On the Dynamic Allocation of Mobile Agents
by On-line Task Scheduling

Xinfeng Yang1,2, Weimin Ma1, Jane You1,2, James Liu1
1Department of Computing, The HK Polytechnic University, Kowloon, Hong Kong
2School of Computing and Information Technology, Griffith University, QLD, Australia, 4111
{X.Yang , J.You}@cit.gu.edu.au, {cswmma , csnkliu}@comp.polyu.edu.hk

Abstract
This paper presents a new approach to the dynamic allocation of mobile agents by on-line task scheduling for high performance in Internet computing. In contrast to the existing approaches, which apply pre-defined task scheduling schemes to allocate computing resources, we introduce a new on-line competitive algorithm to achieve flexibility. A guided allocation scheme based on the system competitive ratio is developed to optimize the allocation of mobile agents. The proposed system combines push-based technology with an innovative on-line task-scheduling scheme to speed up system response time and minimize system overheads. The system analysis and simulation demonstrate the feasibility and effectiveness of our approach to mobile agent-based Internet computing.

1 Introduction
With the rapid development of information technology and the widespread popularity of the Internet and World Wide Web, the mobile agent paradigm is becoming increasingly important for network-based applications. It is viewed as a new trend in distributed artificial intelligence research including information gathering, data mining, workflow and electronic commerce [1]. In general, a mobile agent is regarded as a program which represents a user in the network and is capable of migrating autonomously from one host machine to another [2]. Mobile agents can also be viewed as a mechanism for introducing parallel activities by concurrent execution. According to [2], the design of mobile agent systems involves several key issues such as the provision of code mobility, object naming, portability, scalability and security. The mobile agent paradigm offers a variety of research topics, ranging from low-level system administration tasks to user-level applications.

Task scheduling in mobile agent system design has received much attention in recent years. In the work of Henrik Stormer [3], an agent-based workflow system is developed where a workflow-instance agent is applied for task scheduling. The workflow agent identifies which task is to be executed next for the optimal operation. The main advantage of this approach is that there is no central server involved during the workflow running. However, the workflow needs to be pre-defined and the performance of the system depends on the definition of the workflow. Zakaria Maamar [1] introduces an approach to optimizing a mobile agent itinerary. A broker agent is created to coordinate resource allocation among agents. The set of links that has the minimized value of network traveling from the first task to the last task will be the optimized itinerary. This system presents a group of intelligent agents with cooperating and task scheduling capabilities, but such capability depends on complete knowledge about the resources location and cost, which have to be specified beforehand with a link or node value. A utility-driven mobile agent-scheduling scheme is discussed in the paper of Jonathan Bredin [4]. In such a system, the agents are equipped with information about a resource over the network. A demand function is defined to guide the resource allocation for efficiency. Unfortunately, quantitative information about resources and resource consumption relies heavily on traditional microeconomic estimation, which to some extent limits the system’s flexibility in the real electronic market.

In the research described above, the advantages of using mobile agents are featured as mobility, portability and scalability. Task scheduling plays an important role in mobile agent system design for high performance. Much of the current work has been
focused on allocating resources to mobile agents intelligently and managing business processes automatically while mobile agents are generated statically to fulfill a given task or an off-line request sequence. However, there has been little work on dynamically allocating mobile agents in an on-line fashion for Internet-based applications. It is essential to introduce a new system structure on dynamic allocation of mobile agents by on-line task scheduling to address the limitations of the current approaches and achieve flexibility.

In this paper, we propose a multi-agent system structure enhanced by push-based technology and an on-line task-scheduling algorithm. Mobile agents will be created and cloned dynamically, initialized with service units and pushed from the remote site to a local site which is more convenient for local clients to access, thus speeding up the system response. Such new architecture is initially being proposed in response to communication asymmetry exhibited by many applications, such as software distribution, news delivery, and traffic information systems. In these environments, communication from the clients to the server is more restricted than communication from the server to the clients, so it may make more sense to push service from the servers without waiting for the clients to pull it. In the proposed system design, attention will be paid to satisfying clients’ requests as soon as possible and minimizing system-handling time while not knowing what the future requests will be. All potential costs will be identified, and an on-line task-scheduling algorithm will be sought.

This paper is organized as follows. Section 2 outlines the mobile agent computing paradigm and Section 3 highlights the proposed structure for the multi-agent system. An on-line task-scheduling algorithm for dynamically allocating mobile agents is introduced in Section 4 and the experimental results are reported in Section 5. Finally, the conclusion is presented in Section 6.

