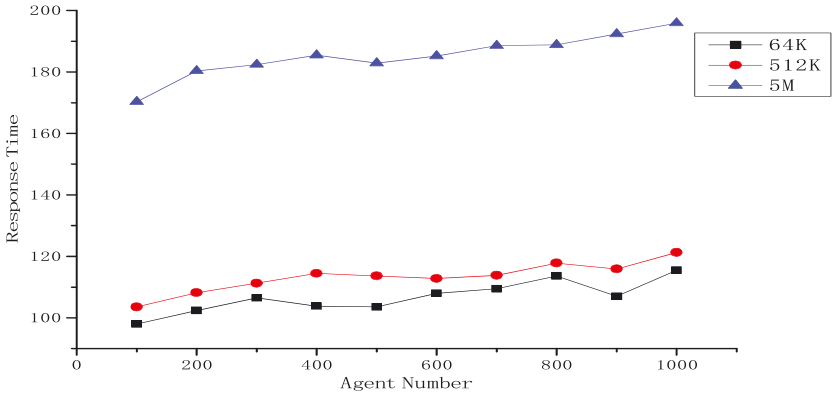

Figure 7. Agent number versus response time (agent size = 64 K, 512 K, 5 M).

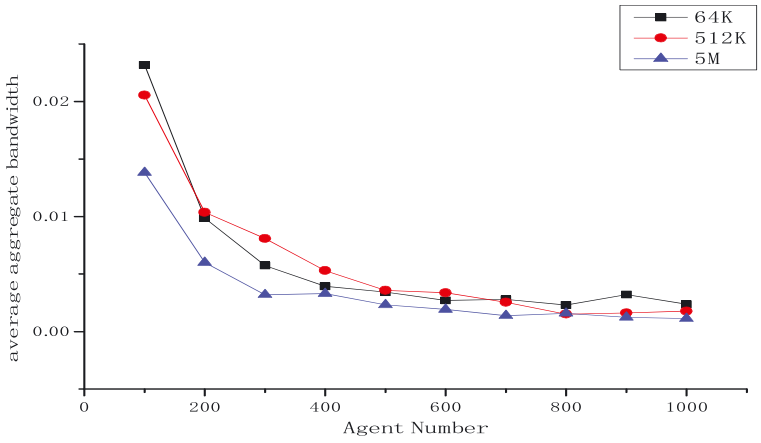

Figure 8. Agent number versus average aggregate bandwidth (agent size = 64 K, 512 K, 5 M).

number of agents changes; it also suggests that the migration length will depend on the resource distributions rather than on the number of agents.

DOI: 10.1002/cpe

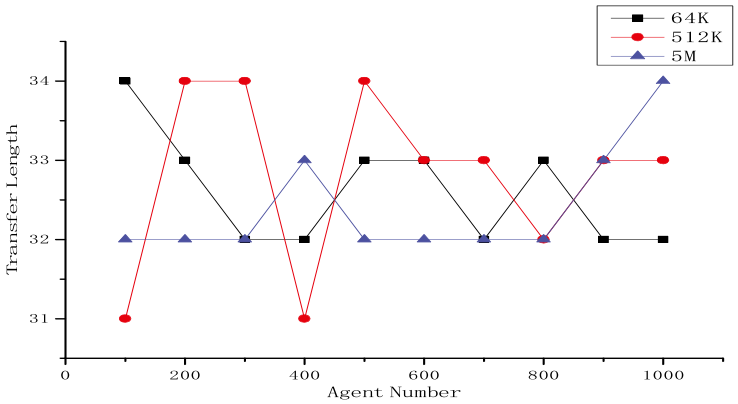

Figure 9. Agent number versus transfer length (agent size = 64 K, 512 K, 5 M).

Table II. Parameters and corresponding parameter relationships between the mobile agent (MA) approach and the client/server (C/S) approach.

