A Novel Priority-based Deadlock Detection and Resolution Algorithm in Mobile Agent Systems

Wei Lu[1], Yong Yang[1,*], Liqiang Wang[2], Weiwei Xing[1], Xiaoping Che[1]
[1] School of Software Engineering
Beijing Jiaotong University
Beijing, China
[2] Department of Computer Science
University of Central Florida
Orlando, USA
Email: [1]{luwei,12112088,wwxing,xpche}@bjtu.edu.cn
[2]{lwang}@cs.ucf.edu

Abstract

Deadlock detection and resolution is one of the challenges in mobile agent systems, especially, when concurrent execution (i.e., more than one algorithm instances executing simultaneously) of algorithm instances. In this paper, we propose a deadlock detection and resolution algorithm in mobile agent systems. Priority-based approach is adopted in our algorithm to coordinate concurrent execution of algorithm instances. The liveness and safety properties of our algorithm are proved. Analysis and simulation results indicate that our algorithm can provide better performance and avoid duplicate detection and resolutions of the same deadlock in condition of concurrent execution.

Deadlock detection; Deadlock resolution; Mobile-Agent system; Distributed system; Concurrent coordination

1 Introduction

Rapidly expanding of available on-line information increases the need for bandwidth to support the industry infrastructure and bandwidth optimization [13]. For finding solutions of high bandwidth demand, many schemes are proposed in the past years.

Especially, Remote Procedure Calls (RPC) [7], Remote Programming (RP) and Code on Demand [3] are proposed to reduce network load in distributed systems before the emergence of mobile agent. Mobile agent with property of mobility greatly enhances the productivity of each computing element in the network and creates a uniquely powerful computing environment. A mobile agent acts as a self-contained software element responsible for executing a programmatic process, which is capable of autonomously migrating through a network.

There is no standard definition of mobile agent (some definitions can be found in literature [4, 10, 11, 13, 20]). Even so, we can obtain some common characteristics of mobile agent as summarized in [5]: mobility, autonomy, sociality, reactivity, proactivity, data acquisition, and route determination, etc.

Despite of many practical benefits, mobile agent technology also introducing new challenges (e.g., deadlock, rendezvous, and leader election). In addition, problem will become more complicated when concurrency error and considered [9, 14, 15]. In this paper, we propose a priority-based deadlock detection and resolution algorithm that can support concurrent execution of algorithm instances and provides better performance in mobile agent systems. Our algorithm can be roughly divided into two phases: decentralized information collection phase and centralized deadlock detection and resolution phase.

The paper is organized as follows: in Section 2, we review some related works. Description of our algorithm is presented in Section 3. Section 4 proves our algorithm. Section 5 illustrates the theoretical analysis and simulation results. Section 6 is the conclusions and future works.

2 Related Works

Some related works on deadlock detection and resolution in mobile agent systems as [1, 2, 6, 8, 16, 18, 19, 21].

Ashfield proposed a path pushing similar approach to detect deadlock in mobile agent system in [1, 2]. A consume

DOI reference number: 10.18293/DMS2016-014

agent creates a shadow agent to monitor its activities when it requests an exclusive lock on a resource. Shadow agent will create detection agent to perform deadlock detection if the consume agent has been block for a predetermined time interval. Detection agent responds for constructing WFG and detecting deadlock until a deadlock is detected by visiting related shadow agents.

In [8], Hosseini et. al, presented an improved algorithm based on Ashfield [1, 2]. They assigned a priority to both resource and consumer agent, and the detection agent created by consume agent with the least priority will suspend other detection agents that have larger priority. Therefore, only one detection agent (the lowest priority one) will pass through the whole cycle and can return.

Different with [1, 2], Elkady proposed an edge chasing similar algorithm to detect deadlock in mobile agent system in [6]. The proposed algorithm asserts a deadlock when a detection agent visiting a consume agent twice. In addition, the proposed algorithm supports deadlock avoidance and multiple detection agents.

Yang proposed two path pushing based deadlock detection algorithms named: MA-WFG and Host-WFS in [21]. In MA-WFG, mobile agent trying to construct a WFG to detect loop structure. Local construct WFG will pass to the mobile agent which locks the required resource to construct its local WFG. A deadlock is detected if there is loop topology in the collected WFG. To reduce the number of agent movements and relieve the network load, Host-WFS makes the WFG arranged in a form of "wait for set" that are distributed and stored on hosts. Therefore, the hosts control the termination of the path pushing, and mobile agent need not participate in the operation of path pushing.

