World Wide Web
DOI 10.1007/s11280-010-0088-y

An Improved Protocol for Deadlock and Livelock Avoidance Resource Co-allocation in Network Computing

Jie Cao · Zhiang Wu

Received: 8 February 2010 / Revised: 14 March 2010
Accepted: 16 March 2010

Abstract A multitude of applications require simultaneous access to multiple kinds of resources scatted in distributed sites. This problem is known as resource co-allocation which has evolved as a hot topic in network computing. How to design a kind of high-performance protocol for deadlock and livelock avoidance resource co-allocation becomes a challenging problem. In this paper, we propose a new protocol OODP^3 (Optimal ODP^3) which is based on the currently popular protocol ODP^3 (Order-based Deadlock Prevention Protocol with Parallel requests).OODP^3 not only inherits the advantage of ODP^3 but also guarantees the fulfillment of resource co-allocation within polynomial time. Theoretical proof is conducted to verify the correctness of OODP^3. Experimental results also show that OODP^3 achieves the better performance improvements than the existing deadlock and livelock avoidance protocol.

Keywords resource co-allocation · deadlock · livelock · network computing

1 Introduction

Network computing refers to computers working together over a network as opposed to stand alone computers [8]. The early-proposed P2P, subsequently proposed Grid Computing and recently popularized Cloud Computing are all in the scope of network computing. Network computing enables the execution of applications across multiple sites

J. Cao · Z. Wu (✉)
Jiangsu Provincial Key Laboratory of E-business, Nanjing University of Finance and Economics, Nanjing, People's Republic of China
e-mail: zawu@seu.edu.cn

J. Cao
e-mail: caojie690929@163.com

Z. Wu
School of Computer Science and Engineering, Southeast University, Nanjing, People's Republic of China

Published online: 17 April 2010

and internet-wide collaborations become more and more popular. Ian Foster proposed the term *co-allocation* in the area of Grid Computing in 1999 [5]. The term co-allocation refers to the simultaneous access to multiple kinds of resources hosted by autonomous providers, and we comply with this Grid Computing community definition in this paper. However, as the scale of internet has expanded and various applications increased drastically, compared to these increasing applications, resources are relatively scarce, so resource contention is pervasive.

In the case of co-allocating resources for large number of applications, centralized resource co-allocation strategy is infeasible, for it is almost impossible to obtain global resource state information in network computing environment which spans wide-scale physical sites and contains various categories of resources. Therefore, the feasible strategy is that each application, according to resource co-allocation protocol, has to apply for the needed resources from distributed physical sites. A big challenge of designing this resource co-allocation protocol is that the protocol should ensure a set of concurrently active applications which not only apply for resources according to the protocol but also are free of deadlock and livelock. If a set of concurrently active applications are permanently blocked, for each application in the set is allocated and non-preemptively holds resources requested by some other application in the set, this resource allocation state is called deadlock. And its close relative state—livelock happens when applications hold resources preemptively and may obtain and release resources repetitively without progress.

To solve deadlock in distributed resource co-allocation, there often exist two kinds of methods: deadlock detection and deadlock avoidance. Deadlock detection periodically detects the state of the set of concurrently active applications, and once a deadlock is detected, it is recovered by releasing one or more deadlocked applications. This method relays on message transmission among applications. However, since one application cannot know the existence of other applications in network computing environment, let alone message transmission among applications, the deadlock detection method is not suitable for network computing. Therefore, to solve deadlock and livelock in distributed resource co-allocation has to resort to deadlock and livelock avoidance protocol which guarantees that any application will never get into a deadlock or livelock state.

How to design a kind of high-performance protocol for deadlock and livelock avoidance resource co-allocation becomes a challenging problem. Many protocols have been proposed in previous work which will be reviewed in Section 2, and ODP^3 (Order-based Deadlock Prevention Protocol with Parallel requests) is the most efficient protocol proposed by Jonghun Park in 2004 [13]. ODP^3 is of great significance that supports multiple co-allocation requirements of applications and ensures that any application will never get into a deadlock or livelock state. However, ODP^3 reduces its sub-problem, which is used to search next *safe state*, to a knapsack problem which is NP_Complete[4]. Therefore, ODP^3 is a kind of heuristic which cannot guarantee the successful finding of the optimal next safe sate in polynomial time. To remedy this, our work presents an improved ODP^3 called OODP^3(Optimal ODP^3) which ensures that the optimal next safe sate can be found in polynomial time. In a word, OODP^3 proposed in this paper not only inherits the advantage of ODP^3, but also finishes resource co-allocation in polynomial time.

The remainder of this paper has the following structure: in Section 2, related work of deadlock and livelock avoidance protocol is reviewed; in Section 3, resource co-allocation problem is formalized; in Section 4, OODP^3 which is the main contribution of this paper is described in detail; in Section 5, an illustrative example demonstrating the process of OODP^3 is analyzed; in Section 6, experimental results are presented; in Section 7, we make a conclusion and look forward to future work.

2 Related work

The concept of deadlock and its close relative, that of livelock, have been widely investigated in various branches of computer science [1], in particular with regards to networking routing [10, 15]. Our research focuses on deadlock and livelock prevention in resource co-allocation. A large number of research efforts have already been devoted to resource co-allocation in the scope of network computing. Aspects that have been explored include distributed transactions, fault tolerance, inter-site network overhead, and schedule optimization [12]. Our research belongs to the aspect of distributed transactions which conduct the research on protocols in an effort to co-allocate resources for internet applications without getting into deadlock and livelock. In addition, there exist some researches on protocols to maintain strong data consistency which is also an important factor of internet application [7].

According to resource management method, we can divide deadlock and livelock free protocols into two categories. The protocols in the first category are designed for resource broker which manages a set of resources and accepts tasks submitted by users. The representative protocols in the first category are 2PC (Two-Phase Commit) protocol and 3PC (Three-Phase Commit) protocol [6, 16], which can prevent deadlock and livelock when the broker co-allocates resources for applications. The protocols in the second category is designed for applications and allows an application to simultaneously acquire multiple resources from multiple sites which are managed by multiple resource brokers. ODP^2 (Order-based Deadlock Prevention Protocol), ODP^3 and $OODP^3$ proposed in this paper all belong to this category. However, the two categories of protocols are not incompatible. Protocols such as ODP^2, ODP^3 and $OODP^3$ are suitable for users to apply for resources from multiple sites, and each site can utilize 2PC or 3PC to co-allocate resources for accepted tasks. Therefore, 2PC or 3PC are implemented in resource brokers of many software systems, such as RealityGrid [9], GridARS (Grid Advance Reservation-based System) [16], HARC (Highly-Available Robust Co-allocator) [11], etc.