Parameters	MA	C/S
Task transfer data	Agent size	Request data size
Composing of task	Substates of MA	Services binding in nodes
Task number	Agent number	Requestor number
The time to finish the task	Total agent executing time	Total services response time
The efficiency of finishing the task	Services availability	Total service time/total response time

4.2. Comparison experiments between the agent approach and the client/server approach

Because the agent approaches are always introduced to the SOA architecture to improve the performance and adaptively of such systems [37], we also implement our performance model to evaluate the performance between the agent approach and the C/S approach. To evaluate how effective the MA approach is compared with the traditional C/S approach, we designed one comparison experiment to explore the performances of these two approaches. In the experiment, we implemented the same workload task by these two approaches. The benchmark task consists of four subtasks and a request for four different resources. In the MA approach, the benchmark task can be presented as a substate MA. In the C/S-based SOA approach [5], four web services bind in four different servers, and the client should accomplish the task by requesting these four services in sequence.

In this experiment, these two approaches adopt the same network topology, and both approaches complete the same number of tasks. Although the node distributions of the two approaches are the same, we add several additional service binding nodes in the C/S approaches, which are deployed randomly. The details of the parameters in the experiment and the corresponding parameter relationships between the two approaches are shown in Table II.

The experimental results are shown in Figures 10–12. We compare the time to finish the task and the efficiency of finishing the task for the two approaches with different agent sizes (64 KB, 512 KB, and 5 MB). In the figures, MA presents the performance of the MA approach, and the ‘concurrency 25’ presents the performance of the C/S approach when the maximum concurrencies of the servers is 25. From the figures, we can observe the following:

(i) The number of requestors in the C/S approach will greatly affect the task finishing time and the task finishing efficiency. As the number of requestors increases, the response time of the C/S approach increases sharply, and the efficiency of the C/S approach decreases sharply. In contrast to the C/S approach, the MA approach might not be affected by the number of agents, and there is only a slight increase in the task finishing time and a decrease in the service availability when the number of agents increases.

DOI: 10.1002/cpe

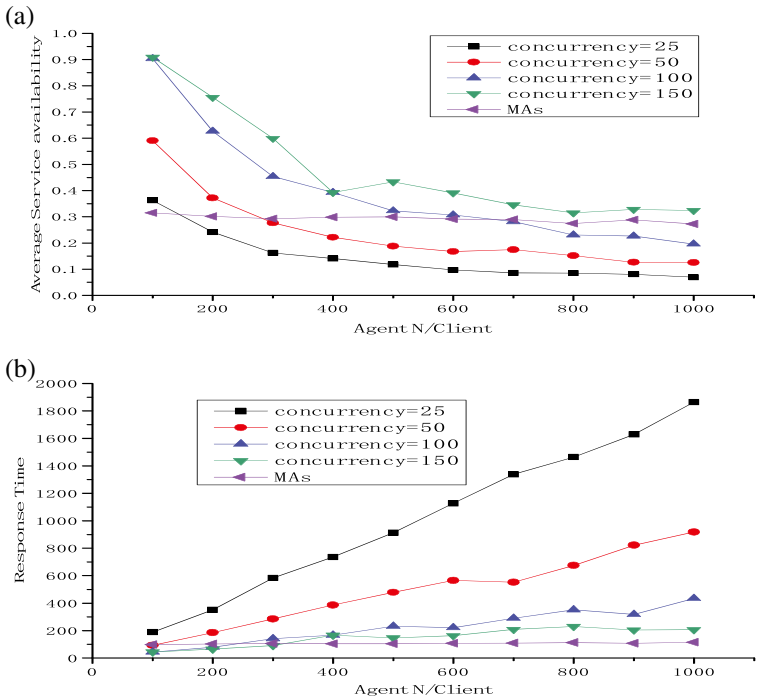

Figure 10. Comparison between the mobile agent (MA) approach and the client/server approach when the task transfer data size is 64 KB. (a) Agent number/client versus availability and (b) Agent number/client versus response time.

