A Novel Concurrent Generalized Deadlock Detection Algorithm in Distributed Systems

Wei Lu[1], Yong Yang[1(✉)], Liqiang Wang[2], Weiwei Xing[1], and Xiaoping Che[1]

[1] School of Software Engineering, Beijing Jiaotong University, Beijing 100044, China
{wlu,12112088,wwxing,xpche}@bjtu.edu.cn
[2] Department of Electrical Engineering and Computer Science,
University of Central Florida, Orlando, FL 32816, USA
lwang@cs.ucf.edu

Abstract. Detecting deadlocks has been considered an important problem in distributed systems. Many approaches are proposed to handle this issue; however, little attention has been paid on coordinating concurrent execution of distributed deadlock detection algorithms. Previous approaches may report incorrect results (false negatives), and they are inefficient due to lack of proper coordination of concurrent execution. In this paper, we present a novel concurrent coordination algorithm for distributed generalized deadlock detection. The proposed algorithm aims to avoid false negatives and improve the performance when concurrently executing deadlock detection in a distributed system. Our algorithm adopts diffusion computation to distribute probe messages and employs priority-based method to coordinate concurrent algorithm instances. Priority carried in the received probe messages will be locally recorded by each initiator. Instead of being suspended by higher priority algorithm instances, lower priority algorithm instances can accomplish deadlock detection locally. The initiator with the highest priority will receive and collect all related resource requests information from lower priority instances in a hierarchical manner and perform global deadlock detection at last. We evaluate our algorithm on a bunch of event-driven simulations. The experimental results show that our approach can achieve better accuracy and efficiency compared to previous approaches.

Keywords: Distributed system · Deadlock detection · Concurrent coordination · False negative

1 Introduction

Problems of detecting deadlocks have been long considered important problems in distributed systems due to the vulnerability to deadlocks. A deadlock occurs when processes wait for resources held by other processes, such as data object in database systems, buffers in store-and-forward communication networks, or messages in message passing systems. The wait-for relationships are usually represented by WFG (Wait-for graph), a directed graph in which vertices represent

G. Wang et al. (Eds.): ICA3PP 2015, Part II, LNCS 9529, pp. 479–493, 2015.
DOI: 10.1007/978-3-319-27122-4_33

processes and edges indicate not granted resource requests between processes, in a distributed system [1–3].

Some researches have classified existing deadlock detection algorithms in terms of underlying resource request models [3–5]. For example, in the *p-out-of-q* (also called *generalized*) model, a process makes requests for q resources and remains idle until p out of the q resources are granted [6–8].

Deadlocks in the generalized model correspond to generalized deadlocks. Detection of generalized deadlock is rather difficult, because the resource dependent topology is more complex than those in other models. A *cycle* in the WFG is a necessary but not sufficient condition in the AND model, whereas a *knot* (which is a strongly connected sub-graph with no edge directed away from the sub-graph in a directed graph [9–11]) is a sufficient but not necessary condition for a generalized deadlock [12]. It become more complicated when several processes initiating the deadlock detection algorithm concurrently, e.g., a process might be involved in more than one instance and will be declared deadlocked and resolved by more than one algorithm instance. This problem can result in useless abortion, i.e., false deadlock resolution. Priority-based approach is often used to address the aforementioned problem; however, improper coordination of algorithm instances may cause false negative and poor performance.

To avoid false negatives and improve performance in concurrent execution of algorithm, we propose a novel concurrent coordination algorithm for distributed generalized deadlock detection. During the probe phase, each initiator will record the priority locally when receiving a probe message. After the probe phase, a hierarchical resource request report mechanism is exploited to construct a global WFG in the initiator with the highest priority.

Contributions of proposed algorithm are summarized as follows:

(1). The proposed algorithm can handle all kinds of aforementioned resource request models;
(2). Our algorithm can avoid false negatives of deadlock detection and provide better performance.

The rest of this paper is organized as follows. Related work is presented in Sect. 2. System models and definitions are described in Sect. 3. Section 4 details the proposed algorithm. Experiment results are shown in Sect. 5. At last, Sect. 6 gives conclusions and future works.

2 Related Work

2.1 Categories of Deadlock Detection Algorithms

Deadlock detection approaches can be classified in different ways according to classification criteria. *Singhal* [3] classified the deadlock detection in distributed systems into three types: centralized, decentralized, and hierarchical, according to the way in which WFG is maintained and how to detect cycles and knots. *Knapp* described three taxonomies based on theoretical principle: path pushing [14] (which is later disproved due to asynchronous snapshots at different

sites [15]), probe-based (includes edge chasing and diffusing computing) [16], and global state detection, to detect deadlocks in distributed systems [2]. *Brzezinski* defined five types of resource request models: Single-Request, AND, OR [12,17], AND-OR, and generalized model (p-out-of-q) [18–22], based on the complexity of resource requests [3–5].