Since our research is closely related to ODP^2 and ODP^3, we investigate ODP^2 and ODP^3 in detail as follows. The early well-known protocol is ODP^2 which defines a global linear ordering of all the resources and requires each application to secure its resources one by one in increasing order according to this ordering. However, ODP^2 only supports one kind of resource co-allocation requirement, but there often exists alternative resource co-allocation requirements for application in network computing. Moreover, with ODP^2, applications are susceptible to long waits and congestion. To overcome these deficiencies of ODP^2, Jonghun Park has proposed ODP^3 which ensures that each application can obtain resources to satisfy one of its alternative resource co-allocation requirements, and prevents the application from getting involved in deadlock and livelock. Compared with ODP^2, ODP^3 can even send parallel requests to increase its efficiency. However, the sub-process of ODP^3—the next safe state search problem, is reduced to knapsack problem. Since knapsack problem is NP_complete, ODP^3 is an exponential time complexity algorithm. In this respect, this paper studies the next safe state search problem from a new perspective and then proposes *NSS* (Next State Search) algorithm for searching the next state. Since *NSS* algorithm is based on lemma 2, the theoretical proof for lemma 2 demonstrates that the next state calculated by *NSS* algorithm is the most optimal and safe. Our improvement can decrease the time complexity of ODP^3 to the polynomial level. ACT (Availability Check Technique) is introduced by Azougagh to reduce the conflicts during the process of resource co-allocation [3]. ACT is a kind of supplementary technique of ODP^2 or ODP^3 for enhancing their performance. So ACT is orthogonal to the problems we study in this paper.

3 Problem formalization

Although resource co-allocation problem has been formalized in [13], to make it clear and easy for understanding, we briefly describe resource co-allocation problem in network computing in this section.

Network computing paradigm connects various kinds of resources in distributed sites to serve for Internet applications. We assume that there are M kinds of resources scattering in many distributed physical sites denoted as $R = \{R_1, R_2, ..., R_M\}$. Each site consists of some kinds of resources, each of which has certain number. We also assume there are K resources in a set of concurrently active applications which competitively apply for those resources hosted by different physical sites. The set of concurrently active applications is denoted as $\text{App} = \{\theta_1, \theta_2, ..., \theta_K\}$.

An application often has several alternative resource co-allocation requirements, any of which can ensure the successful execution of the application. In order to satisfy one of the resource co-allocation requirements, the application will send parallel requests to distributed physical sites. After receiving the requests from concurrently active applications, physical sites allocate certain kinds and numbers of resources to applications. Meanwhile, if the allocated resources cannot satisfy any of the resource co-allocation requirements of a certain application, the application will hold a part of these allocated resources and continue to send parallel requests to distributed physical sites. The application will not start execution until one of its resource co-allocation requirements can be satisfied.

We utilize a FSM (Finite State Machine) to formalize the above-mentioned process, and the FSM is called SD-RAS defined by Jonghun Park [13, 14]. We select SD-RAS as formalization tool for two reasons: (1) OODP^3 which is the main contribution of our work is based on SD-RAS; (2) SD-RAS can describe alternative resource co-allocation requirements conveniently and record every temporary resource allocation state in the process of applying for the resources by the application.

Since SD-RAS is introduced detailedly in [13], this section will briefly introduce SD-RAS for the exposition of OODP^3. Let θ be an application in consideration and M_θ be a finite state machine. Each state $s^i{}_\theta$ in M_θ represents the possible resource combination obtained by θ. $s^0{}_\theta$ which is the initial state of M_θ represents that there is no resource allocated to θ. Each state is represented by means of a multi-set notation, such as $s^i{}_\theta = R_1 + 2R_2$, implying that one unit of resource type R_1 and two units of R_2 are co-allocated at state $s^i{}_\theta$. Notation $s^i{}_\theta(R_j)$ represents the number of resource type R_j are allocated at state $s^i{}_\theta$. We also define goal states set which is notated as G_θ. Each element in G_θ represents one of the alternative resource co-allocation requirements of the application θ.

Figure 1 shows an example of SD-RAS. There are two goal states which are $s^5{}_\theta$ and $s^7{}_\theta$ respectively, namely $G_\theta = \{R_1 + 2R_3, 2R_2\}$. We define R_θ as the set of resources which are elements of multi-sets in G_θ. In this example, $R_\theta = \{R_1, R_2, R_3\}$.

It is the fact that some sequences of resource allocation may lead to a deadlock, and thus not every possible state of an application should be permitted in order to prevent deadlock. For this purpose, Park proposed the concept of *safe state* that characterizes the middle state from which an application may successfully reach one of its goal states without getting into deadlock. Park firstly proposes the concept of safe path, and then on basis of this concept, proposes the concept of safe state. Then a lemma to determine safe state is proposed and proofed. Since related results of the safe state are the basis of this paper instead of the focal point, we only give the definition of safe state and safe state

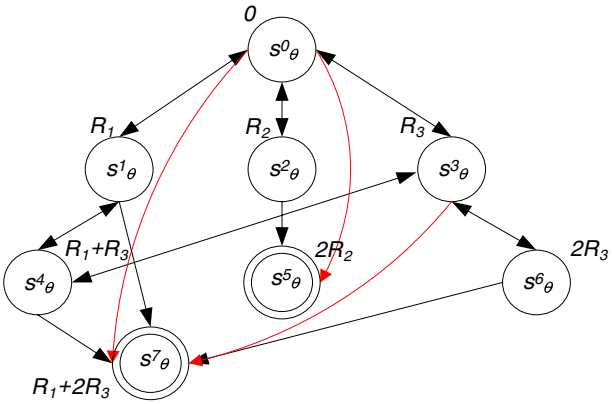

Figure 1 An example of SD-RAS.

determination lemma without proofing this lemma. The definition and lemma are described as follows:

Definition 1 Let θ be an application in SD-RAS. A path of finite length in M_θ, $< s^{i1}\theta, ..., s^{in}\theta >$, such that $n>1$ and $0 < i_l \le L_\theta$, for any $l=1,...,n$, is said to be a *safe path* if (1) $s^{il}{}_\theta \in G_\theta$, and (2) $s^{il}{}_\theta(R_j) \ge s^{il+1}{}_\theta(R_j)$, for any $l=1,...,n$ and $R_j \in R_\theta$, such that $o_j \ge \pi^{il}{}_\theta$, where $\pi^{il}{}_\theta = \min\{o_k | S^{il}{}_\theta(R_k) > 0,\ R_k \in R_\theta\}$.