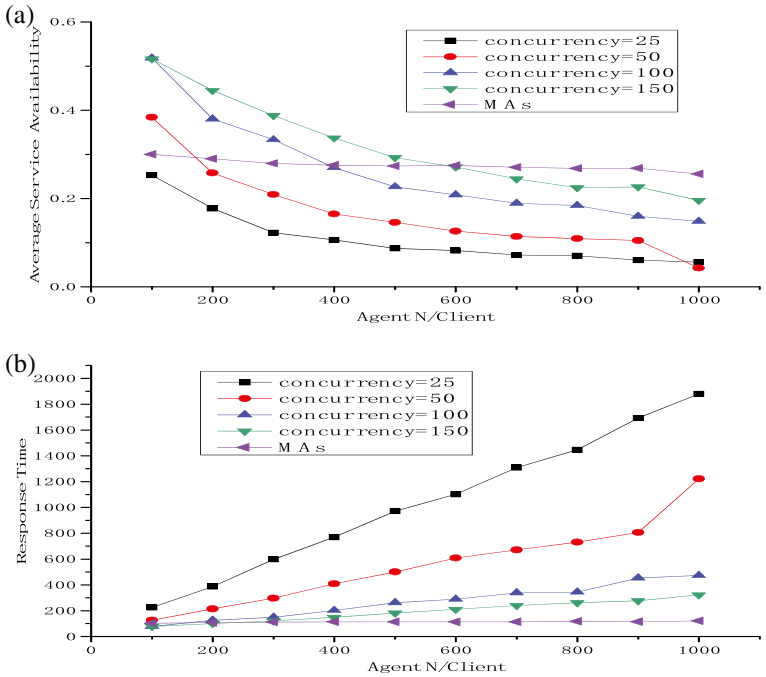

Figure 11. Comparison between the mobile agent (MA) approach and the client/server approach when the task transfer data size is 512 KB. (a) Agent number/client versus availability and (b) Agent number/client versus response time.

DOI: 10.1002/cpe

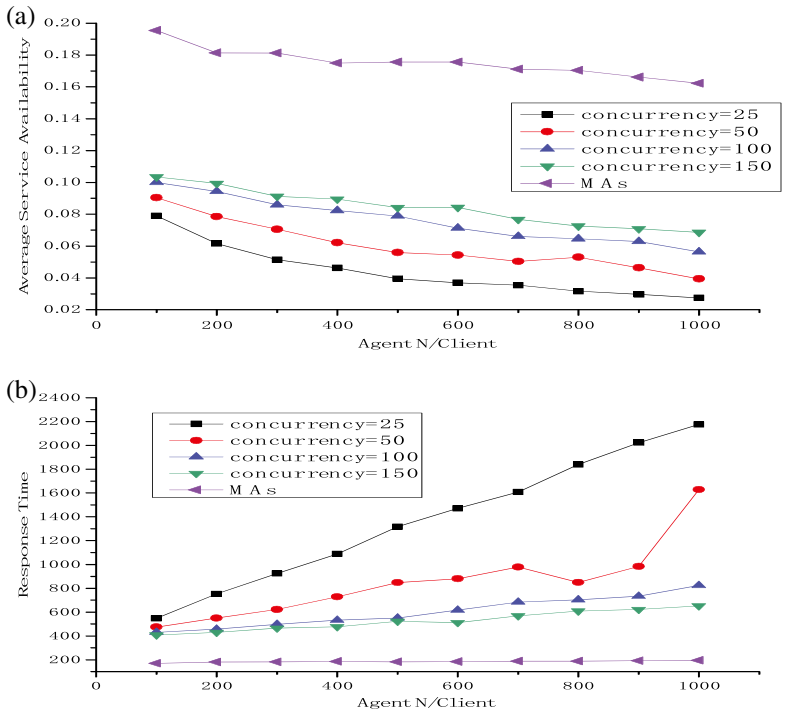

Figure 12. Comparison between the mobile agent (MA) approach and the client/server approach when the task transfer data size is 5 MB. (a) Agent number/client versus availability and (b) Agent number/client versus response time.