Mani and Moshirpour, et.al, proposed an UML model based approaches to detect deadlock in Multi-Agent manufacturing systems in [16, 18]. They propose a behavioral model in form of UML 2.0 sequence diagrams from the modeling artifacts of the Multi-agent Software Engineering (MaSE) [17] to analysis and detect deadlock in multi-agent systems. Firstly, agents task diagrams are built during the analysis and designing of the MaSE methodology. The diagrams illustrate the activities performed by several cells (e.g., machines and robots). Each task diagram is a UML-like state chart diagram and will be converted to an UML activity diagram. Control Flow Paths (CFP) will be derived from the task diagrams. Then, resource requirement information can be gathered from each CFP that will be map into a machine requirement table. The table records the information and tasks of each agent with CFP and required machines (resources) of each CFP. During the run-time, each CFP that is running by an agent can initiate deadlock detection to query when blocked for a predefined time interval. The query will be sent to its direct depended CFP and propagated to indirect CFPs. The initiating CFP will determine itself deadlocked if it never receives a message from other CFPs to inform changing of the blocked status.

There are some limitations in previous works, such as: either no local deadlock detection or infinite deadlock detection, probability of false results or state explosion, performance inefficiency, high complexity, etc. To improve the performance efficiency and provide better solution in condition of concurrent execution. We propose a priority-based deadlock detection and resolution algorithm with better concurrent execution supporting in this paper.

3 Proposed Approach

3.1 Premises and Assumptions

Some properties in mobile agent systems as follows.

Network organization independence: neither the nodes nor the agents need to maintain knowledge about the size or topology of the network. The deadlock detection and resolution algorithm should not depend on a particular organization of the nodes in a mobile agent system.

Agent movement: agents must be allowed to lock resources and move for performing additional tasks.

Fault tolerance: messages or steps could be lost and gracefully recovery when it occurs.

Limit coupling: mobile agents that consume and manipulate resources in a mobile agent system should be separated from the deadlock detection and resolution process.

Based on the above properties of the mobile agent systems, we give some assumptions as:

- Agents can move through the network without lost, and network topology keeps static once the deadlock detection algorithm starts.
- A consume agent can choose to blocked when its resource lock request is rejected, or neglect the rejection and perform other tasks.
- There is no phantom deadlock based on the using of standard deadlock avoidance techniques, such as 2PC (two phase commit) or priority transactions.
- Host environment is the ultimate authority that can allow or deny the lock request of a resource.
- Host environment will be notified if a consume agent moves to another host environment.
- Agents and host environments are uniquely identified in the system through techniques such as static path proxy or naming service.
- Each consume agent can request at most one resource at a time, and will not request resource any more when blocked. It means the resource request model in this paper is single request model [12].

3.2 Algorithm Overview

We apply priority-based and edge chasing technique, that are commonly used in traditional deadlock detection algorithm in the distributed systems, in our algorithm.

Three kinds of mobile agents and the host environment are used in our algorithm descried as following. If a CA creates a DA or a RA, we call the CA hosting agent and the DA or RA spawned agent.

- **CA**, consume agent that can performs common tasks and exclusively locking needed resources. It spawns DA(s) and RA(s) to initiate deadlock detection and publish deadlock resolution. It does not have an active role in deadlock detection and resolution procedure. It reports its new location to the HE(s) that host resources locked by this CA when it move to a new HE.

- **DA**, detection agent that is spawned by CA and responsible for deadlock detection and resolution. It visits HEs to collect information of related CA(s) and construct global WFG. It responsible for detecting and resolving deadlocks according to the global WFG.

- **RA**, resolution agent spawned by CA used to notify the deadlock resolution to the victim CA.

- **HE**, host environment hosts mobile agents and provides APIs to hosted agents to interactive with itself. It controls and coordinates resource locking and unlocking. It provides information of hosted resources and mobile agents to DAs in deadlock detection activity.

3.2.1 Algorithm Initiation

A CA is assigned an unique id value when it is created in the mobile agent systems. It moves to a HE and requests resource to this HE when the needed resource is hosted by the HE. HE has the authority to grant or reject the request from CA. HE rejects the request if the resource is being locked by another CA and notifies the failure of resource requesting to the requesting CA. A CA can choose to blocked (waiting for the request resource to be unlocked) or discard this request to perform other tasks. The CA transfers from active to blocked state if it chooses blocked and wait.

The blocked CA initiates a deadlock detection and resolution algorithm instance by creating a DA after being blocked for an explicit time interval. The spawned DA has unique priority with the same value as the id of its hosting CA. Spawned DA starts distributed information collection phase by moving to the HE that hosts the needed resource and asking for the location of the CA that is locking this resource. Note that, it is possible that more than one DAs that roaming in the mobile agent system at the same time.