2.2 Review of Algorithms of Generalized Deadlock Detection

Centralized Deadlock Detection Algorithms. A process, acting as initiator, sends probe messages to its direct and indirect successor processes when it suspecting itself blocked in a defined time interval. All reply messages will be sent back directly to the initiator where a global WFG and deadlock detection will be constructed and performed.

The authors of [13] proposed a centralized algorithm in which an initiator send probe messages directly to all the processes reachable from itself, and replies from successor processes are sent back to the *initiator* process directly. The Initiator is in charge of constructing global WFG and detecting deadlocks.

The authors of [21] proposed a two-phases centralized algorithm. Unlike classical diffusion computation, the resource request is carried in the probe messages, and report messages are send to initiator directly rather than backward along the opposite direction of edges. The resource requests are equal distributed in each probe message, and the performance is improved in this manner.

The authors of [19,20] proposed a centralized algorithm to detect and resolve deadlocks in which the termination of the algorithm depended on either technique of weight distribution liked [1] or whenever a deadlock was detected. Reply messages carrying weight information will be delayed until all the probe messages have been received from all its predecessor processes. Message overhead will be minimized through merging weighted messages into one message. To decrease time complexity, the proposed algorithm performs reduction as soon as a reply message from an active process is received.

Decentralized Deadlock Detection Algorithms. Different with centralized approaches, no complete global WFG is constructed by the initiator node (site, process, and node will be used interchangeably throughout this paper) in decentralized approaches. Nevertheless, the WFG constituted in multiple sites in decentralized deadlock detection algorithms.

The paper [12,23] presented a one-phase decentralized algorithm for detecting generalized distributed deadlocks. The algorithm initiated by an initiator consists of outwards and inwards sweep. The outwards sweep induces a spanning tree and records WFG by propagating the probe messages. The inwards sweep performs a reduction in an up-tree direction started at an active node. An ECHO message will be replied by an active node when receiving probe messages. A PIP message, contains ids of nodes that sent PIP messages but reduces later and residual requirement conditions, will be replied when a node receiving the second and subsequent probe message if the state of this node is indeterminate at this instant. When a node that sent PIP message receiving ECHO

or PIP message that makes it reduced would send ECHO message to its parent node in directed spanning tree. This algorithm performed reduction in a "lazy evaluation" manner.

The paper [22] proposed a semi-centralized algorithm for distributed generalized deadlock detection and resolution. They adopted hash table to save global ids, initiation time-stamp (hash key) of initiator, resource requests and phantom edges (hash value). The approach delays the phantom edge reply message reporting operating and combines it with common reply messages, and reply messages are in a reduction manner by reduction of request conditions of involved nodes.

Hierarchical Deadlock Detection Algorithm. Sites are arranged in a hierarchical way, and each site only responses for detecting deadlock that involves its child processes.

A hierarchical algorithm was proposed to detect deadlocks in distributed database system in [24]. They arranged all the resource controllers in a tree structure. Controllers were classified into two types: leaf-controllers and nonleaf-controllers. A leaf-controller responds to manage resources and collect part of the global TWF (Transaction Wait-For) that represent the wait-for relations of corresponding resources. Whenever there is a change of TWF occurs in a leaf-controller, the change will be propagated to its parent non-leaf controller to do deadlock detection locally.

The authors of [25] proposed a hierarchical deadlock detection algorithm different from [24]. A central control site is chosen among all sites, and the rest of sites will be divided into several clusters in which an inter-cluster control site will be selected by the central control site. The inter-cluster control site responds to detecting inner cluster deadlock and reports its detection results to the central control site. The global deadlocks will be detected by the central control site finally.

2.3 Previous Concurrent Coordination Strategies

Most of the previous works focused on performance improvement of single execution of the deadlock detection algorithm. Very few effort was paid on handling the practical and important problem: concurrent execution in which more than one node conduct concurrent deadlock detection in a distributed system. Priority-based approach was commonly used by most representative algorithm that address problem of concurrent execution [12,20–22]. They can be classified into mainly two categories: [12]-like and [21]-like.

The authors of [12] proposed a concurrent execution strategy in which high priority algorithm instance will suspend low-priority one. So, algorithm instance with lower priority cannot complete in this approach, and message overload (number and size) is high due to repeatedly sending probe messages by processes with lower priority. That is, a process will terminate the current algorithm instance to join a higher priority instance when it receives a probe message with a higher priority.