2 Mobile Agent Computing Paradigm

Compared with earlier paradigms such as process migration or remote evaluation in distributed computing, the mobile agent model is becoming increasingly popular for network-centric programming. The traditional client/server paradigm relies on the handshake mechanism to communicate over a network. The client requests information, while the server responds. Each request/response has to be a complete round trip on the network. The emerging mobile agent paradigm has provided a dynamic and flexible platform for software development and redefined the way Internet-based applications work. As an autonomy software entity with pre-defined functionality and certain intelligence, the mobile agent is capable of migrating autonomously from one host machine to another, making its request to the server directly and performing tasks on behalf of its master. Some of the advantages of this model are better bandwidth usage, reliable network connection and reduced design work.

During the past few years, more than a dozen Java-based mobile agent systems have been developed. The Java Virtual Machine, the standard security manager and two other functional facilities, namely remote method invocation and object serialization have made it simple to build a mobile agent workbench. Of all these available Java-based systems, ObjectSpace’s Voyager [5], General Magic’s Odyssey [6] and IBM’s Aglet [7] are three leading commercial ones. A detailed discussion of current commercial and research-based mobile agent systems can be found in [8].

Voyager [5] is a Java-based mobile agent development platform supported by ObjectSpace. It is the first platform to smoothly integrate traditional distributed computing with cutting edge agent technology. The virtual object is the core facility and framework in Voyager to support inter-agent communication and mobile agent migration. It can migrate not only between agent servers, but also to Java runtimes of other arbitrary virtual objects, which is a feature not found among other existing mobile agent systems. However, use of the virtual object brings complexity as well, so that most developers have to change their traditional way of building distributed applications in order to make use of this new feature. General Magic’s Odyssey [6] is another pure Java-based mobile agent development platform. It utilizes Java RMI as well as CORBA and DCOM protocols for mobile agent transport, and provides a collaboration facility to allow agents’ meeting at particular host on the network. However, in such a system, a rmiregistry name server must be installed on every machine with an Odyssey agent server, which increases the complexity of the system’s maintenance. The Aglet system [7] developed by IBM is chosen as the implementation example in our proposed system. Although it is not a full-fledged platform at this point, it has received the most press coverage and shows promise as a functional technology that fits very well into the Java world. Several phrases can be summarized to characterize an aglet: lightweight objects migration, built with persistent support, event driven, etc.
As a fundamental component, Java Aglet API (JAAPI) makes the design of an aglet system very clean. It not only supports the initialization of an aglet but also all services for aglet migration and inter-aglet communication. From a technical point of view, an aglet is a composite Java object with its own thread of execution, so an aglet’s mobility in migrating from one host to another is implemented using Java’s object serialization mechanism. Inter-aglet communication is established through message passing, which can be one-way, synchronous or future-reply. AgletContext is the execution environment for an aglet running which is much like AppletContext for an applet. The aglet also has a well-planned life cycle [9]. Between the starting and stopping stages an aglet can experience many events, such as creation, cloning, dispatching, disposal, etc. Figure 1 illustrates four of the fundamental operations of an aglet.

**Figure 1: Four of the fundamental operations of an aglet**

*Creation* is a primary way of producing a new aglet in the current context; its state is initialized at this stage. *Cloning* is an important way of creating a copy of the original aglet in the same context. The cloned aglet has the same state but a different identifier from the original. *Dispatch* is an active way to ship an aglet from one host to another. Upon arriving at the new host, the aglet restarts its execution. *Disposal* is a useful way of controlling the number of aglets residing in a context and reducing resources consumption. Only these four operations will be implemented in our proposed system structure. In addition, *retraction* provides a way to draw back an aglet from a remote host to the home site. *Deactivated and activated* are ways to push an aglet out of the current context temporarily or bring it back. When an aglet migrates from node to node across the network, it carries both code and state with it until it returns back to its original host, which is different from the way an applet works.

### 3 Multi-agent System Structure

To achieve flexibility and efficiency, we propose a multi-agent system which includes two major modules:

- A remote system module working as a remote agent server that hosts three kinds of agents: stationary agents, mobile agent controllers and mobile service agents.
- A local system module working as a local agent server hosting two kinds of agents: mobile service agents and client agents.

Figure 2 shows the general structure of the system. It also includes two interfaces: the JDBC interface resides at the remote server side to retrieve service units from the backend database; the user interface resides at the local server side to accept incoming requests that have been grouped according to each client’s request quantity.