3.2.2 Deadlock Detection

The DA communicates with the HE to collects the resource locking information (i.e., wait-for relationship) of a CA, and determines whether continuing its visiting or not according to the situation of current visiting CA. A CA can not create a DA after been visited by other DA(s) until its request resource is granted.

A CA may have two kinds of situations when a DA arriving the HE that hosts this CA:

- Case1: the CA **has not** initiated a deadlock detection algorithm instance yet (e.g., no DA has been created by this CA when its HE is visited);

- Case2: the CA **has been** initiated a deadlock detection algorithm instance by creating a DA, and the detection process has been not terminated.

There are some possible sub-conditions in Case1 and Case2 of this CA:

- Case1-1: it is in an **active** state (e.g., this CA is not blocked by any resource at this moment);

- Case1-2: it is in a **blocked** state, but has been not initiated a deadlock detection instance by creating DA.

- Case2-1: its spawned DA has a **larger** priority than the priority of visiting DA.

- Case2-2: its spawned DA has a **lower** priority than the priority of visiting DA.

- Case2-3: its spawned DA has a **same** priority with the priority of visiting DA.

In Case1-1, the visited CA is in active state. In this condition, the visiting DA will determine that there is no deadlock and return to its hosting CA with information of visited CA(s). The hosting CA will make the decision of no deadlock and wait for locked resource to be available.

In Case1-2, different with Case1-1, the visited CA is blocked for resource that is locked by another CA. The visiting DA will move to the next HE. This procedure will continues till one of the following conditions is meet. One is that a current visited CA is in a active state like Case1-1, another is that this DA visits a CA that has initiated an algorithm instance by spawning a DA with a different priority, this falls into the cases of Case2.

In Case2-1, the visiting DA visits a CA that has been initiated an algorithm instance by spawning a DA with a higher priority. In this case, the visiting DA adds information of the visited CA into its visiting list and return to its hosting CA. The hosting CA will determine whether performing a second round of detection or not according to the visiting list collected by its returning DA.

In Case2-2, the visiting DA will not return until the visited CA receives the returning of its own spawned DA. The visiting DA combines its own visiting list and the visiting list of the visited CA together, and then, returns to its hosting CA with the combined visiting list.

In Case2-3, this condition represents the existence of deadlock when a visiting loop forms.

In Case1-1, Case1-2, and Case2-3, an explicit decision of deadlock can be made. However, there needs some other steps to continues the deadlock detection in Case2-1 and Case2-2 after returning of spawned DA(s) with visiting list.

After the returning of spawned DA(s) in Case2-1 and Case2-2, the hosting CA(s) will locally record the visiting list and make determination of a second round detection. A CA will launch the next round of detection if the following two conditions are satisfied:

- Condition-1: this CA has the largest id among the CAs in the local visiting list;
- Condition-2: this CA has been visited by other DAs.

The spawned DA will execute a next round of visiting with a same priority as the last round. In addition, the target visiting CA is the rear CA of the collected visiting list. The rest procedure is the same as Case2-1 or Case2-2 till a explicit decision of deadlock can be make, then, the detection process terminates.

3.2.3 Deadlock Resolution

A global WFG can be constructed after the first phase terminating. The second phase starts detecting and resolving deadlock if there exists a deadlock. Proposed algorithm can guarantee that only the DA with the highest priority will detect the deadlock at last. This DA will select a victim CA to release its locked resource to break the deadlock (simply, itself). The hosting CA which spawn a RA and send this RA to the victim CA for deadlock resolution. The victim CA releases the locked resource after receiving the RA. Then, the deadlock is resolved, and all blocked CA(s) can perform the next tasks.

3.3 Formal Description

Formal description of our algorithm is given in this section. Variables used in our algorithm are given in Table 1.

Algorithm 1 illustrates that how to handle the arriving of a detection agent in a visited CA (the possible cases in the previous subsection). Algorithm 2 gives the description of the strategy when a DA returning to its hosting CA.

3.4 Case Study

As shown in Figure 1,there are 6 HEs ($HE1 - HE6$), 6 CAs ($CA1 - CA6$), and 15 resources ($R1 - R15$) in a

Table 1: Variables used in the proposed algorithm.