Definition 2 Given application of SD-RAS, its local resource allocation state, $s^i{}_\theta \notin G_\theta$ such that $0 < i \le L_\theta$, is *safe* if there exists a safe path starting from $s^i{}_\theta$.

Lemma 1 *For application of SD-RAS, its local resource allocation state,* $s^i{}_\theta$, *such that* $0 < i \le L_\theta$ *and* $s^i{}_\theta \notin G_\theta$ is *safe if there exists a goal state* $s^i{}_\theta \in G_\theta$ *such that* $s^i{}_\theta(R_k) \ge s^j{}_\theta(R_k)$, *for any* $Rk \in R_\theta$ *satisfying* $o_k \ge \pi^i{}_\theta$.

In this paper, we assume that $o_1 < o_2 < ... < o_i < ... < o_M$, and o_i represents the order of the *i-th* kind resource R_i. Therefore, in Figure 1, only $s^6{}_\theta$ is a safe state, and $s^1{}_\theta, s^2{}_\theta, s^3{}_\theta$ are not safe state.

4 Optimal ODP^3

In this section, we present the main contribution OODP^3 (Optimal ODP^3) of this paper. The discussion of OODP^3 is divided into two parts. First, the main process of OODP^3 is proposed. Although the main process of OODP^3 is similar to ODP^3 proposed by Park, two improvements are made. Second, the sub-process of searching next safe state is presented. By means of proofing the safety and optimality of the next state calculated by lemma 2, the correctness of *NSS* algorithm is verified.

4.1 Main process of OODP^3

As the resource co-allocation process is recalled in Section 3, an application sends requests to resource providers in different physical sites. After requests are received, resource providers often allocate part of the required resources to the application. Then the application sends

requests again for the remaining required resources. In SD-RAS, the state in which the application holds part of its required resources and waits for another resource allocation from resource providers is a middle state in FSM. OODP^3 guarantees the safety of every middle state in the path which is from the initial state to one of the goal state, thus avoiding both deadlock and livelock. The pseudocode of main process of OODP^3 is as follows.

Algorithm 1 The main process of OODP^3

Input goal states set G_θ of a certain application θ

Output the satisfied goal state $G^l{}_\theta$

Description

1: $Q_\theta \leftarrow \emptyset$; /*Initialize the up-to-now allocated resource set Q_θ*/

2: **while** (TRUE) **do**

3: $T_\theta \leftarrow$ request $G^+{}_\theta \backslash Q_\theta$; /*Send requests to resource providers*/

4: **for** $l \leftarrow 1$ **to** N

5: **if** ($G^l{}_\theta \backslash (Q_\theta \cup T_\theta) = \emptyset$) /*Judge one of the goal state whether it is satisfied*/

6: cancel $(Q_\theta \cup T_\theta) \backslash G^l{}_\theta$; /*If $G^l{}_\theta$ is satisfied, aborting the unwanted resources*/

7: **return** *l;* /*Return $G^l{}_\theta$,OODP^3 terminates*/

8: **end if**

9: **end for**

10: **if** (!safe($Q_\theta \cup T_\theta$)) /*If none of the goal states is satisfied, judge the safety of $Q_\theta \cup T_\theta$*/

11: $T'{}_\theta \leftarrow NSS(Q_\theta, T_\theta)$; /*If $Q_\theta \cup T_\theta$ is not safe, invoke *NSS* algorithm to obtain $T'{}_\theta$*/

12: cancel $T_\theta \backslash T'{}_\theta$;

13: $Q_\theta \leftarrow Q_\theta \cup T'{}_\theta$; /*Allocate $T'{}_\theta$ to the application θ and update Q_θ*/

14: **else**

15: $Q_\theta \leftarrow Q_\theta \cup T_\theta$ /*If $Q_\theta \cup T_\theta$ is safe, allocate T_θ to the application θ and update Q_θ*/

16: **end if**

17: **end while**

In the above mentioned pseudocode of main process of OODP^3, notation $G^+{}_\theta$ represents globally required kind and number of resources and it is calculated by Eq. 1.

$$G^+_\theta(R_i) = Max^N_{i=1} G^i_\theta(R_i) \quad \text{where } \mathrm{G}_\theta = \left\{\mathrm{G}^1{}_\theta, \mathrm{G}^2{}_\theta, ..., \mathrm{G}^\mathrm{N}{}_\theta\right\} \tag{1}$$

Equation 1 shows that each kind of resource in $G^{+}{}_{\theta}$ is the maximum of the very kind of resource in G_{θ}. The definition of operator "\" as is shown in Eq. 2.

$$S_1 \backslash S_2 = \begin{cases} Max\{(S_1(R_i) - S_2(R_i)), 0\}, & if\ R_i \in S_1 \\ 0, & otherwise \end{cases} \quad (2)$$

In line 3, the application sends request to apply for resources recorded in the multiset $G^{+}{}_{\theta} \backslash Q_{\theta}$, which represents up-to-now lacking kind and number of resources. After the results of the multicast of requests for resources are received, the resources which are successfully allocated are temporarily recorded in T_{θ}. Since notation Q_{θ} represents the up-to-now allocated resource set, from line 4 to 9, by means of checking all goal states in G_{θ}, OODP^3 judges whether there exists one of the goal state satisfied by the multiset $Q_{\theta} \cup T_{\theta}$. If one of the goal state notated as $G^{l}{}_{\theta}$ is satisfied, OODP^3 terminates and returns $G^{l}{}_{\theta}$. Otherwise, OODP^3 switches to the line 10, and the safety of the multiset $Q_{\theta} \cup T_{\theta}$ is judged according to lemma 1. If the multiset $Q_{\theta} \cup T_{\theta}$ is safe, none of the resources in T_{θ} is aborted and all resources in T_{θ} are added to Q_{θ}. Then OODP^3 goes back to the line 3 and new requests for the remaining required resources are sent. If the multiset $Q_{\theta} \cup T_{\theta}$ is not safe, it is necessary to drop out some of the resources in T_{θ} to make the state safe. *NSS* algorithm is invoked to obtain T'_{θ} which is the subset of T_{θ} and the safety of $Q_{\theta} \cup T'_{\theta}$ is guaranteed. Then in line 12 and 13, the unwanted resources in T_{θ} are aborted and all resources in T'_{θ} are added to Q_{θ}. It is worth noting that no resource in Q_{θ} is allowed to be aborted and the application θ can always go back to a safe state by releasing all the resources in T_{θ}.