**Figure 2: Multi-agent system structure enhanced by on-line task-scheduling algorithm**
The following summarizes the major functionality of the proposed system: service registration, service preparation, service consumption and service completion.

- **Service registration**

Service registration is the first step. When a client wants remote service, it is required to register at the remote service provider and to indicate how many service units are needed. The remote service provider keeps a client list all the time. This list is subject to change frequently due to the increase or decrease in client requests. All potential clients are divided into several groups according to each client’s request quantity. Based on different request ranges, different groups of clients will be guided to access different local agent hosts. In our proposed system, there are totally \( n \) groups of clients available. Clients in Group\(_1\) with requests ranging \([m, M]\) will be directed to Local Agent Host 1 (\(LAH_1\)) and obtain service units there.

- **Service preparation**

Speeding up system response is one of our aims. In the proposed system, our strategy is to create and clone a number of mobile agents at the remote agent host dynamically, initializing with service units and dispatching these agents to the local agent hosts beforehand. To serve \( n \) groups of clients, \( n \) Mobile Agent Controllers (\(MAC\)) are needed. In our system, \(MAC_i\) creates and clones a number of worker agents. These ready-made Mobile Service Agents (\(MSA\)), each equipped with one service unit, are dispatched to \(LAH_i\) to be accessed locally.

- **Service consumption**

At this stage, the client will access the local agent host instead of the remote agent host to enjoy quick service. When a client (agent) with \(x\) requests consumes ready-made \(d\) service units carried by \(d\) MSAs, three scenarios are most likely to occur: (1) \(x = d\): the client happens to consume all ready-made service units; (2) \(x < d\): the ready-made service units are more than client’s requests. Unclaimed units (agents) will be destroyed later; (3) \(x > d\): the numbers of ready-made service units are not enough. Therefore, the client agent sends a message to the remote agent controller asking for more. After receiving the message, the controller will repeat the cloning and dispatching processes to serve the unsatisfied requests.

- **Service completion**

This is the last step in the whole process. The remote agent controller \(MAC\) will be informed of the service completion after the client’s requests have been successfully satisfied. One major concern at this stage is for the unclaimed mobile service agents (\(MSA\)) left at the local host, which have to be disposed of explicitly before the new service process starts. For the service provider, the new process will begin at the preparation step.

The design of the proposed system structure is straightforward. However, a problem arises when the remote agent controller prepares for service units. How many mobile service units (agents) need to be created, cloned and dispatched before it is known which client will ask for service next time? If the number of prepared service units is not enough, the agent controller has to clone more and supply later, which will delay the system response time. If the number of prepared service units is greater than that of the client’s requests, the unclaimed service agents have to be disposed of explicitly, which will increase system overheads. In the proposed system, both shorter overheads and better handling performance need to be taken into account. The following section presents an on-line algorithm addressing the above problem, which we call the On-line Task Scheduling Problem (\(OTSP\)).

4 Competitive Algorithm for On-line Task Scheduling

This section presents an on-line competitive algorithm for dynamic task scheduling. On-line fashion can be found in many research areas such as data structuring, task scheduling or resource allocation [10], [11]. In a data-structuring problem, it is desired to access elements in a given data structure at lower cost, without knowing which element will have to be accessed in the future. For the paging problem in a two-level memory system, it is necessary to decide which referenced pages have to be taken out of fast memory while not knowing which page will be requested next. In a multi-processor network, it is a challenging job to serve the accesses in the network with lower communication costs through dynamically realocating files but without any knowledge of future accesses. All these problems are characterized by satisfying requests or making decisions before knowing what the future requests will be, which is different from traditional system analysis where the complete sequence of requests is known before designing an algorithm.
An extensive study of the on-line problem started in the mid eighties when Sleator and Tarjan [12] suggested that the competitive algorithm is a powerful tool for addressing on-line problems. In a study of the competitive snoopy caching problem [13], researchers compared the performance of an on-line algorithm (A) to an optimal off-line strategy (OPT). Taking a measurement of the cost, their study focused on minimizing the worst case ratio of the on-line cost to the optimal off-line cost. For any possible sequence of request σ, let C_A(σ) and C_OPT(σ) denote on-line cost and off-line cost respectively, the algorithm A is called α-competitive and the competitive ratio for the algorithm. Competitive algorithm A is efficient if constant α is small.