Variable	Descriptions
CA_i	Consume agent with a unique $id = i$, the hosting CA of its spawned DA_i.
DA_i	Detection agent created by CA_i with a priority i, spawned DA of CA_i.
DA_i^{VL}	Visiting list recorded in DA_i. It is ordered in visited order. Such as,$\{CA_1, CA_2, ..., CA_k\}$ means the visiting order is $CA_1 \rightarrow CA_2 \rightarrow, ..., \rightarrow CA_k$.
VL_i	Visiting list recorded at CA_i.
RA_i	Resolution agent created by a consume agent with $id = i$. It is used to notify the victim consume agent of deadlock resolution.

Algorithm 1 when DA_i arriving at CA_j

$VL_j = DA_i^{VL} \cup VL_j$;
$DA_i^{VL} = DA_i^{VL} \cup VL_j$;
if CA_j has initiated a deadlock detection instance **then /*Case 2*/**
    if $DA_j < DA_i$ **then /*** ***Case 2-1*****/**
        waiting till the returning of DA_j;
        $DA_i^{VL} = DA_i^{VL} \cup VL_j \cup DA_j^{VL}$;
        DA_i return to CA_i with DA_i^{VL};
    else if $DA_j > DA_i$ **then /*** ***Case 2-2*****/**
        $DA_i^{VL} = DA_i^{VL} \cup VL_j$;
        DA_i return to CA_i with DA_i^{VL};
    else/* ***Case 2-3*****/**
        a deadlock is detected;
        creating a RA_i and sending RA_i to the CA that is selected as victim by DA_i to resolve the deadlock;
    end if
else/*Case 1*/
    if CA_j is active **then /*** ***Case 1-1*****/**
        $DA_i^{VL} = DA_i^{VL} \cup \{CA_j \rightarrow \varnothing\}$;
        DA_i return to CA_i with DA_i^{VL};
    else/* ***Case 1-2*****/**
        $DA_i^{VL} = DA_i^{VL} \cup \{CA_j \rightarrow CA_k\}$;
        DA_i moves to the next CA_k;
    end if
end if

Algorithm 2 when DA_i returning to CA_i with DA_i^{VL}

$VL_i = VL_i \cup DA_i^{VL}$;
if there is cycle in VL_i and DA_i has the largest priority among all priority values of collected CAs **then**
    a deadlock is detected;
    creating a RA_i and sending RA_i to the victim CA;
else
    if DA_i is the largest in VL_i and HE of CA_i has been visited by other DA **then**
        DA_i moves to CA_j;
    else
        while the requested resource is not granted **do**
            waiting the granting of resource;
            if receiving a RA_i **then**
                releasing the locked resource, and break the while loop;
            end if
        end while
    end if
end if

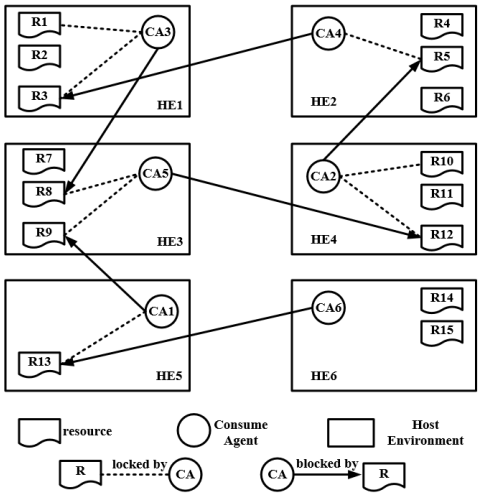


Figure 1: Instantaneous state of a mobile agent system.

mobile agent system. 8 of the 15 resources are locked. Assuming that Figure 1 is a transient state of the mobile agent system. Take HE1 in Figure 1 for example, HE1 holds three resources ($R1$, $R2$,and $R3$) and a Consume Agent ($CA3$). $CA3$ has locked resource $R1$ and $R3$, and is blocked by $R8$ which is locked by $CA5$ locates at $HE3$.

Assuming that Consume Agents $CA1$, $CA2$, $CA3$, $CA4$, $CA5$, and $CA6$ have been blocked for a specific time interval. Therefore, they will spawn DAs to execute deadlock detection and resolution. We also assume that the time of a DA's movement is finite but uncertain, and the DAs will be not lost or tampered.

One possible scenario is all consume agents creating detection agent and initiating deadlock detection simultaneously. In this condition, detection agents $DA1$, $DA2$, $DA3$, $DA4$, $DA5$, and $DA6$ will visit $HE5$, $HE4$, $HE1$, $HE2$, $HE3$, and $HE6$ to collect information.