Although the main process of OODP^3 is similar to ODP^3, two improvements have been made:

(1) In line 11, T'_{θ} that is the subset of T_{θ} is determined by the *NSS* algorithm which will be presented in next section.
(2) Symbol $\uplus$ in ODP^3 is changed to symbol $\cup$in OODP^3. According to OODP^3, the application θ holds the sum of Q_{θ} and T_{θ} (or T'_{θ}) instead of the maximum of Q_{θ} and T_{θ} (or T'_{θ}). For instance, θ requires three units of R_1 in total. The first request makes θ obtain one units of R_1, namely $Q_{\theta}(R_1)=1$. The results of the second request contain two units of R_1, namely $T_{\theta}(R_1)=1$. $Q_{\theta} \uplus T_{\theta}(R_1)=2$. However, it is obvious that θ should hold three units of R_1,namely $Q_{\theta} \cup T_{\theta}(R_1)=3$.

4.2 Next state search algorithm

The *NSS* (Next State Search) algorithm is the sub-process of the OODP^3, and it aims to search the next safe state that the time to reach one of its goal states is minimized. The time to reach one of the goal states is measured by a value function called objective function. When $Q_{\theta} \cup T_{\theta}$ is not safe, it is necessary to drop out some of the resources in T_{θ} to make the state safe. What should be noted is that the application holds the resources in Q_{θ} permanently. However, in order to achieve progress, OODP^3 seeks to drop out as few resources as possible while it ensures the safety of the next state. Therefore, though the objective function can be defined in various ways, it should be a non-increasing function with the decrease of the number of resources in T_{θ}. Definition 3 is given as follows with which all the objective functions should comply.

Definition 3 Given an objective function *f(*)*, if the number of resources in T'_{θ} is smaller or equal to the number of resources in T''_{θ}, $f\left(Q_{\theta} \cup T'_{\theta}\right) \geq f\left(Q_{\theta} \cup T''_{\theta}\right)$ is satisfied.

Park utilizes distance as the objective function which is referred to the similar concept in CPN (Colored Petri Net). The definition of distance is depicted as Eq. 3. It is obvious that Eq. 3 complies with definition 3.

$$distance\left(Q_{\theta}^{'}\right) = \frac{\#\left(\uplus_{i=1}^{n_\theta}\left(G_{\theta}^{i} \backslash Q_{\theta}^{'}\right)\right)}{n_\theta} \tag{3}$$

If we consider all the subsets of T_θ to search one of it satisfying (1) is safe, and (2) *f(*)* is minimized, it is a NP_Complete problem for the number of subsets is exponential. However, we find not all of the subsets of T_θ should be considered, for lemma 2 is satisfied. Lemma 2 is described as follows:

Lemma 2 *Let* $S_\theta = Q_\theta \cup T_\theta$ *and* T'_θ *is a subset of* T_θ. *We assume* $S'_\theta = Q_\theta \cup T'_\theta$ *is the optimal safe state, namely* $f(S'_\theta)$ *is minimized.* T'_θ *is the multiset that must be derived from one of the goal states notated as* G^l_θ. *We also assume* G^l_θ *consists of m kinds of resources and represent as* $R_{l1}, R_{l2}, \ldots, R_{lm}$. *To simplify our discussion, we define* T^*_θ *which satisfies* $T'_\theta = T_\theta - T^*_\theta$ *and the calculation of* T^*_θ *is divided into two cases:* (1) $\vee k \in [1,i], Q(R_k) = 0$, $T^*_\theta = T_\theta(R_1)R_1 + T_\theta(R_2)R_2 + \ldots + T_\theta(R_i)R_i$; (2) *existing* $k \in [1,i], Q(R_k) \neq 0, T^*_\theta = T_\theta$. Where $i = \min\{k | S_\theta(R_{lp}) \geq G^l_\theta(R_{lp}), \vee p = k+1, \ldots, m, Rlp \in G^l_\theta\}$.

Proof (1) We prove S'_θ is safe state.

- **Case (1)** Because $\vee k \in [1,i]\, Q(R_k) = 0$ is satisfied, all resource categories in [1,i] of S_θ are from T_θ. Meanwhile, since $S'_\theta = S_\theta - T^*_\theta$ and $T^*_\theta = T_\theta(R_1)R_1 + T_\theta(R_2)R_2 + \ldots + T_\theta(R_i)R_i$, $\vee k \in [1,i]\, S'_\theta(R_k) = 0$ is satisfied. i is the minimal serial number of resources in G^l_θ that satisfies $S_\theta(R_{lp}) \geq G^l_\theta(R_{lp})$. Then, a conclusion can be drawn that $\vee p \in [i+1, m] \vee R_{lp} \in G^l_\theta, S_\theta(R_{lp}) \geq G^l_\theta(R_{lp})$ satisfies, and $\vee k \in [i+1, m]\, S'_\theta(R_k) = S_\theta(R_k) \geq G^l_\theta(R_{lp})$ also satisfies. Therefore, the state S'_θ complies with Lemma1, so S'_θ is a safe state.
- **Case (2)** Since the main process of OODP[3] sends parallel resource requests on the pre-condition of safe state, the prior state S_θ is safe. At the same time, $S^*_\theta = T_\theta$ and $S'_\theta = S_\theta$, so S'_θ is a safe state.

(2) We prove safe states derived from G^l_θ cannot be more optimal than S'_θ.

