Journal of Visual Languages and Computing 42 (2017) 23–30

Contents lists available at ScienceDirect

Journal of Visual Languages and Computing

journal homepage: www.elsevier.com/locate/jvlc

Detecting and resolving deadlocks in mobile agent systems

Yong Yang[a], Wei Lu[a,*], Weiwei Xing[a], Liqiang Wang[b], Xiaoping Che[a], Lei Chen[a]

[a] School of Software Engineering, Beijing Jiaotong University, Beijing 100044, China
[b] Department of Computer Science, University of Central Florida, Orlando, FL 32816, USA

ARTICLE INFO

Article history:
Received 15 January 2017
Revised 10 August 2017
Accepted 13 August 2017
Available online 18 August 2017

Keywords:
Deadlock detection
Deadlock resolution
Mobile agent system
Concurrent execution
Priority-based
Lazy reaction

ABSTRACT

Mobile agents environment is a new application paradigm with unique features such as mobility and autonomy. Traditional deadlock detection algorithms in distributed computing systems do not work well in mobile agent systems due to the unique feature property of the mobile agent. Existing deadlock detection and resolution algorithms in mobile agent systems have limitations such as performance inefficiency and duplicate detection/resolution when multiple mobile agents simultaneously detect/resolve the same deadlock. To address these problems, we propose an improved deadlock detection and resolution algorithm that adopts priority-based technique and lazy reaction strategy. The priority-based technique aims to ensure that there is only one instance of deadlock detection and resolution, and it also helps reduce mobile agent movement and data traffic together with the lazy reaction strategy. The liveness and safety properties of the proposed algorithm are proved in this paper. Theoretical analysis and experimental results show that the proposed algorithm provides better performance in terms of agent movement, data traffic, and execution time.

1. Introduction

A mobile agent represents a software object that migrates through multiple host machines in a heterogeneous network. It controls its own movement in order to perform tasks by consuming resources on different host machines [17]. Advantages of mobile agent include mobility, autonomy, sociality, reactivity, proactivity, data acquisition, and route determination [3,4,8,9,11,19]. However, mobile agent technology also introduces some new challenges, such as deadlock, rendezvous, leader election, itinerary formal description, trustworthy, and scheduling, while bringing benefits [1,5,12,22].

In many traditional distributed applications, there exists a strong relationship between the code and data, which leads to various assumptions that may no longer be valid when mobile agents are involved [1,2,5]. Take deadlock detection for instance, traditional deadlock detection algorithms rely on the hypothesis that a process and its locked resources are in the same host machine, which makes it difficult to apply traditional deadlock detection algorithms to mobile agent systems. In addition, existing algorithms of deadlock detection in mobile agent systems mainly focus on performance improvement under single execution (*i.e.*, only one mobile agent detects or resolves a deadlock at any given time). In practice, multiple mobile agents may simultaneously detect the same deadlock (*i.e.*, concurrent execution), which makes the problem more complex [6,13,14,20]. For example, algorithm performance may be degraded significantly under concurrent execution due to more agent movements and data transmissions.

In this paper, we propose an improved priority-based deadlock detection algorithm with a lazy reaction strategy. The proposed algorithm supports both single execution and concurrent execution. In the proposed algorithm, a mobile agent with higher priority suspends the movement of another mobile agent with a lower priority. Only the mobile agent with the highest priority can collect all necessary wait-for relationships among all deadlocked mobile agents. Then, this mobile agent constructs a Wait-For Graph (*abbr.*, WFG, which is a directed graph where a vertex represents a mobile agent and an edge indicates the wait-for relationship between two mobile agents [10]). Deadlocks can be detected through checking a cycle in the constructed WFG. To reduce mobile agent movements, we adopt a lazy reaction strategy when a mobile agent is visiting another mobile agent. This improves the algorithm performance by reducing both mobile agent movements and data transmissions.

The proposed algorithm can be roughly divided into three phases: algorithm initiation, deadlock detection, and deadlock resolution. In the first phase, a consumer agent launches the deadlock detection algorithm by creating a detection agent. In deadlock detection phase, each detection agent creates necessary probe agents. All detection agents and probe agents cooperate to collect wait-

* Corresponding author.
E-mail address: luwei@bjtu.edu.cn (W. Lu).

http://dx.doi.org/10.1016/j.jvlc.2017.08.002
1045-926X/

for relationships, construct WFG, and detect deadlocks. In the last phase, a resolution agent will be created and moves to victim consumer agents for conveying deadlock resolution.

The rest of this paper is organized as follows. Section 2 reviews the related work. Prerequisites and assumptions are given in Section 3. The proposed algorithm is introduced in Section 4. Section 5 gives proofs of the liveness and safety properties. Section 6 illustrates theoretical analysis and experimental results. Section 7 gives conclusions and future work.

2. Related work

In this section, we review the related work [1,2,5,7,15,16,18,21] on deadlock detection and resolution in mobile agent systems.