(ii) Increasing the maximum concurrencies of the servers in the C/S approach will improve its performance efficiently. However, the cost of promoting the maximum concurrencies of the servers is quite expensive in practice. On the other hand, the super-servers with higher concurrencies, in practice, may be idle most of the time, which causes a waste of resources.

(iii) Comparing Figures 10–12, we can see that the performance of the C/S approach will decrease greatly as the task transfer data size becomes large. Meanwhile, the MA approach will retain a similar performance when the transferring data size becomes bigger; so, the MA will be more efficient for distributed cases.

5. CONCLUSION

A FS-MA approach based on virtual hierarchical architecture—VIRGO—has been presented in this article. To evaluate the performance of this approach, the evaluation model of MA on VIRGO has also been proposed, and the experiments in multiple agents executing simulation show the capability of our prototype. The relationships among the service availability, the total executing time, the average service state time, the migration time, the average aggregate bandwidth, and the number of agents are discussed. The results indicate that the FS-MA system will maintain the acceptable performance (the average aggregate bandwidth is quite low, and both the average total agent's service executing time and average service availability are stably holding at satisfied values) when the number of agents increases sharply. We also conducted comparable experiments between the MA-based approach and the C/S-based approach. The results also indicate that the MA-based approach is superior to the C/S-based approach.

The experimental results show that the performance evaluation model presented here is reasonable. We plan to investigate a large-scale environment, such as millions of nodes in VIRGO, to further evaluate the model.

DOI: 10.1002/cpe

The full tree search could encounter heavy traffic in the nodes within root layers of the VIRGO [6] architecture. In the future, we plan to use resource classification on VIRGO [6] to solve the problem.

We will use more parameters involved in the performance evaluation process of agent systems in future studies, such as different parameters for VIRGO, policies of migration and interoperation [37, 38], and the system parameters, that is, the workloads of memory, CPU, and I/O. Furthermore, according to the simulation results, we plan to integrate MA into a VIRGO project [34] and to implement it in a cloud computing architecture [2].

ACKNOWLEDGEMENT

This research is based upon work supported in part by National Natural Science Foundation of China (61173123) and Natural Science Foundation of Zhejiang Province (Z1100822,Y1101237). We would like to deliver our sincere thanks to the anonymous reviewers for their constructive suggestions in improving our paper.