The authors of [21] proposed an improved approach to improve performance by equal distributing resource requests in probe messages to reduce the message size in the probe phase. In addition, a higher priority process will report an active state to a lower priority process when the higher priority process is receiving a probe message from the lower one. It is based on the assumption that processes with higher priority will always resolve the detected deadlocks. The lower priority algorithm can accomplish and do local deadlock detection and resolution after receiving enough resource request information [21].

The shortage of [12] is that repeatedly work of low-priority instance will be done by high priority instance again. And, false negatives might happen if a probe message is sent to a process with a higher priority by a process with a lower priority in an OR model [21].

To address the aforementioned shortages in previous approaches, we propose a novel concurrent coordination algorithm for generalized distributed deadlock detection by:

(1). Utilizing equal distribution of resource requests in probe messages to reduce the message size like [21];
(2). Reusing the information discarded by previous approaches to reduce message overhead, that is, lower priority initiator process's execution will not be suspended by higher ones any more;
(3). Recording priority of probe messages locally and adopting a hierarchical resource request report mechanism to collect global WFG to avoid false negatives.

3 Preliminaries

3.1 System Model

This paper follows the system model described in [12,20–22]. A distributed system consists of various site communicating with each other by message passing to exchange messages or access resource. Messages are sent and received in a reliable way but their delay time is uncertain. A node has one of the following two states: ***active*** and ***blocked***. In a generalized model a node is on ***active*** state when sufficient number of its resource requests are granted, otherwise it is on ***blocked*** state. Two kinds of messages, computation message and control message, are transmitted in the system. Computation messages are triggered by the execution of applications, and control messages are issued when the execution of deadlock detection algorithm. Both computation message and control message could be sent by an ***active*** process, however, a ***blocked*** process can only send control message.

3.2 Definitions of Deadlocks

Definition 1. *WFG is a directed graph $G(V, E)$, V denotes the set of all vertices and E is the set of directed edges.*

A generalized deadlock exists in a distributed system *iff* the resource request of processes can never be granted. Actually, the resource request of deadlocked processes consists of a sub-graph $D(N, K)$ in $G(V, E)$, and all resource requests belonging to each node in D will never be granted while the resource requests belonging to the node out of D is always satisfied.

Definition 2. *Let rc_i denote the resource conditions or requests of node i.*

For example, $rc_i = (j\&k)|l$ means node i becomes **active** if both node j and node k grant the request resources to i, or node l grants the request resource to node i.

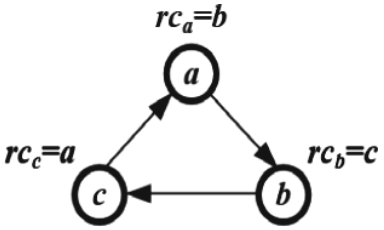

Fig. 1. Example of deadlocked WFG

Definition 3. *Let evaluate(rc_i) be a recursive operation based on the following:*

(1). $evaluate(rc_i) = true$ for an active node i,
(2). $evaluate(i) = evaluate(rc_i)$,
(3). $evaluate(P \vee Q) = evaluate(P) \vee evaluate(Q)$,
(4). $evaluate(P \wedge Q) = evaluate(P) \wedge evaluate(Q)$.

where P and Q are non-empty AND/OR expression of node identifiers.

Definition 4. *A sub-graph $D(N, K)$ involves a deadlock if the following conditions are satisfied:*

(1). $evaluate(rc_i) = false, \forall\ i \in D$.
(2). No computation message is under transmission between any nodes in D.

For example, $D(N, K)$ presents a WFG in which $K = \{\langle a, b\rangle, \langle b, c\rangle, \langle c, d\rangle\}$ and $N = \{a, b, c\}$ in Fig. 1. $evaluate(rc_a) = evaluate(a) = evaluate(b)$. State of node a is decided by node b, and other nodes are evaluated in a similar way.

4 Proposed Algorithm

In this section, we will omit describing the single execution of the proposed generalized deadlock detection algorithm, but detail the proposed algorithm used to coordinate the concurrent execution of deadlock detection algorithm instances.

In a distributed system, processes can be blocked concurrently, and all the blocked processes may initiate the deadlock detection algorithm after being blocked for a certain time interval. It will become more complex when more than one algorithm instance executing concurrently in a distributed system. Especially, a process is participating more than one algorithm instance. The proposed algorithm aims to handle the concurrent execution of deadlock detection algorithm to avoid false negatives and improve performance.

4.1 Informal Description of Concurrency Execution

The proposed algorithm adopts a priority-based approach like previous works to coordinate the concurrency execution. Instead of being suspended by higher priority instances, lower priority instances can continue executing even if encountering a higher priority instance in the proposed algorithm. The initiator will record both lower and higher priority in the probe messages when receiving a probe message with a different priority. The probe phase will complete when all initiator nodes finish local procedure. At last, the initiator with the highest priority will collect resource requests by receiving report messages from lower priority initiator nodes in a hierarchical manner.