In this paper, we integrate the competitive algorithm with task scheduling for allocating mobile agents. Our aim is to minimize the total cost of service in an on-line environment. Firstly, we measure each service with computing cost, for instance, Cost (creation) denotes the cost of the average creation service. Thus, five types of costs are involved in our proposed system: Cost (creation), Cost (cloning), Cost (dispatching), Cost (disposal) and Cost (messaging).

For three scenarios when a client with (x) requests consumes ready-made (d) service units, we have

- \( x = d \): Client happens to consume all ready-made service units (agents). In this scenario, each service agent costs \( c = \text{Cost (creation)} + \text{Cost (cloning)} + \text{Cost (dispatching)} \), the total cost will be \( cd \) or \( cx \).

- \( x < d \): The number of ready-made service units is more than that of the client’s request, so \( (d - x) \) units (agents) have to be disposed of. For those agents, we have \( c_1 = \text{Cost (creation)} + \text{Cost (cloning)} + \text{Cost (dispatching)} \), so that the total cost will be \( cx + c_1(d - x) \).

- \( x > d \): The number of ready-made service units is less than that of the client’s request, so a further \( (x - d) \) units (agents) will be reproduced and dispatched. For these agents, we have \( c_2 = \text{Cost (messaging)} + \text{Cost (cloning)} + \text{Cost (dispatching)} \), so that the total cost will be \( cd + c_2(x - d) \) or \( cx + (c_2 - c)(x - d) \).

If the actual request sequence is denoted with \( \sigma = (x_1, x_2, ..., x_n) \), where \( x_i \) denotes the actual request number on the \( i \)th time, we can get the optimal off-line cost of the problem as follows:

\[
C_{OPT}(\sigma) = c \cdot \sum_{i=1}^{n} x_i
\]  

(2).

For the same request sequence, if \( d_i \) represents the service units that should be prepared by the on-line decision-maker on the \( i \)th time, we can get the on-line cost of competitive algorithm A as follows:

\[
C_A(\sigma) = c \cdot \sum_{i=1}^{n} x_i + (c_2 - c) \sum_{i=d_i}^{n} (x_i - d_i) + c_1 \sum_{i=d_i}^{n} (x_i - d_i)
\]  

(3).

For any on-line algorithm \( A \), the competitive ratio is defined as

\[
\alpha = \inf_{\sigma} \frac{C_A(\sigma)}{C_{OPT}(\sigma)}
\]  

(4).

A small competitive ratio implies that \( A \) can do well in comparison with \( OPT \). For the given \( OTSP \) problem, the agent controller (on-line decision-maker) doesn’t know which client asks for service, namely, the actual request number on the \( i \)th time, but is aware of the possible request range of any given group, which is denoted by \( [m, M] \). The on-line competitive algorithm \( A \) should give the best choice \( d \) that needs to be prepared for the \( i \)th time with the smallest competitive ratio \( \alpha \).

### 4.1 General Harmonic Algorithm (GHA)

**Theorem 1:** For the on-line \( OTSP \) problem, the best choice that an on-line decision-maker should make is

\[
d = \frac{Mm \cdot (p + q - 1)}{Mp + m \cdot (q - 1)}
\]  

(5).

where \( p = c_1 / c \), \( q = c_2 / c \) and \( [m, M] \) is the possible requests range of any registered group.

The proof will be given with Theorem 2.

### 4.2 Competitive Ratio of GHA

**Theorem 2:** For the competitive \( GHA \) algorithm given in Theorem 1, the competitive ratio is

\[
\alpha = 1 + p(q - 1) \frac{M - m}{Mp + m \cdot (q - 1)}
\]  

(6).
Proof. To prove Theorem 2, we need to consider two possible worst request sequences, which an off-line adversary may choose, namely: \( \sigma_M = (M, M, \ldots, M) \) and \( \sigma_m = (m, m, \ldots, m) \).

Firstly, letting \( C_A(\sigma_M) \) and \( C_{OPT}(\sigma_M) \) denote the on-line cost and optimal off-line cost under the circumstance of \( \sigma_M \) respectively. According to formula (3) and (5), we have

\[
C_A(\sigma_M) = c \cdot \sum_{i=1}^{n} M + (c_2 - c) \cdot \sum_{i=1}^{n} (M - d_i)
= cMN + (c_2 - c) \cdot (M - d) \cdot n
= \left( 1 + (q-1) \left( 1 - \frac{d}{M} \right) \right) \cdot cmn
= \left( 1 + (q-1) \left( 1 - \frac{d}{M} \right) \right) \cdot C_{OPT}(\sigma_M)
= \left( 1 + p(q-1) \frac{M - m}{Mp + m \cdot (q-1)} \right) \cdot C_{OPT}(\sigma_M)
\]