REFERENCES

1. Papazoglou MP, van den HW. Blueprinting the cloud. *IEEE Internet Computing* 2011; **15**(6):74–79. DOI: 10.1109/MIC.2011.147.
2. Brian H. Cloud computing. *Communication of ACM* 2008; **51**(7):9–11. DOI: 10.1145/1364782.1364786.
3. Michael A, Armando F, Rean G, Anthony DJ, Randy HK, Andy K, Gunho L, David AP, Ariel R, Ion S, Matei Z. A view of cloud computing. *Communication of ACM* 2010; **53**(4):50–58. DOI: 10.1145/1721654.1721672.
4. Huang LC, Wu ZH, Pan YH. Virtual and dynamic hierarchical architecture for E-science grid. *International Journal of High Performance Computing Applications* 2003; **17**(3):329–347. DOI: 10.1177/1094342003173007.
5. Huang LC. A P2P service discovery strategy based on content catalogues. *Data Science Journal* 2007; **6**:S492–S499. DOI: 10.2481/dsj.6.
6. Huang LC. VIRGO: Virtual hierarchical overlay network for scalable grid computing. In *Proceeding of the European Grid Conference (EGC2005),* February 2005, Vol. 3470. Springer: Berlin, 2005; 911–921.
7. Christensen JH. Using RESTful web-services and cloud computing to create next generation mobile applications. In *Proceedings of the 24th ACM SIGPLAN Conference Companion on Object Oriented Programming Systems Languages and Applications (OOPSLA)*. ACM: New York, NY , USA, 2009; 627–634.
8. Chun BG, Ihm S, Maniatis P, Naik M, Patti A. CloneCloud: elastic execution between mobile device and cloud. In *Proceedings of the Sixth Conference on Computer Systems (EuroSys),* Salzburg, Austria, 2011. ACM: New York, 2011; 301–314.
9. Ong KL, Zhang Z, Ng WK, Lim EP. Agents and stream data mining: a new perspective. *IEEE Intelligent Systems* 2005; **20**(3):60–67. DOI: 10.1109/MIS.2005.39.
10. Danny BL, Mitsuru O, Günter K, Kazuya K. Aglets: programming mobile agents in Java. *Lecture Notes in Computer Science* 1997; **1274**:253–266. DOI: 10.1007/3-540-63343-X_52.
11. Nestinger S, Chen B, Cheng HH. A mobile agent based framework for flexible automation systems. *IEEE/ASME Transactions on Mechatronics* 2010; **15**(6):942–951. DOI: 10.1109/TMECH.2009.2036169.
12. Theilmann W, Rothermel K. Optimizing the dissemination of MAs for distributed information filtering. *IEEE Concurrency* 2000; **8**(2):53–61. DOI: 10.1109/4434.846194.
13. Vu AP, Ahmed K. Mobile software agents: an overview. *IEEE Communication Magazine* 1998; **36**(7):26–37. DOI: 10.1109/35.689628.
14. Marios DD, Melinos K, George S. Performance evaluation of mobile-agent middleware: a hierarchical approach. In *Proceeding of the 5th International Conference on Mobile Agents,* Atlanta, GA , USA , December 2001, Vol. 2240. Springer: Berlin, 2002; 244–259.
15. Sergio I, Eduardo M, Arantza I. A system based on MAs to test mobile computing applications. *Journal of Network and Computer Applications* 2009; **32**(4):846–865. DOI: 10.1016/j.jnca.2009.01.003.
16. Chung YF, Chen TS, Lai MW. Efficient migration access control for mobile agents. *Computer Standards & Interfaces* 2009; **31**(6):1061–1068. DOI: 10.1016/j.csi.2008.09.039.
17. Elena GM, Sergio I, José M. Performance analysis of MAs tracking. In *Proceedings of the 6th International Workshop on Software and Performance (WOSP)*. ACM Press: New York, 2007; 181–188.
18. Ismail L, Hagimont D. A performance evaluation of the mobile agent paradigm. In *ACM SIGPLAN Notices*, Vol. 34(10). ACM Press: New York, 1999; 306–313.
19. Kotz D, Cybenko G, Gray RS, Jiang G, Peterson RA, Hofmann MO, Chacón DA, Whitebread KR, Hendler JA. Performance analysis of mobile agents for filtering data streams on wireless networks. *Mobile Networks and Applications* 2002; **7**(2):163–174. DOI: 10.1023/A:1013778922814.
20. Marios DD, George S. Performance evaluation of mas: issues and approaches. In *Performance Engineering: State of the Art and Current Trends*, Vol. 2001. Springer: Berlin, 2001; 148–166. DOI: 10.1007/3-540-45156-0_10.