A node initiates a deadlock detection algorithm instance by sending probe messages with a priority (denoted as its *id* for simple in this paper) and a weight value to its direct successor nodes when it suspects itself being blocked as **Algorithm** 1. The probe message will be propagated by nodes that receive the probe message and have not participated any other algorithm instance yet. Resource requests are distributed among all probe messages sent to successor processes. Probe message will be not propagated by the node that has participated an algorithm instance. There are three cases when a node receiving a probe message:

Case 1, the node has participated an algorithm instance and has a lower priority than received probe message as **Algorithm** 2;

Case 2, the node has participated an algorithm instance and has a higher priority than received probe message as **Algorithm** 2;

Case 3, it is a leaf node as **Algorithm** 3 **line 4**;

The node will record the priority in the probe message and send a REPORT message with its local current priority to the initiator process that propagates the probe message in both case 1 and case 2. The priority in the REPORT message will be also recorded locally or REPORT to the initiator of the instance that this node is participating now as **Algorithm** 5. In case 3, the node will send REPORT message to the initiator node directly.

A weight value will also be carried by the REPORT message and sent back to reply PROBE message. An algorithm terminates when the initiator receives all distributed weights and the sum of all the weight is 1. An initiator will complete its local deadlock detection algorithm if the sum of weight value is 1 as **Algorithm** 2 **line 14 - line 23** and **Algorithm** 4. If a node is a leaf node (i.e., has no resource request from any other processes) or has the least priority, detection results (deadlocked and active) will be reported to the initiator that has the minimal priority in the set of initiators with priority higher than local

priority as **Algorithm 7 line 10 - line 13**. Generally, a node will report its local detection results to the higher one after receiving REPORT messages from all processes have lower priority recorded locally. The process within the highest priority will receive all local detection results and resource requests, construct a global WFG, and perform global deadlock detection at last as **Algorithm 6**.

We also detect phantom edge, node i send a probe message to node j while a reply message is being transmitted to i by j in edge $i \rightarrow j$, during the deadlock detection procedure to avoid false results as **Algorithm 3**. Proposed algorithm will determine an edge is a phantom edge when the probe message is replied.

4.2 Formal Description of Concurrency Execution

Data types, message types and operations are formally defined as follows (Table 1).

Table 1. Date types, additional data types, message types, and operations at node i

Data type	Description
W	Weight value that is carried in PROBE and REPORT messages
rc_i	Resource request of node i
RC_i	Resource request that is collected by node i
IN_i	Ids of node that has requested resource to node i and has been not granted by node i
OUT_i	Ids of node that has been requested resource by node i and has been not granted resource to node i
$PROBE$	$PROBE(pri, i, RC_i, W)$ is sent by node i with priority (pri), resource request (RC_i), and WEIGHT (W)
$REPORT$	$REPORT(pri, i, RC(j), W, Pri_{cur})$ is sent by node j to initiator node that has a priority (pri) with local priority (Pri_{cur}), resource request (RC_j), weight (W)
$NOTIFY$	$NOTIFY(pri, Pri_{cur})$ is sent by node that has Pri_{cur} to initiator node that has pri
$SUBMIT$	$SUBMIT(pri, i, RC_i, Pri_{high}, Pri_{low})$ is sent by node i to initiator node has pri with resource request (RC_i), priority (Pri_{high} and Pri_{low}) collected locally
$WEIGHT$	Sum of weight values in $REPORT$ messages (additional data type)
Pri_{high}	Set of priority that is higher that Pri_{cur} (additional data type)
Pri_{low}	Set of priority that is lower that Pri_{cur} (additional data type)
$max\{Pri_{high}\}$	The maximum value in Pri_{high}
$min\{Pri_{low}\}$	The minimum value in Pri_{low}
$\lvert\{1, 2, ..., n\}\rvert$	The size of a set

Algorithm 1. When node i initiating an deadlock detection algorithm instance

for $j \in OUT_i$ **do**
    Sending **PROBE**$(i, i, rc, \frac{1}{|OUT_i|})$ to node j;
end for