Similarly, letting \( C_A(\sigma_m) \) and \( C_{OPT}(\sigma_m) \) denote the on-line cost and optimal off-line cost under the circumstance of \( \sigma_m \), we have

\[
C_A(\sigma_m) = c \cdot \sum_{i=1}^{n} m + c_1 \cdot \sum_{i=1}^{n} (d_i - m)
= (cm + pc \cdot (d - m)) \cdot n
= \left( 1 + p \left( \frac{d}{m} - 1 \right) \right) \cdot cmn
= \left( 1 + p \left( \frac{d}{m} - 1 \right) \right) \cdot C_{OPT}(\sigma_m)
= \left( 1 + p(q-1) \frac{M - m}{Mp + m \cdot (q-1)} \right) \cdot C_{OPT}(\sigma_m)
\]

\( \sigma_M \) and \( \sigma_m \) are the two possible worst cases in the given problem, so it seems that, for any \( \sigma \),

\[
\alpha = \frac{C_A(\sigma)}{C_{OPT}(\sigma)} \leq 1 + p(q-1) \left( \frac{M - m}{Mp + m \cdot (q-1)} \right)
\]

The proof is completed.

### 4.3 Lower Bound for the Competitive Ratio

**Theorem 3:** For the competitive ratio of the on-line GHA algorithm, \( 1 + p(q-1) \frac{M - m}{Mp + m \cdot (q-1)} \) is the lower bound.

**Proof.** We need to prove that if the on-line player chooses another number, e.g. \( d' \neq \frac{Mm \cdot (p + q - 1)}{Mp + m \cdot (q-1)} \), the competitive ratio will get worse. Firstly, we assume that \( d' < \frac{Mm \cdot (p + q - 1)}{Mp + m \cdot (q-1)} \), we then need to prove that if the off-line adversary chooses \( \sigma_M \), the competitive ratio \( 1 + p(q-1) \frac{M - m}{Mp + m \cdot (q-1)} \) cannot be achieved.

According to formula (3), we have

\[
C_A(\sigma_M) = c \cdot \sum_{i=1}^{n} M + (c_2 - c) \cdot \sum_{i=1}^{n} (M - d_i)
= (cm + (q-1) \cdot c \cdot (M - d')) \cdot n
= \left( 1 + (q-1) \left( 1 - \frac{d}{M} \right) \right) \cdot cmn
> \left( 1 + p(q-1) \frac{M - m}{Mp + m \cdot (q-1)} \right) \cdot C_{OPT}(\sigma_M)
\]

The last inequality holds if \( d' > \frac{Mm \cdot (p + q - 1)}{Mp + m \cdot (q-1)} \).

Similar proof can be given when condition \( d' > \frac{Mm \cdot (p + q - 1)}{Mp + m \cdot (q-1)} \) happens.

**Corollary 1:** If we denote the fluctuation rate as \( \phi = M / m \), for the competitive GHA algorithm given in Theorem 1, the best choice \( (d) \) depends on \( p / q \) while the fluctuation rate remains constant.

**Proof.** According to formula (5), we have

\[
\frac{d}{m} = \frac{\phi p + \phi \cdot (q-1)}{\phi p + (q-1)}
\]

That means \( d \to m \) when \( p \gg q \), and \( d \to M \) when \( q \gg p \).

In the proposed system, if the cost of disposal is much higher than that of reproduction, the number we choose will approach the lower boundary \( m \). By contrast, our choice will approach the upper boundary \( M \) if
the cost of reproduction is much higher than that of disposal. This conclusion is illustrated in Figure 3.

Corollary 2: For a special case of the on-line OTSP problem, if we have \( c_1 = c_2 = c \), namely, \( p = q = 1 \), the best choice that an on-line player can make will be

\[
\hat{d} = \frac{2Mm}{M + m}
\]

where \([m,M]\) is the possible requests range of any registered group.

The proof is omitted here.

Corollary 3: For the special case of the competitive GHA algorithm given in Corollary 2, the competitive ratio is

\[
\alpha = 1 + p \frac{M - m}{M + m}
\]

The proof is omitted here.