- **Case (1)** We assume that there exists a safe state S''_θ derived from G^l_θ is more optimal than S'_θ. Because the objective function defined from definition 3 is a non-increasing function, the resource amounts in S''_θ is more than those in S'_θ. If we consider $S''_\theta = S_\theta - T^{**}_\theta$, then the resource amounts in T^{**}_θ is less than those in T^*_θ. Since $T^*_\theta = T_\theta(R_1)R_1 + T_\theta(R_2)R_2 + \ldots + T_\theta(R_i)R_i$, if we exclude any resource in T^*_θ, and assume this excluded resource is R_j, $T^{**}_\theta = T^*_\theta - R_j$. Meanwhile, because $\pi^j_\theta = \min\{o_k | S''_\theta(R_k) > 0, R_k \in R_\theta\}$, $\pi^i_\theta \geq \pi^j_\theta$ exists and $S_\theta(R_i) \geq G^l_\theta(R_i)$ satisfies. Therefore, the state S''_θ does not complies with lemma 1. In conclusion, S''_θ is not safe and the conflict is drawn.
- **Case (2)** Since there exists $k \in [1,i]\, Q(R_k) \neq 0$ and $S_\theta(R_k) \leq G^l_\theta(R_k)$, if we want to obtain a new safe state from G^l_θ, it is necessary for us to exclude R_k in Q_θ at least. However, the main process of OODP[3] does not allow any resource to be excluded from Q_θ. Therefore, there does not exist any new safe state derived from G^l_θ and it is obvious that there is not any safe state more optimal than S'_θ. □

From lemma 2, we find out that the optimal next safe state, namely the subset of S_θ whose *f(*)* is minimized, is surely constructed on the basis of one of the goal states. Therefore, if we construct the local optimal next safe states on the basis of all goal states, we can obtain the optimal next safe state just by selecting the minimized *f(*)* among these local optimal next safe states. The *NSS* algorithm carries out according to the idea mentioned above, and the pseudocode of the *NSS* algorithm is depicted as follows.

Algorithm 2 The sub-process of OODP^3—*NSS*

Input the allocated resource set Q_θ, results of the requests T_θ

Output the optimal next safe state $T'_\theta[k]$

Description

1: **for** $l \leftarrow 1$ **to** N /*N is the number of goal states in G_θ*/

2: $i \leftarrow$ M;

3: **while** $(Q_\theta \cup T_\theta(\mathrm{R_i}) \geq G^l_\theta(\mathrm{R_i}))$

4: $i \leftarrow i$-1; /* Determine the value of i in lemma 2*/

5: **end while**

6: **for** $j \leftarrow 1$ **to** i /*Judge $j \in [1,i]$ whether $Q_\theta(\mathrm{Rj}) \neq 0$ satisfies*/

7: **if** $(Q_\theta(\mathrm{R_j}) \text{ !=0})$ **then** /* The case (2) of lemma 2 is satisfied*/

8: $T'_\theta[l] \leftarrow \emptyset$; /* $T'_\theta[l]$ derived from G^l_θ is null*/

9: **Go to step 1;** /*Check next goal state*/

10: **end if**

11: **end for**

12: $T^*_\theta[l] \leftarrow T_\theta(\mathrm{R_1})\mathrm{R_1} + T_\theta(\mathrm{R_2})\mathrm{R_2} + \ldots + T_\theta(\mathrm{R_i})\mathrm{R_i}$; /* The case (1) of lemma 2 is satisfied*/

13: $T'_\theta[l] \leftarrow T_\theta - T^*_\theta[l]$;

14: **end for**

15: $k \leftarrow$ Min (f($T'_\theta[l]$),l=1,2,...,N); /* Select the minimized *f(*)* among $T'_\theta[l]$,l=1,2,...,N */

16: **return** $T'_\theta[k]$; /* $f(T'_\theta[k])$ is minimized*/

In the above mentioned *NSS* algorithm, the subset of T_θ which is derived from G^l_θ is denoted as $T'_\theta[l]$ l∈[1,N], and the minimized *f(*)* among all $T'_\theta[l]$ is represented as $T'_\theta[k]$. Therefore, $Q_\theta \cup T'_\theta[k]$ is the optimal next safe state in OODP^3. From the line 2 to 5 of the pseudocode of *NSS* algorithm, the value of i in lemma 2 is calculated. From line 6 to 10, *NSS* algorithm judges whether for any $j \in [1, \mathrm{i}]$ $Q_\theta(\mathrm{R_j}) \neq 0$ is satisfied. If so, the case (2) of lemma 2 is satisfied, and $T'_\theta[l]$ which is derived from G^l_θ is null. Otherwise, the case (1) of

lemma 2 is satisfied, and $T'_{\theta}[l]$ is evaluated in line 13. Up to line 14, all of the subsets derived from N goal states are calculated. Then, in line 15, the minimized *f(*)* is extracted and returned to the main process of the OODP^3.

The complexity of *NSS* algorithm is $O(NM)$ which is polynomial function. Since the complexity of OODP^3 relies on the number of goal states, the average number of goal sates is represented as $\mathrm{Avg}(|G^l{}_{\theta}|, 1 = 1, 2, ..., N)$ and the complexity of OODP^3 is $O(\mathrm{Avg}(|G^l{}_{\theta}|, 1 = 1, 2, ..., N)NM)$ which is also polynomial function.

5 Illustrative example

To facilitate the understanding of OODP^3, we give an illustrative example in this section. In this example, there exist ten kinds of resources, each kind of which has 20 resources. Meanwhile, there exist five tasks in this example and each task has three goal states. The input data configuring these five tasks is listed in Table 1.

We assume that resource providers process the requests according to the order from 1 to 5. Table 2 describes the process of applying for resources by these five tasks in accordance to OODP^3. At the starting time, since the amount of resources is adequate, the first goal state of task 1 is satisfied. During the process of task 2, only the third goal state of task 2 can be satisfied. When task 3 is processed, none of its goal states can be satisfied, and task 3 finds the next safe state according to *NSS* algorithm, as is shown in line 4 of Table 2. The goal states of task 4 and task 5 also cannot be satisfied, and they find their next safe states according to *NSS* algorithm, as is shown respectively in line 6 and 8 of Table 2. After 16 s, task 1 is completed and releases the occupied resources. Then, the first goal state of task 3 can be satisfied and it releases other unwanted resources, as is shown in line 12 of Table 2. Since the process from line 13 to the end of Table 2 is similar with the above-mentioned process and in order to simplify our analysis, we will not discuss it in detail.