Algorithm 2. When a **non-leaf** node j receiving a $PROBE(pri, i, rc, w)$ message

if $i \rightarrow j$ is not a phantom edge **then**
    if $Pri_{cur}! = NULL$ and $Pri_{cur} > pri$ **then**
        $Pri_{low} := Pri_{low} \cup \{pri\}$;
        Sending **NOTIFY**(pri, Pri_{cur});
        if node j is not an initiator **then**
            Sending **NOTIFY**(Pri_{cur}, Pri_{cur});
        end if
    else if $Pri_{cur}! = NULL$ and $Pri_{cur} < pri$ **then**
        $Pri_{high} := Pri_{high} \cup \{pri\}$;
        Sending **NOTIFY**(pri, Pri_{cur});
        if node j is not an initiator **then**
            Sending **NOTIFY**(Pri_{cur}, Pri_{cur});
        end if
    else if $Pri_{cur}! = NULL$ and $Pri_{cur} == pri$ **then**
        if node i is an initiator **then**
            $WEIGHT := WEIGHT + w$;
            if $WEIGHT == 1$ and $Pri_low == \emptyset$ **then**
                Algorithm 7;
            end if
        else
            Sending **REPORT**$(Pri_{cur}, Pri_{cur}, rc, w, Pri_{cur})$;
        end if
    end if
    if $Pri_{cur} == NULL$ **then**
        $Pri_{cur} := pri$;
        $RC_j := rc \cup rc_j$;
        $w^{'} := \frac{w}{|OUT_j|}$
        for $k \in OUT_k$ **do**
            Sending **PROBE**$(Pri_{cur}, j, \frac{RC_j}{|OUT_j|}, w^{'})$ to node k;
        end for
    end if
else
    Discarding the PROBE message (a phantom edge: $i \rightarrow j$);
    Sending **REPORT**$(pri, i, rc, w, \emptyset)$;
end if

5 Experiment

In this section, we evaluate the correctness and performance of the proposed algorithm and present the simulation results. False negative is mainly

Algorithm 3. When a **leaf** node j receiving a $PROBE(pri, i, rc, w)$ message

if $i \in IN_j$ **then**
  $Pri_{cur} := pri$;
  $RC_j := rc \cup rc_j$;
  Sending **REPORT**$(pri, i, RC_j, w, Pri_{cur})$;
else
  Discarding the PROBE message (a phantom edge: $i \rightarrow j$*)*;
  Sending **REPORT**$(pri, i, rc, w, \emptyset)$;
end if

Algorithm 4. When a node i receiving a $REPORT(pri^{'}, j, rc, w, pri^{''})$ message

$WEIGHT := WEIGHT + w$;
$RC_i := RC_i \cup rc$
if $pri^{''} ! = \emptyset$ **then**
  if $WEIGHT == 1$ and $Pri_{low} == \emptyset$ **then**
    Algorithm 7;
  end if
else
  $RC_i := RC_i / i \rightarrow j$;
end if

Algorithm 5. When a node i receiving a $NOTIFY(pri^{'}, pri^{''})$ message

if $Pri_{cur} > pri^{'}$ **then**
  $Pri_{low} := Pri_{low} \cup \{pri^{'}\}$;
else if $Pri_{cur} < pri^{'}$ **then**
  $Pri_{high} := Pri_{high} \cup \{pri^{'}\}$;
end if

Algorithm 6 . When a node i receiving a $SUBMIT(pri, j, RC_j, Pri^{'}_{high}, Pri^{'}_{low})$ message

$RC_i := RC_i \cup RC_j$;
$Pri_{high} := Pri_{high} \cup Pri^{'}_{high}$;
$Pri_{tmp} := Pri_{tmp} \cup Pri^{'}_{low}$;
if $Pri_{cur} == max(Pri_{cur})$ **then**
  if $Pri_{tmp} \cap Pri_{low} == Pri_{low}$ **then**
    Algorithm 7 (line 1-line 7) and terminating;
  end if
else
  if $Pri_{tmp} \cap Pri_{low} == Pri_{low}$ **then**
    Algorithm 7 (line 11-line 13) and terminating;
  end if
end if

considered as the correctness metric. The performance metrics mainly total message number and total message size of submitted messages in the life-cycle of

Algorithm 7. Locally deadlock detection procedure $LDD(RC_i)$

```
for rc^j ∈ RC_i do
    if evaluate(rc^j) == flase then
        Deadlocked := Deadlocked ∪ {j}
    else
        Active := Active ∪ {j}
    end if
end for
if Pri_cur == max{Pri_high} then
    Algorithm terminating;
else
    pri := min{Pri_high};
    Sending SUBMIT(pri, i, RC_i, Pri_high, Pri_low);
end if
```

algorithm. Especially, total message size is the number of resource request contained in a PROBE message.