$DA1$ will return to $HE1$ immediately with a new resource request (i.e., wait-for) information of $CA5$ ($\{CA5 \rightarrow CA2\}$) when $DA1$ arrives at $HE3$ before returning of $DA5$. Because $DA1$ has a lower priority than $DA5$ that is spawned by $CA5$. $DA2$ and $DA3$ returns with ($\{CA4 \rightarrow CA3\}$) and ($\{CA5 \rightarrow CA2\}$). Assuming that $DA4$ arrives at $HE1$ and returns to $CA4$ before the returning of $DA3$, therefor, $DA4$ will collect the information of $CA3$ as ($\{CA3 \rightarrow CA5\}$) and $DA6$ return after the returning of $DA2$ and $DA1$, the visit lists of $DA5$ and $DA6$ will be ($\{CA2 \rightarrow CA4 \rightarrow CA3\}$) and ($\{CA1 \rightarrow CA5 \rightarrow CA2\}$), respectively.

The local visiting list in each consume agent are illustrated in Table 2 after the above steps.

Only $DA5$ will launch a second round of detection, because the id of it is the largest among CAs in the local visiting list. $DA5$ will move to $HE3$ for collecting wait-for information of $CA3$. The other consume agents will suspend to wait for the detection and resolution results. As the same rule in the first round, the local visiting list will be changed as shown in Table 3 after the second round returning of detection agents

Table 2: Local visiting list in CAs after returning of their spawned DAs.

VL_i	**Local visiting list**
VL_1	$\{CA1 \rightarrow CA5 \rightarrow CA2\}$
VL_2	$\{CA2 \rightarrow CA4 \rightarrow CA3\}$
VL_3	$\{CA3 \rightarrow CA5 \rightarrow CA2\}$
VL_4	$\{CA4 \rightarrow CA3 \rightarrow CA5\}$
VL_5	$\{CA5 \rightarrow CA2 \rightarrow CA4 \rightarrow CA3\}$
VL_6	$\{CA6 \rightarrow CA1 \rightarrow CA5 \rightarrow CA2\}$

Table 3: Local visiting list in CAs after returning of their spawned DAs.

VL_i	**Local visiting list**
VL_1	$\{CA1 \rightarrow CA5 \rightarrow CA2\}$
VL_2	$\{CA2 \rightarrow CA4 \rightarrow CA3\}$
VL_3	$\{CA3 \rightarrow CA5 \rightarrow CA2\}$
VL_4	$\{CA4 \rightarrow CA3 \rightarrow CA5\}$
VL_5	$\{CA5 \rightarrow CA2 \rightarrow CA4 \rightarrow CA3 \rightarrow CA5\}$
VL_6	$\{CA6 \rightarrow CA1 \rightarrow CA5 \rightarrow CA2\}$

A deadlock is detected by $DA5$ after its returning to $CA5$ according to the local visiting list. $DA5$ will select a victim CA that is involved the deadlock from the local visiting list. Then, deadlock resolution task will be conducted by $CA5$ by spawning a $RA5$ and sending it to the victim CA(e.g., $CA2$). $CA2$ will release the lock on resource $R12$ after receiving of $RA5$, and the deadlock will be broken.

4 Correctness Proofs

In this section, we prove the liveness and safety property of our algorithm. In addition, some other necessary theorems are also proved.

Theorem 1 *At any given time, there is at most one deadlock in a WFG.*

Proof 1 *Since the resource request model is single request model, the number of outgoing arc at each node is one at most. Assuming that there are more than one deadlock in the collection WFG. It means at least two cycles in the WFG and they are either independent or intersection with each other. Due to the single request model, it is impossible that two cycles are intersection. So, they should be detected in separated WFG rather than in one WFG. Therefore, there is at most one deadlock in one WFG at any given time.*

Table 4: Comparison of the worst performance in aspect of concurrent execution.

	[2]	[8]	[21]^1	[21]^2	[6]	Our
number of DA moves in detection	∞	$\frac{n(1+n)}{2}$	$\geqslant \frac{n(1+n)}{2}$	$\geqslant \frac{n(1+n)}{2}$	n^2	$4(n-1)$
number of RA moves in resolution	$-$	1	$-$	$-$	$\geqslant n^2$	1
number of transmitted CA information	∞	$\frac{n^2(1+n)}{2}$	$\geqslant \frac{n^2(1+n)}{2}$	$\geqslant \frac{n^2(1+n)}{2}$	$\frac{n^2(1+n)}{2}$	$\frac{n(1+n)}{2}$
Time delay	∞	n	n	n	n	$n+1$

[21]^1 and [21]^2 are the two algorithm proposed in literature [21]; $-$: no deadlock resolution or difficult to analysis theoretically; n: number of detection agent in a WFG.