Table 1 The input data of example.

Task number	R1	R2	R3	R4	R5	R6	R7	R8	R9	R10	Execution time
1	17	15	18	14	8	0	0	0	0	0	16
	0	2	20	5	7	19	0	0	0	0	
	0	0	16	17	3	10	18	0	0	0	
2	0	0	0	20	20	15	7	6	0	0	43
	0	0	0	0	16	3	13	18	5	0	
	0	0	0	0	0	15	18	8	13	20	
3	16	14	18	5	7	0	0	0	0	0	11
	0	17	16	16	9	9	0	0	0	0	
	0	0	20	3	6	4	8	0	0	0	
4	0	0	0	18	20	3	6	3	0	0	20
	0	0	0	0	20	16	20	17	5	0	
	0	0	0	0	0	12	8	10	5	6	
5	12	16	17	8	7	0	0	0	0	0	18
	0	16	10	12	4	16	0	0	0	0	
	0	0	17	11	16	6	1	1	0	0	

Table 2 The process of applying for resources by the five tasks in accordance to OODP[3].

Line	State	R1	R2	R3	R4	R5	R6	R7	R8	R9	R10
1	The 1st goal state of task 1 is satisfied	17	15	18	14	8	0	0	0	0	0
2	The 3rd goal state of task 2 is satisfied	0	0	0	0	0	15	18	8	13	20
3	Remaining Resources	3	5	2	6	12	5	2	12	7	0
4	NSS results for task 3	0	0	0	6	9	5	2	0	0	0
5	Remaining Resources	3	5	2	0	3	0	0	12	7	0
6	NSS results for task 4	0	0	0	0	0	0	0	12	5	0
7	Remaining Resources	3	5	2	0	3	0	0	0	2	0
8	NSS results for task 5	0	0	0	0	0	0	0	0	0	0
9	After 16 s, task 1 is completed and releases the occupied resources	17	15	18	14	8	0	0	0	0	0
10	Remaining Resources	20	20	20	14	11	0	0	0	2	0
11	The 1st goal state of task 3 is satisfied	16	14	18	5	7	0	0	0	0	0
12	Resources released by task 3	0	0	0	1	2	5	2	0	0	0
13	Remaining Resources	4	6	2	15	13	5	2	0	2	0
14	NSS results for task 4	0	0	0	0	0	0	0	12	5	0
15	NSS results for task 5	0	0	0	12	13	5	1	0	0	0
16	After 11 s, task 3 is completed and releases the occupied resources	16	14	18	5	7	0	0	0	0	0
17	Remaining Resources	20	20	20	8	7	0	1	0	2	0
18	NSS results for task 4	0	0	0	0	0	0	0	12	5	0
19	The 1st goal state of task 5 is satisfied	12	16	17	8	7	0	0	0	0	0
20	Resources released by task 5	0	0	0	4	6	5	1	0	0	0
21	Remaining Resources	8	4	3	12	13	5	2	0	2	0
22	After 5 s, task 2 is completed and releases the occupied resources	0	0	0	0	0	15	18	8	13	20
23	Remaining Resources	8	4	3	12	13	20	20	8	15	20
24	The 3rd goal state of task 4 is satisfied	0	0	0	0	0	12	8	10	5	6

6 Experimental results

We have built a simulation system which can evaluate the performance of various protocols for resource co-allocation. The simulation system is implemented by MATLAB and Java. Program written by MATLAB is used to produce input data which consists of a set of applications described by multiple resource requirements and its execution time. Program written by Java reads the input data and achieves resource co-allocation according to various protocols. The output of the simulation system is the Gantt chart of each resource, which can depict the state of each resource in each moment. To start a simulation of resource co-allocation, we should configure various parameters listed in Table 3.

Parameters *M, n* and N_i are the main factors influencing the performance of resource co-allocation protocols. We change these three main parameters in different experimental scenes. To simplify our experiments, we assume that the numbers of resources in all categories *c* and the numbers of categories contained in each goal state *CR* are equal. We also assume the execution time of *i-th* application ET_i is a random number between 10 s and 1,000 s.

Table 3 Experimental parameters.

Notations	Description
M	The number of categories of resources
c	The number of resources in each category
n	The number of concurrently active applications
ET_i	The execution time of the *i-th* application
N_i	The number of goal states of the *i-th* application
CR	The number of categories contained in each goal state
RT_i	The back-off time of the *i-th* application

Back-off time is the time between two successive multicasts when an application fails to obtain the resources that satisfy one of its goal states. Since the execution time takes second as a unit in our simulation, the minimized back-off time is 1 s. But such short back-off time will lead to a large volume of message transmission and increased resource contention. In turn, too long back-off time will result in increased waiting time of applications and low resource utilization. Though how to design the optimal back-off time is a complex problem [2], the setting of the back-off time as 1 s will not influence the evaluation of the performance of resource co-allocation protocols. Therefore, this paper will not make further study on designing the optimal back-off time and thus set the back-off time as 1 s in all experimental scenes.

Another factor influencing the performance of protocols is the resource allocation policy of resource providers. To simplify our experiments, we adopts FIFO (First In First Out) policy to process requests sent by applications and utilizes the best-effort policy to process each request.

We have conducted two groups of experiments to evaluate the performance of OODP^3. Because OODP^3 is the same as ODP^3 when ODP^3 utilizes an exhaustive search to obtain the next safe state, we compare OODP^3 with ODP^3 on their execution time instead of performance metrics in the first group of experiments. Since OODP^3 and ODP^2 are entirely different, we compare OODP^3 with ODP^2 on two performance metrics in the second group of experiments.

6.1 OODP^3 vs. ODP^3

The first group of experiments compares OODP^3 with ODP^3. The difference between OODP^3 and ODP^3 is the calculation of the next safe state, i.e. the subset of T_θ. If ODP^3 utilizes an exhaustive search to obtain the next safe state, ODP^3 will be the same as OODP^3. Figure 2 shows the comparison of the execution time of OODP^3 and ODP^3 according to the increase of n from 5 to 50. We set M as 10, c as 20, N_i as 3, CR as 5 and RT_i as 1. In this experiment, since the execution time refers to the time used by the program of OODP^3 or ODP^3, we take millisecond as the unit of the execution time. The programs of OODP^3 and ODP^3 are run on a standard PC with two 2.33 GHz Quad8200 processors and 3 GB memory.