5.1 Experiment Setup

Simulation programs are event-driven and written in Python2.7. Processes are used to simulate nodes in a distributed system, and network socket is used to implement message passing and simulate wait-for relationship between processes (nodes). The number of socket calls (e.g., socket.send() and socket.recv()) and the size of data that presents resource request are collected and compared. Each simulation result is the mean value obtained after running the program for 100 times. We choose two extreme resource request models: Single-Request and Generalized model, to perform simulation. Each process waits for another one process that has a lower id value (i.e., priority) in Single-Request model. In the Generalized model, each process waits for all the other processes excepting for itself. Six special cases are constructed manually in Fig. 2 to examine the correctness and efficiency of the distributed deadlock detection algorithms.

5.2 Simulation Results

Table 2 gives the quantitative comparison among [12,21] and the proposed methods where notation "NFN" denotes no false negatives and notation "FN" denotes false negatives. The reason of false negative in [21] comes from the OR request model. Based on the assumption that higher priority algorithm instance will resolve the deadlock, detected deadlocks (e.g., a and c) in Fig. 2(d) can be resolved even if there is a false negative. However, no deadlocks will be detected in Fig. 2(e) and (f), because the edge $c \rightarrow d$ propagates the false negative result to sub-graph (a and c included). The proposed algorithm can detect all deadlocks in Fig. 2 without false negatives.

The value of x-axis represents the number of processes. The value of y-axis of left sub-figure in Figs. 3, 4, 5, and 6 means the total number of messages that

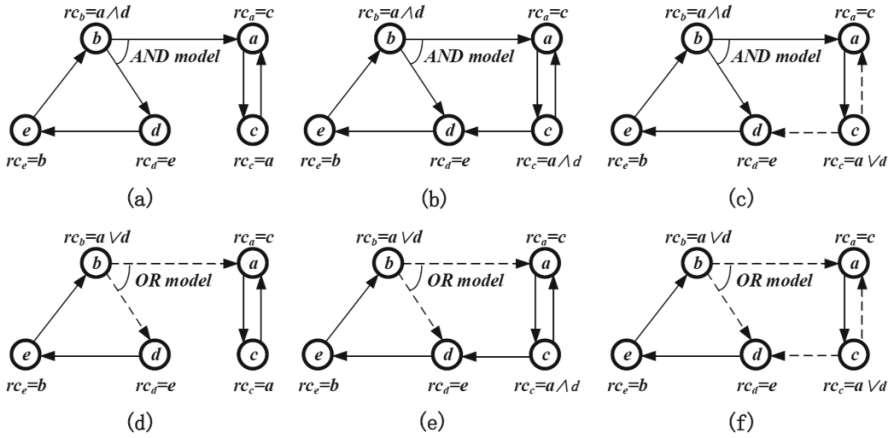

Fig. 2. Six WFGs of different resource request models

Table 2. Results of correctness simulations

	Fig. 2(a)	Fig. 2(b)	Fig. 2(c)	Fig. 2(d)	Fig. 2(e)	Fig. 2(f)
[12]-like	NFN	NFN	NFN	NFN	NFN	NFN
[21]-like	NFN	NFN	NFN	FN	FN	FN
Proposed	NFN	NFN	NFN	NFN	NFN	NFN

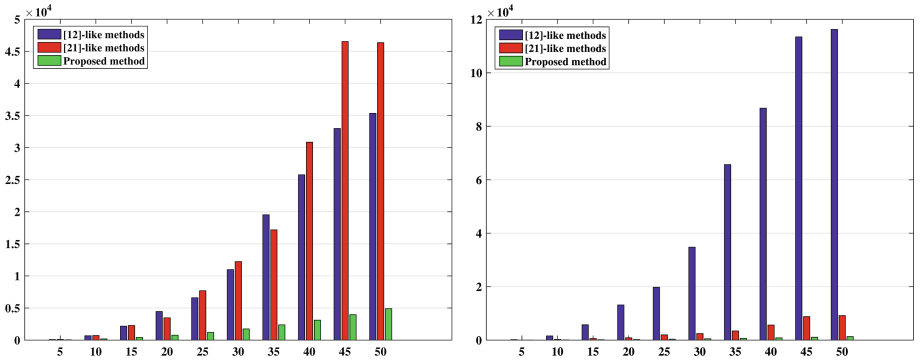

Fig. 3. All processes as initiator in Generalized model

are transmitted, and the value of y-axis of right sub-figure represents the total data size of transmitted messages.