Theorem 2 *If a deadlock exists, only one DA will detect it and one CA will resolve it.*

Proof 2 *According to Algorithm 1, only DA with the highest priority will collected the whole WFG and perform final deadlock detection. Therefore, only the hosting with the largest value of id will resolve a detected deadlock. The theorem is proved.*

Theorem 3 *No phantom deadlock will be detected and resolved.*

Proof 3 *The phantom deadlock appears when two deadlock detection and resolution algorithm instance attempt to perform on the same CA. By Theorem 2, there can be a maximum of one algorithm instance will detect and resolve a deadlock. Hence, there is no possibility for a phantom deadlock in our algorithm.*

Theorem 4 *(Liveness) If there is a deadlock in the system, it should be detected and resolved in finite time.*

Proof 4 *Let's assume that there is a deadlock in the system and not be detected when at least one of the CA initiate deadlock detection procedure. There are two possible situations, one is that the DA lost during its travel. This is not possible duo to the assumption that DA will not lost in the system. Another one is no deadlock detects when a DA with the highest priority collected WFG. This contradicts the assumption. Therefore, if the movement of agent is in finite time, the deadlock will be detected and resolved in finite time.*

Theorem 5 *(Safety) If a algorithm instance resolves a deadlock in the system, it is in fact in the system and resolved by only one algorithm instance.*

Proof 5 *By Theorem 3, we can prove that the detected deadlock actually in the system. By Theorem 1 and Theorem 2, we can derive that the detected deadlock will be detected and resolved by only one algorithm instance. The theorem is proved.*

5 Performance Analysis

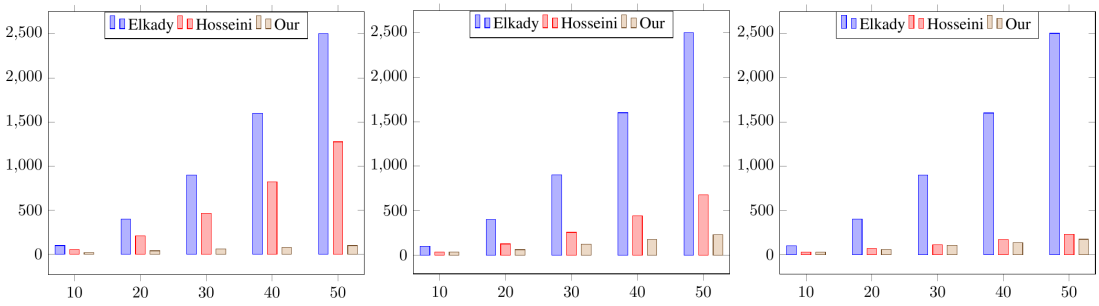
In this section, we analysis the worst case of performance among previous and our algorithms, and perform simulation each algorithm. It should be noted, only the situation of concurrent execution is considered in the analysis. This analysis contains number of DA moves in detection, number of RA moves in resolution, total number of transmitted CA information and time delay. Specifically, one piece of transmitted CA information means information of one CA (e.g., $CA_i \rightarrow CA_j$ as one piece information of CA_i).

5.1 Theoretical Analysis

In this theoretical analysis, we assume that n consume agents (CA_1, CA_2,...,CA_n) involved in a deadlock, and all consume agents initiate deadlock detection and resolution algorithm instances with arbitrary time order. Our algorithm has the worst case of DA movements in Table 5.B and the worst case of transmitted CA information in Table 5.A. In Table 5.B, the first round of deadlock detection will has $2n$ DA moves. The second round has $2 * (\frac{n}{2})$ DA moves and the i_{th} round has $2 * \frac{n}{2^{(i_{th}-1)}}$ DA moves. This procedure continues till $1 \leqslant \frac{n}{2^{(i_{th}-1)}} < 2$. Therefore, number of DA moves in detection is $2 * \frac{2*(1-2^{log_2^n})}{1-2} = 4(n-1)$. In Table 5.A, each DA_i will return to CA_i with $i-1$ pieces of CA (these CAs has lower id than i) information. Therefore, total number of transmitted CA information is $\frac{n(1+n)}{2}$.