The comparison of the plots in Figure 2 indicates that the execution time of OODP^3 is always shorter than ODP^3. Figure 2 also shows that with the increase of n, the execution time of both OODP^3 and ODP^3 does not always increases, for the execution time relies on actual input data including the categories and numbers of all three requirements of each application.

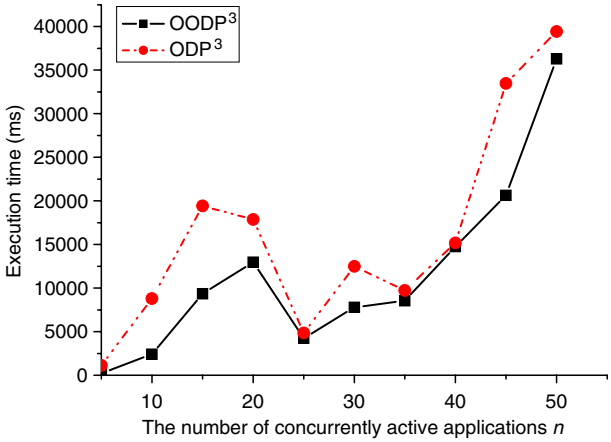

Figure 2 Comparison of the execution time between $OODP^3$ and ODP^3.

6.2 $OODP^3$ vs. ODP^2

The second group of experiments compares $OODP^3$ with ODP^2. Since ODP^2 only supports single goal state, ODP^2 selects one of goal states of each application randomly. Under ODP^2, the application applies for resources according to resource ordering and holds resources only if the allocated number of this resource category is greater than or equal to the required number specified in the selected goal state. We firstly define two performance metrics as follows.

(1) **AWT (Average Waiting Time)** Waiting time refers to the consumed time from when an application sends the first request to the time when it starts execution. AWT is the average of waiting time of all concurrently active applications.
(2) **RU (Resource Utilization)** This metric is the average percentage ratio of total resources used by applications, and it can be calculated by Eq. 4 where *Makespan* is

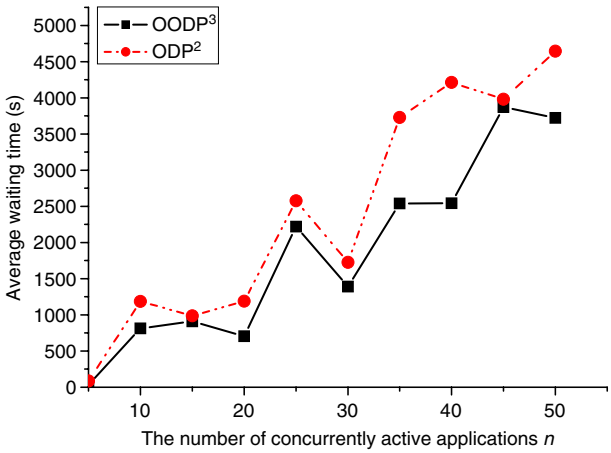

Figure 3 Comparison of AWT between $OODP^3$ and ODP^2 with the alteration of *n*.

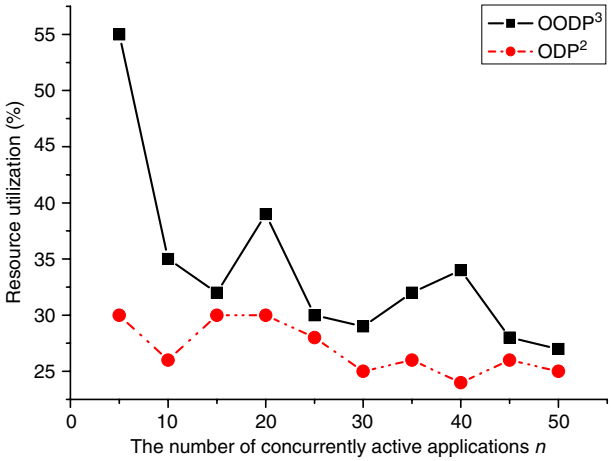

Figure 4 Comparison of RU between OODP[3] and ODP[2] with the alteration of *n*.

defined as the amount of time from the initial time to the time when all applications are finished.

$$RU = \frac{\sum_{i=1}^{Mc} RT_i}{M \times c \times Makespan} \tag{4}$$

Figures 3 and 4 show the comparison of AWT and RU according to the increase of *n* from 5 to 50 respectively. The comparison of the plots in Figure 3 indicates that OODP[3] and ODP[2] exhibit the similar performance in some plots, for in these cases, the required numbers of resources specified in goal states are small and then the applications can easily apply for resources. However, when the required numbers of resources specified in goal states become large, OODP[3] achieves better improvements than ODP[2]. The comparison of RU depicted in Figure 4 shows the similar characteristic with Figure 3. Both AWT and RU of OODP[3] are better than those of ODP[2] in any case. We argue that the discrepancy of the

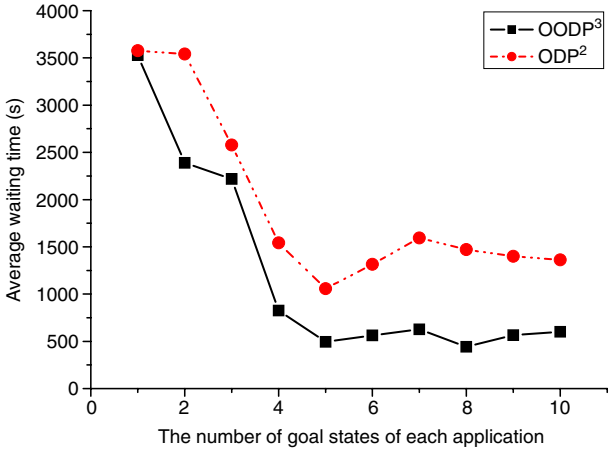

Figure 5 Comparison of AWT between OODP[3] and ODP[2] with the alteration of *N*.