From left sub-figures of Figs. 3, 4, 5, and 6, we can find that the proposed algorithm has better performance than [12,21] in the aspects of total number of message transmitted during the life-cycle of algorithm. [12,21] transmitting almost the same number of messages in the simulations, because they both used the strategy of suspending lower priority instance with higher one. Proposed

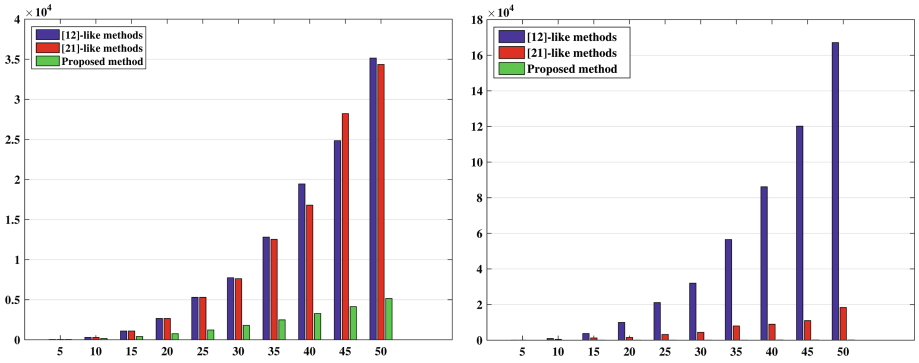

Fig. 4. 1/5 of processes as initiator in Generalized model

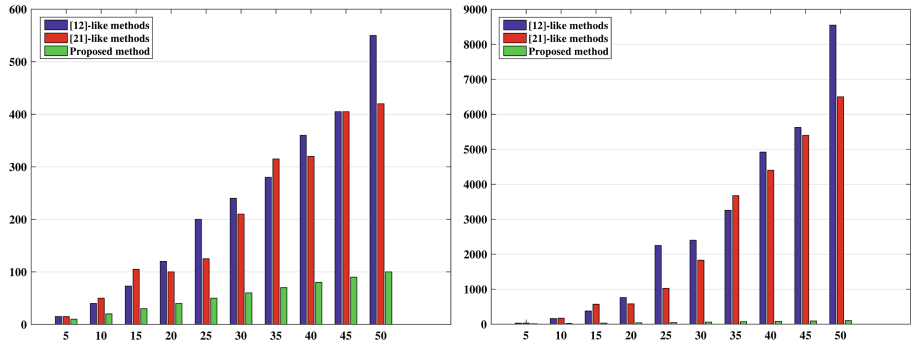

Fig. 5. All processes as initiator in Single-Request model

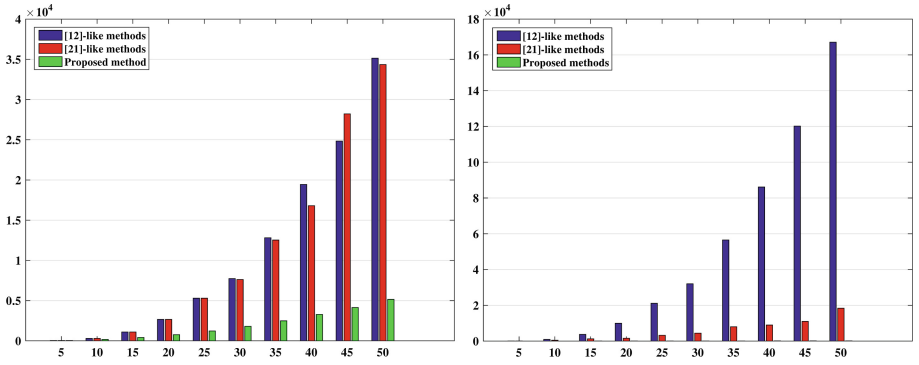

Fig. 6. 1/5 of processes as initiator in Single-Request model

method reduces the number messages by recording priority and report resource requests in the hierarchical report phase.

From right sub-figures of Figs. 3, 4, 5, and 6, we can find that [21]-like methods transmits smaller number of message size than [12]-like methods. The reason is equal distribution of resource requests in PROBE phase. Proposed methods has smaller number of message size than [21]-like methods, because proposed method avoid sending PROBE message by lower priority initiators when receiving PROBE messages from other initiators based on improvement of [21]-like methods.

6 Conclusions and Future Works

This paper focuses on the problem of coordinating the deadlock detection algorithm instances that executing concurrently in a distributed system. The proposed algorithm coordinates the execution of concurrent instances by recording priority of different algorithm instance and reporting resource requests in a hierarchical manner. The proposed algorithm can avoid false detection results (false negatives) and reduce the number and size of message transmitted during the life-cycle of a complete deadlock detection. Simulation results show that the proposed algorithm can report correct detection results and have better performance.

The approach presented in this paper can be seen as a very first step towards a solution for the problems of generalized distributed deadlock detection. A limitation exists in the current proposal with respect to the time efficiency of the algorithm. In the future, we will aim to improve the time complexity and apply the proposed algorithm to detect run-time deadlocks in real scenario, such as, MPI programs, network forwarding or distributed database systems.