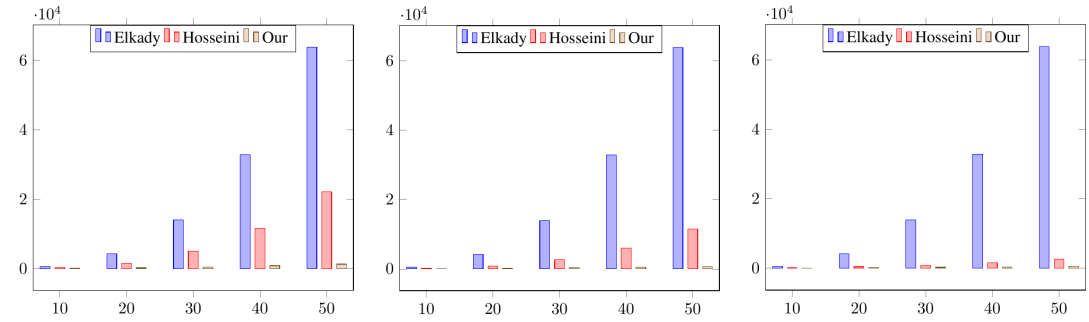
The DA with the highest priority performs deadlock detection after constructing WFG, and it will select a victim from the deadlocked CAs. Then, its hosting CA will create a resolution agent (RA) and send it to the victim consume agent. Hence, the resolution transmits only one time of RA to resolve the deadlock. If the priority of each DA in the cycle within a decreasing order and the least priority waits for the highest one, the worst case of time delay in our algorithm is n.

Based on the same criteria, Table 4 illustrates performance analysis results (in the worst case) between previous works and our work.



(a) Number of DA movements in Table 5.A. (b) Number of DA movements in Table 5.B. (c) Number of DA movements in Table 5.C.

Figure 2: Performance comparison on detection agent movement between some previous works and our work.



(a) Totle number of transmitted CA information transmitted in Table 5.A. (b) Totle number of transmitted CA information transmitted in Table 5.B. (c) Totle number of transmitted CA information transmitted in Table 5.C.

Figure 3: Performance comparison on information transmission between some previous works and our work.

5.2 Simulation Results

The simulation program is written in Scala (version 2.11.7) with Akka (version 2.3.11). We compares two previous works (Elkady and Hosseini [6, 8]) with our work, and run the simulation on three types of scenes as shown in Table 5. The program of each algorithm runs 100 times, and the result is the average value.

Table 5: Description of possible scenes in the simulations.

Type	Description
A	Each agent requests a resource that is locked by another agent, wait-for relationship as: $n \rightarrow n-1 \rightarrow ... \rightarrow 1 \rightarrow n$;
B	Each agent requests a resource that is locked by another agent, wait-for relationship as: $n \rightarrow 1 \rightarrow n-1 \rightarrow 2... \rightarrow \lfloor \frac{n}{2} \rfloor \rightarrow \lfloor \frac{n}{2} \rfloor - 1$;
C	Each agent request a resource that is locked by another agent, and each agent can be waited by one another agent.

We just statistic the number of DA moves and total number of transmitted CA information in detection phase.

Figure 2 and Figure 3 give the simulation results of performance comparisons. Figure 2 and Figure 3 illustrate the results of the comparisons in aspect of DA moves and transmitted CA information, respectively. Label on horizontal axis and ordinate axis represent the number of consume agents and results, respectively.

From the simulation results, we can find that the number of detection agent movements increasing linearly with the growth of the number of consume agent. The other two algorithms have an exponential growth. The total number of transmitted CA information is the same. The main reason is that our algorithm reuses the collected wait-for information of consume agents with lower id. This dramatically reducing the number of detection movements and total number of transmitted CA information during the detection procedure. In addition, another benefit of our algorithm is that only one agent will detect and resolve the deadlock. This can avoid all associated problems that is caused by multiple resolution of the same deadlock.

Note that, results of Hosseini has better performance than Elkady in Figure 2 and Figure 3. Actually, they

have the same theoretically performance complexity (see Table 4). However, the worst case of Hosseini's approach need extreme conditions of algorithm initiation time that is difficult to implement in this simulation. However, it may happen in real system.

6 Conclusions and Future Works

In this paper, we propose a priority-based deadlock detection and resolution algorithm in mobile agent systems. Our algorithm adopts priority-based approach to coordinate the concurrent execution of algorithm instances. Our algorithm guarantees that a deadlock can only be detected and resolved by one agent. This simplifies the resolution of deadlock, and no phantom deadlock will be detected. In addition, our algorithm provides better performance when concurrent executing of algorithm instances. We prove two important properties (liveness and safety) of our algorithm. Performance analysis and simulation results demonstrate that our algorithm has better performance in aspect of agent movements and information volume.

In the near future, we will pay more attention on the other complexity resource request models in deadlock detection and resolution of mobile agent systems. Besides that, fault tolerance of agent and host environment crashing will be another topic of research in our future works.