Ashfield *et al.* propose an edge chasing based deadlock detection algorithm with dedicated shadow agents [1,2]. A consumer agent creates a shadow agent to monitor its activities when it requests an exclusive lock of a resource. The shadow agent creates a detection agent to perform deadlock detection if the consumer agent has been blocked for a predefined time interval. The detection agent constructs a WFG, then detects and resolves a deadlock by visiting related shadow agents and collecting wait-for relationship of the consumer agent. This algorithm has some limitations such as undetected deadlocks, false deadlock detection, and too many detection agent movements [5,7].

Based on [1,2], Hosseini et al. present an improved algorithm [7] that assigns priorities to both resources and consumer agents. The detection agent with the lowest priority created by a consumer agent suspends other detection agents with higher priorities. Therefore, only the detection agent with the lowest priority can pass through the whole cycle and return under concurrent execution. However, this algorithm is limited by the order in which a detection agent visits other consumer agents. Performance is degraded significantly if a consumer agent is visited by detection agents when priorities of these detection agents are in descending order. In this case, wait-for relationships collected by a detection agent is duplicate and useless. In other words, there exists redundant collection of the same wait-for relationships.

Elkady proposes a path pushing based algorithm to detect deadlock [5]. This algorithm asserts a deadlock when a detection agent visits a consumer agent twice. In addition, it supports deadlock avoidance and concurrent execution. However, it does not consider the problem of performance inefficiency and duplicate detection/resolution under concurrent execution.

Yang proposes two deadlock detection algorithms named MA-WFG and Host-WFS [21]. In the MA-WFG algorithm, a mobile agent tries to construct a WFG to detect a loop topology. Locally constructed WFG will be passed to the mobile agent that locks the required resource to construct its local WFG. A deadlock is detected if there is a loop topology in the constructed WFG. To reduce agent movements and the network load, Host-WFS algorithm encodes the WFG in the form of "wait-for set" and distributes them to different host machines. Thus, the host machines control path pushing, and mobile agents do not need to participate in path pushing. However, propagation of WFG leads to large amount of message transmissions that cause heavy data traffic.

Mani et al. [15,16] design diagrams and models resembling the standard Unified Modeling Language (UML) to describe the architecture-independent structure of agents and their interactions. They propose an algorithm to extract scenarios that express the overall functionality and behaviors from Multi-agent Software Engineering Models. Scenarios are used to examine the system for possible design faults. Due to the problem of state explosion, this approach can not detect all run-time errors.

Sofy and Sarne propose a game-theoretic based approach to handle distributed deadlock resolution of autonomous self-interested partially rational agents [18]. They do not consider the performance degradation under concurrent execution, and focus on the deadlock resolution rather than deadlock detection.

There are some limitations in existing deadlock detection algorithms.

1) Performance is degraded under concurrent execution due to duplicate transmission of the same wait-for relationships during mobile agent movement.
2) There may be non-optimal or wrong deadlock resolution under concurrent execution.

3. Prerequisites

3.1. Abbreviations

1) *CA* (Consumer Agent): It performs common tasks and communicates with host environment to request and lock a resource. It is inactive in the deadlock detection procedure but can spawn a detection agent.
2) *DA* (Detection Agent): It is spawned by CA and manages deadlock detection and resolution procedures.
3) *PA* (Probe Agent): It is spawned by DA and responsible for collecting wait-for relationships between consumer agents.
4) *RA* (Resolution Agent): It is spawned by DA and responsible for conveying deadlock resolutions.

3.2. Assumptions

We assume that mobile agent systems satisfy the following conditions.

1) Network organization independence: Neither the host machines nor the mobile agents maintain the state about the size or topology of the network. The deadlock detection and resolution algorithm should not depend on a particular topology of underlying network.
2) Agent movement: Mobile agents are allowed to move around in the system and lock granted resources.
3) Fault tolerance: A failed host machine or a mobile agent can be recovered.

Based on the above premises, we give some assumptions as follows.

1) Host Environment, which is an execution environment for mobile agents, provides APIs to mobile agents. A mobile agent communicates with the host environment to obtain needed information through these APIs. Host environment is the ultimate authority that allows or denies a resource locking request.
2) Each mobile agent can lock multiple resources; however, it can request only one resource at any moment and does not request resource anymore when it is blocked. A mobile agent can choose to be blocked when the resource locking request is denied, or neglect the rejection and perform other tasks. A mobile agent can move through the network without being lost or tampered. A mobile agent makes the decision of moving by itself within an uncertain but finite time. A mobile agent can be located through its itinerary (which is recorded in the visited host environments) by the techniques in [12,22].
3) Both mobile agents and host environments have global unique identities in the system through techniques such as static path proxy or naming service.[1]

[1] We assume that there exists a method to obtain a globally unique identity in a mobile agent system. It is another research topic beyond the scope of this paper.