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

References

1. Kshemkalyani, A.D., Singhal, M.: Efficient detection and resolution of generalized distributed deadlocks. IEEE Trans. Softw. Eng. **20**(1), 43–54 (1994)
2. Knapp, E.: Deadlock detection in distributed databases. ACM Comput. Surv. (CSUR) **19**(4), 303–328 (1987)
3. Singhal, M.: Deadlock detection in distributed systems. Computer **22**(11), 37–48 (1989)
4. Brzezinski, J., Helary, J.M., Raynal, M., Singhal, M.: Deadlock models and a general algorithm for distributed deadlock detection. J. Parallel Distrib. Comput. **31**(2), 112–125 (1995)
5. Singh, S., Tyagi, S.S.: A review of distributed deadlock detection techniques based on diffusion computation approach. Int. J. Comput. Appl. **48**(9), 28–32 (2012)
6. Chandy, K.M., Misra, J., Haas, L.M.: Distributed deadlock detection. ACM Trans. Comput. Syst. (TOCS) **1**(2), 144–156 (1983)

7. Edgar, K.: Deadlock detection in distributed databases. ACM Comput. Surv. (CSUR) **19**(4), 303–328 (1987)
8. Lee, S.: Efficient generalized deadlock detection and resolution in distributed systems. In: 21st International Conference on Distributed Computing Systems, pp. 47–54 (2001)
9. Gunther, K.: Prevention of deadlocks in packet-switched data transport systems. IEEE Trans. Commun. **29**(4), 512–524 (1981)
10. Gambosi, G., Bovet, D.P., Menascoe, D.A.: A detection and removal of deadlocks in store and forward communication networks. In: Performance of Computer-Communication Systems, pp. 219–229 (1984)
11. Cidon, I.: An efficient distributed knot detection algorithm. IEEE Trans. Softw. Eng. **15**(5), 644–649 (1989)
12. Kshemkalyani, A.D., Singhal, M.: A one-phase algorithm to detect distributed deadlocks in replicated databases. IEEE Trans. Knowl. Data Eng. **11**(6), 880–895 (1999)
13. Chen, S., Deng, Y., Attie, P., Sun, W.: Optimal deadlock detection in distributed systems based on locally constructed wait-for graphs. In: Proceedings of the 16th International Conference on Distributed Computing Systems, pp. 613–619. IEEE (1996)
14. Beeri, C., Obermarck, R.: A resource class independent deadlock detection algorithm. In: Proceedings of the Seventh International Conference on Very Large Data Bases, vol. 7, pp. 166–178 (1981)
15. Elmagarmid, A.K.: A survey of distributed deadlock detection algorithms. ACM SIGMOD Rec. **15**(3), 37–45 (1986)
16. Chandy, K.M., Ramamoorthy, C.V.: Rollback and recovery strategies for computer programs. IEEE Trans. Computers. (TC) **100**(6), 546–556 (1972)
17. Lee, S., Joo, K.H.: Efficient detection and resolution of OR deadlocks in distributed systems. J. Parallel Distrib. Comput. **65**(9), 985–993 (2005)
18. Selvaraj, S., Ramasamy, R.: An efficient detection and resolution of generalized deadlocks in distributed systems. Int. J. Comput. Appl. **1**(19), 1–7 (2010)
19. Srinivasan, S., Rajaram, R.: A decentralized deadlock detection and resolution algorithm for generalized model in distributed systems. Distrib. Parallel Databases **29**(4), 261–276 (2011)
20. Srinivasan, S., Rajaram, R.: An improved, centralised algorithm for detection and resolution of distributed deadlock in the generalised model. Int. J. Parallel Emergent Distrib. Syst. **27**(3), 205–224 (2012)
21. Lee, S.: Fast, centralized detection and resolution of distributed deadlocks in the generalized model. IEEE Trans. Softw. Eng. **30**(9), 561–573 (2004)
22. Tao, Z., Li, H., Zhu, B., Wang, Y.: A semi-centralized algorithm to detect and resolve distributed deadlocks in the generalized model. In: 2014 IEEE 17th International Conference on Computational Science and Engineering (CSE), pp. 735–740 (2014)
23. Kshemkalyani, A.D., Singhal, M.: Distributed detection of generalized deadlocks. In: Proceedings of the 17th International Conference on Distributed Computing Systems, pp. 553–560. IEEE (1997)
24. Menasce, D.A., Muntz, R.R.: Locking and deadlock detection in distributed data bases. IEEE Trans. Softw. Eng. **3**, 195–202 (1979)
25. Ho, G.S., Ramamoorthy, C.V.: Protocols for deadlock detection in distributed database systems. IEEE Trans. Softw. Eng. **6**, 554–557 (1982)