ACKNOWLEDGMENT

This work was supported in part by National Natural Science Foundation of China (No.61100143, No.61272353, No.61370128, No.61428201, No.61502028), Program for New Century Excellent Talents in University (NCET-13-0659), Beijing Higher Education Young Elite Teacher Project (YETP0583).

References

[1] B. Ashfield. *Distributed Deadlock Detection in Mobile Agent Systems*. PhD thesis, Carleton University Ottawa, 2000.

[2] B. Ashfield, D. Deugo, F. Oppacher, and T. White. *Distributed deadlock detection in mobile agent systems*. Springer, 2002.

[3] P. G. P. Carzaniga, A. and G. Vigna. Is code still moving around? looking back at a decade of code mobility. In *Companion to the proceedings of the 29th International Conference on Software Engineering*, pages 9–20. IEEE, 2007.

[4] M. M. K. G. Challenger, M. and T. Kosar. Declarative specifications for the development of multi-agent systems. *Computer Standards and Interfaces*, 43:91–115, 2016.

[5] V. Della Mea. Agents acting and moving in healthcare scenario - a paradigm for telemedical collaboration. *IEEE Transactions on Information Technology in Biomedicine*, 5(1):10–13, 2001.

[6] A. Elkady. *Mobile Agents Deadlock Detection in Absence of Priorities*. PhD thesis, Carleton University Ottawa, 2006.

[7] D. R. G. Fredriksson, Olle and B. Wheen. Towards native higher-order remote procedure calls. In *Proceedings of the 26nd 2014 International Symposium on Implementation and Application of Functional Languages*. ACM, 2014.

[8] R. Hosseini and A. T. Haghighat. An improved algorithm for deadlock detection and resolution in mobile agent systems. In *International Conference on Computational Intelligence for Modelling, Control and Automation, and International Conference on Intelligent Agents, Web Technologies and Internet Commerce*, volume 2, pages 1037–1042. IEEE, 2005.

[9] H. Huang, L. Wang, B. C. Tak, L. Wang, and C. Tang. Cap3: A cloud auto-provisioning framework for parallel processing using on-demand and spot instances. In *Proceedings of the 2013 IEEE Sixth International Conference on Cloud Computing*, CLOUD '13, pages 228–235. IEEE, 2013.

[10] D. Isern and A. Moreno. A systematic literature review of agents applied in healthcare. *Journal of medical systems*, 40(2):1–14, 2016.

[11] R. Jain, F. Anjum, and A. Umar. A comparison of mobile agent and client-server paradigms for information retrieval tasks in virtual enterprises. In *Research Challenges, 2000. Proceedings. Academia/Industry Working Conference on*, pages 209–213. IEEE, 2000.

[12] E. Knapp. Deadlock detection in distributed databases. *ACM Computing Surveys (CSUR)*, 19(4):303–328, 1987.

[13] D. Kotz and R. S. Gray. Mobile agents and the future of the internet. *Operating systems review*, 33(3):7–13, 1999.

[14] H. Ma, S. R. Diersen, L. Wang, C. Liao, D. Quinlan, and Z. Yang. Symbolic analysis of concurrency errors in openmp programs. In *the 42th International Conference on Parallel Processing*, pages 510–516. IEEE, 2013.

[15] H. Ma, L. Wang, and K. Krishnamoorthy. Detecting thread-safety violations in hybrid openmp/mpi programs. In *Proceedings of the 2015 IEEE International Conference on Cluster Computing*, pages 460–463. IEEE, 2015.

[16] N. Mani, V. Garousi, and B. H. Far. Search-based testing of multi-agent manufacturing systems for deadlocks based on models. *International Journal on Artificial Intelligence Tools*, 19(04):417–437, 2010.

[17] L. Moreau, M. Luck, S. Miles, J. Papay, K. Decker, and T. Payne. Methodologies and software engineering for agent systems. 2004.

[18] M. Moshirpour, N. Mani, A. Eberlein, and B. Far. Model based approach to detect emergent behavior in multi-agent systems. In *Proceedings of international conference on Autonomous agents and multi-agent systems*, pages 1285–1286, 2013.

[19] N. Sofy and D. Sarne. Effective deadlock resolution with self-interested partially-rational agents. *Annals of Mathematics and Artificial Intelligence*, 72(3–4):225–266, 2014.

[20] J. Waldo. Mobile code, distributed computing, and agents. *IEEE Intelligent Systems*, (2):10–12, 2001.

[21] J. Yang. *Design of fault tolerant mobile agent systems*. PhD thesis, Hong Kong Polytechnic University, 2006.