21. Robert SG, David K, Ronald AP, Joyce B, Daria AC, Peter G, Martin OH, Jeffrey MB, Maggie RB, Renia J, Niranjan S. Mobile-agent versus client/server performance: scalability in an information-retrieval task. In *Proceeding of the 5th International Conference on Mobile Agents (MA),* Atlanta, GA, USA, December 2001, Vol. 2240. Springer: Berlin, 2002; 229–243.
22. Robert SG, George C, David K, Ronald AP, Daniela R. D'Agents: applications and performance of a mobile-agent system. *Software: Practice and Experience* 2002; **32**(6):543–573. DOI: 10.1002/spe.449.
23. Strasser M, Schwehm M. A performance model for mobile agent systems. In *Proceeding of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA),* June 30 – July 3, 1997, Vol. 2. CSREA Press: USA; 1132–1140.
24. Trappey CV, Trappey AJ, Huang CJ, Ku C. The design of a JADE-based autonomous workflow management system for collaborative SoC design. *Expert Systems with Applications* 2009; **36**:2659–2669. DOI: 10.1016/j.eswa.2008.01.064.
25. Woodside M. Scalability metrics and analysis of MA systems. In *International Workshop on Infrastructure for Multi-Agent Systems: Infrastructure for Agents, Multi-Agent Systems, and Scalable Multi-Agent Systems*, Vol. 1887. Springer-Verlag: London, UK, 2001; 234–245.
26. Erdem T, Mehmet HG, Mustafa Y, Selahattin K. Performance analysis of mobile agents using simulation. In *Proceedings of the Advanced Engineering Design Conference (AED2003)*. Prague: Czech Republic, 2003; 111-122.
27. Vasileios B, Miltiadis K, Stathes H, Lazaros FM. Performance evaluation of a mobile agent-based platform for ubiquitous service provision. *Pervasive and Mobile Computing* 2008; **4**(5):755–774. DOI: 10.1016/j.pmcj.2008.07.003.
28. Trillo R, Ilarri S, Mena E. Comparison and performance evaluation of mobile agent platforms. In *Third International Conference on Autonomic and Autonomous Systems (ICAS'07),* Athens (Greece). IEEE Computer Society: Washington, DC, USA, 2007; 41–46. DOI: 10.1109/CONIELECOMP.2007.66.
29. Johansen D. Mobile agent applicability. In *Proceedings of the Second International Workshop on Mobile Agents(MA'98),* September 1998. Springer: Stuttgart, Germany, 1998; 80–98.
30. Kpper A, Park AS. Stationary vs. mobile user agents in future mobile telecommunications networks. In *Proceeding of the Second International Workshop on Mobile Agents (MA'98),* September 1998. Springer: Stuttgart, Germany, 1998; 112–123.
31. Puliato A, Riccobene S, Scarpa M. An analytical comparison of the client-server, remote evaluation and MAs paradigms. In *The First International Symposium on Agent Systems and Applications and Third International Symposium on MAs (ASA/MA99),* October 1999. IEEE Computer Society Press: Los Alamitos, CA, 1999; 278.
32. Rahul J, Sridhar I. Performance evaluation of mobile agents for E-commerce applications. *8th International Conference on High Performance Computing*, Hyderabad, India, 2001; 331–340. DOI: 10.1007/3-540-45307-5_29.
33. Liu Y, Xu CF, Wu ZH, Pan YH. A finite state mobile agent computation model. In *Proceeding of 6th Asia-Pacific Web Conference on Advanced Web Technologies and Applications (APWeb),* Hangzhou, China, April 2004, Vol. 3007. Springer: Berlin, 2004; 152–157.
34. VIRGO (2010) HomePage of VIRGO. (Available from: http://virgo.sourceforge.net) [accessed 22–26 April 2001].
35. Dovrolis C, Ramanathan P, Moore D. What do packet dispersion techniques measure? *Proceeding of Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies(INFOCOM2001)*, Anchorage, Alaska, USA, 2001; 905–914.
36. Harfoush K, Bestavros A, Byers J. Measuring bottleneck bandwidth of targeted path segments. *Proceeding of Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)*, San Franciso, CA, USA, March 2003; 2079–2089.
37. Michael NH, Munindar P S, Mark HB, Keith SD, Edmund HD, Timothy WF, Les G, Hrishikesh JG, Nicholas RJ, Kiran L, Hideyuki N, Parunak HVD, Jeffrey SR, Alicia R, Gita S, Samarth S, Katia PS, Milind T, Thomas W, Rosa LZG. Research directions for service-oriented multiagent systems. *IEEE Internet Computing* 2005; **9**(6):65–70. DOI: 10.1109/MIC.2005.132.
38. Giancarlo F, Alfredo G, Wilma R. Achieving MA systems interoperability through software layering. *Information and Software Technology* 2008; **50**(4):322–341. DOI: 10.1016/j.infsof.2007.02.016.