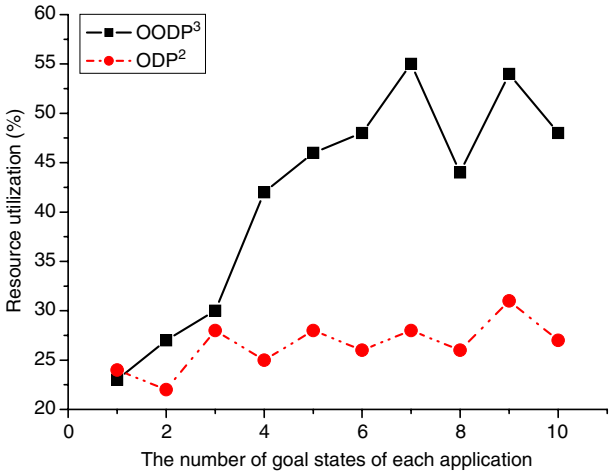

Figure 6 Comparison of RU between $OODP^3$ and ODP^2 with the alteration of N.

performance of $OODP^3$ and ODP^2 is not only determined by the number of concurrently active applications, which is described by mean inter-arrival time in [13], but also affected by the actual required number of each resource category specified in goal states. Therefore, both AWT and RU do not always manifest monotonously increase or decrease with the increase of n.

Figures 5 and 6 demonstrate the comparison of AWT and RU according to the increase of N from 1 to 10 respectively. When there is only one goal state, the AWT of $OODP^3$ and ODP^2 are nearly the same and the RU of ODP^2 is a bit higher than it of $OODP^3$. With the increase of N, AWT of $OODP^3$ decreases dramatically and RU of increases dramatically, while both of AWT and RU of ODP^2 vary rulelessly. It is indicated that with the number of goal state increasing, $OODP^3$ can select alternative co-allocation requirements flexibly, which results the decrease of AWT and the increase of RU. Since ODP^2 only supports one goal state, the increase of N does not influence its performance.

7 Conclusion

In this paper, we study deadlock and livelock prevention protocol for resource co-allocation in network computing. Since the protocol proposed in this paper is on the basis of ODP^3 and ensures the finding of the most optimal next safe state within polynomial time, we call this protocol $OODP^3$. Theoretical proof verifies the correctness of $OODP^3$. Algorithm complexity analysis and experimental results all demonstrate that $OODP^3$ is much more efficient than ODP^3. An illustrative example demonstrating the process of $OODP^3$ is analyzed to facilitate the understanding of $OODP^3$. Experimental results also show that $OODP^3$ achieves the better improvements than ODP^2 on the performance metrics of both AWT and RU. In the future, we plan to study how to design back-off time function to further improve the performance of $OODP^3$.

Acknowledgements This research is supported by the program for New Century Excellent Talents in university (NCET-07-0411), the Natural Science Foundation for key basic research of the Jiangsu Higher Education Institutions of China (07KJA52004) and the forward looking Production, Teaching & Research union project of Jiangsu Province of China (BY20091005).

References

1. Abate, A., D'Innocenzo, A., Benedetto, M.D.D., Sastry, S.: Understanding deadlock and livelock behaviors in hybrid control systems. Nonlinear Anal Hybrid Syst **3**(2), 150–162 (2009)
2. Aldous, D.J.: Ultimate instability of exponential back-off protocol for acknowledgment-based transmission control of random access communication channels. IEEE Trans Inf Theory **IT-33**(2), 219–223 (1987)
3. Azougagh, D., Yu, J.L., Kim, J.S., Maeng, S.R.: Resource co-allocation: a complementary technique that enhances performance in grid computing environment. In: Barolli, L. (ed.) International conference on parallel and distributed systems (ICPADS 05), vol. 1, pp. 36–42. IEEE Computer Society, Los Alamitos (2005)
4. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to algorithms. The MIT (2001)
5. Czajkowski, K., Foster, I., Kesselman, C.: Resource co-allocation in computational grids. International Symposium on High Performance Distributed Computing, pp. 219–228. IEEE Computer Society, Los Alamitos (1999)
6. Haji, M.H., Gourlay, I., Djemane, K., Dew, P.M.: A SNAP-based community resource broker using a three-phase commit protocol: a performance study. Comput J **48**(3), 333–346 (2005)
7. Hossein Sheikh Attar, M., Tamer Özsu, M.: Alternative architectures and protocols for providing strong consistency in dynamic web applications. World Wide Web **9**(3), 215–251 (2006)
8. http://en.wikipedia.org/wiki/Network_computing
9. Kuo, D., Mckeown, M.: Advance reservation and co-allocation protocol for grid computing. In: Stockinger, H., Buyya, R., Perrott, R. (eds.) International conference on e-science and grid technologies, pp. 164–171. IEEE Computer Society, Los Alamitos (2005)
10. Levitin, L., Karpovsky, M., Mustafa, M.: Deadlock prevention by turn prohibition in interconnection networks, Proceedings of the 2009 IEEE International Symposium on Parallel & Distributed Processing (IPDPS 09)
11. Maclaren, J., Keown, M.N., Pickles, S.: Co-allocation, fault tolerance and grid computing. In: Cox, S.J. (ed.), UK e-science all hands meeting, pp. 155–162. NeSC (2006)
12. Netto, M., Buyya, R.: Resource co-allocation in grid computing environments, handbook of research on P2P and grid systems for service-oriented computing: models, methodologies and applications. Antonopoulosa, N., Exarchakosa, G., Li, M., Liottac, A. (eds.), IGI Global, USA (Feb. 2010).
13. Park, J.: A deadlock and livelock free protocol for decentralized internet resource coallocation. IEEE Trans Syst Man Cybern Part A Syst Humans **34**(1), 123–131 (2004)
14. Park, J., Reveliotis, S.A.: Deadlock avoidance in sequential resource allocation systems with multiple resource acquisitions and flexible routings. IEEE Trans Automat Contr **46**, 1572–1583 (2001)
15. Solheim, A.G., Lysne, O., Bermudez, A., Casado, R., Sodring, T.: Efficient and deadlock-free reconfiguration for source routed networks, Proceedings of the 2009 IEEE International Symposium on Parallel & Distributed Processing (IPDPS 09)
16. Takefusa, A., Nakada, H., Kudoh, T., Tanaka, Y., Sekiguchi, S.: GridARS: an advance reservation-based grid co-allocation framework for distributed computing and network resources. In: Frachtenberg, E., Schwiegelshohn, U. (eds.) International workshop on job scheduling strategies for parallel processing. Springer, Berlin (2007)