### 5 Experimental Results

To evaluate the system performance, we apply the IBM Aglet as our implementation example. Under the simulation, creating one stationery agent takes \( t_{\text{creation}} = 110\text{ms} \), both cloning and destroying one mobile agent take \( t_{\text{cloning}} = t_{\text{disposing}} = 30\text{ms} \). For sending one message or dispatching one group of agents, we made an assumption that \( t_{\text{messaging}} = t_{\text{dispatching}} = 60000\text{ms} \). Three groups with clients’ requests ranging between \([30,50]\), \([30,75]\) and \([30,100]\) are examined. Request sequences are generated randomly. Results are shown in Table 1 after 100 trials.

In Table 1, it can be noted that the competitive ratios \( \alpha \) within any group are fluctuant but not beyond the theoretical lower bounds given in formula (6), which are 1.248, 1.42 and 1.52 respectively.

<table>
<thead>
<tr>
<th>( \alpha )</th>
<th>Group1 ([30,50])</th>
<th>Group2 ([30,75])</th>
<th>Group3 ([30,100])</th>
</tr>
</thead>
<tbody>
<tr>
<td>10 Trials</td>
<td>1.168</td>
<td>1.151</td>
<td>1.411</td>
</tr>
<tr>
<td>20</td>
<td>1.157</td>
<td>1.217</td>
<td>1.326</td>
</tr>
<tr>
<td>30</td>
<td>1.162</td>
<td>1.225</td>
<td>1.287</td>
</tr>
<tr>
<td>40</td>
<td>1.158</td>
<td>1.265</td>
<td>1.243</td>
</tr>
<tr>
<td>50</td>
<td>1.164</td>
<td>1.281</td>
<td>1.307</td>
</tr>
<tr>
<td>60</td>
<td>1.147</td>
<td>1.286</td>
<td>1.322</td>
</tr>
<tr>
<td>70</td>
<td>1.139</td>
<td>1.275</td>
<td>1.305</td>
</tr>
<tr>
<td>80</td>
<td>1.145</td>
<td>1.266</td>
<td>1.311</td>
</tr>
<tr>
<td>90</td>
<td>1.153</td>
<td>1.271</td>
<td>1.323</td>
</tr>
<tr>
<td>100</td>
<td>1.142</td>
<td>1.258</td>
<td>1.318</td>
</tr>
</tbody>
</table>

Table 1: Experimental results after 100 trials

From the experiments, we get two other corollaries.

**Corollary 4:** For the on-line OTSP problem where \( p = c_1 / c \), \( q = c_2 / c \) and \( \phi = M / m \), competitive ratio \( \alpha \) depends on \( \phi \) if \( p \) and \( q \) are constants. When \( \phi \) gets bigger, \( \alpha \) approaches \( q \).

**Proof.** According to formula (6), we have

\[
\alpha = 1 + (q - 1) \frac{\phi p - p}{\phi p + (q - 1)}
\]

That means \( \alpha \to 1 \) when \( \phi \to 1 \), and \( \alpha \to q \) when \( \phi \to \infty \).

\( \phi \) is the fluctuation rate of requests within any registered group, so if we divide our clients into more groups, thus narrowing the possible requests’ fluctuation rate of each group, we can always get a better competitive ratio. This conclusion is illustrated in Figure 4.
Corollary 5: For the on-line OTSP problem where $p = c_1 / c$, $q = c_2 / c$ and $\phi = M / m$, if the fluctuation rate changes due to the increase or decrease of clients’ requests within the group, the on-line solution $(d)$ changes accordingly through dynamic allocation.

$$d = \frac{Mp + M \cdot (q - 1)}{\phi p + (q - 1)}$$ (15).

The proof is omitted here.

The relation between $d / m$ and $\phi$ is illustrated in Figure 5.

![Figure 5: Relationship between $d / m$ and $\phi$](image)

6 Conclusion

Task scheduling schemes play an important role in mobile agent system design. The majority of current research has been focused on applying pre-defined schemes for allocating computing resources to fulfill a fixed off-line task. To overcome the limitations of current approaches, which lack the flexibility of dynamically allocating mobile agents in an on-line environment, we introduce a new system structure which integrates push-based technology with an innovative on-line task scheduling algorithm to speed up system response time and minimize system overhead. The derived competitive algorithm GHA (General Harmonic Algorithm) with the optimal competitive ratio has been tested through experiments, demonstrating its feasibility and effectiveness. The system will have potential wide Internet-based applications with the integration of other intelligent decision-making algorithms for high performance with flexibility and reliability.

7 Acknowledgement

The authors would like to acknowledge the partial support of Central Research Grants G-T375 of The Hong Kong Polytechnic University.

References


