マルチメディア通信と分散処理 86－2
グループウェア 26－2
（1998. 1. 29）

Distributed Allocation of Resources to Overlapping Process Groups

Zixue CHENG　Yukiko INOUE　Yutaka WADA

Department of Computer Software, University of Aizu

The distributed resource allocation problem is a well known fundamental problem in distributed systems. Many solutions which avoid the deadlock and starvation have been developed. With the progress of computer networks, however, distributed cooperative group activities in a network environment have been increasing, so that several groups may compete for some resources in the network environment and *deadlock among groups* and *starvation of a group* may happen. Since previous allocation models are mainly for representation of competition for resources among processes, they can't reflect clearly the competition for resources among groups of processes. Moreover, though the previous solutions to the distributed resource allocation problem can avoid the deadlock and starvation, they can't deal with the deadlock among groups and starvation of a group.

In this paper, we propose a solution which allocates resources to groups of processes with deadlock among groups and starvation of a group never happening, by extending an acyclic graph approach to the dining philosopher problem. In addition, our solution guarantees that more than one group work mutual exclusively, if a common process belongs to these groups.

共有プロセスを含むグループ間の分散資源割り当て

程 子学　井上 友紀子　和田 裕

会津大学コンピュータソフトウェア学科

分散システムにおける資源の割り当て問題は，いままで盛んに研究されており，デッドロックや飢餓状態を回避する分散アルゴリズムは多数開発されている．しかしながら，コンピュータネットワークの発展に伴い，ネットワークを通じて様々なグループ協調活動を行なうことが増えるため，複数のグループはネットワーク上の複数の資源を競合し，グループ間のデッドロックやグループの飢餓状態を起こす危険性が生じてきた．今までの資源割り当てモデルは，プロセス間の資源競合を表すもので，グループ間の資源競合を明示的に表すことができない．また，今までの分散アルゴリズムは，デッドロックと飢餓状態を回避できるが，グループ間のデッドロックとグループの飢餓状態には対応できない．そこで，本論文では，まず，グループ間の資源競合を明示的に表す資源割り当てモデルを与え，その上にグループ間のデッドロックとグループ飢餓状態の定義を与える．そして，これらの問題を解決するため，従来の有向無閉路に基づいた資源割り当て法を拡張した分散アルゴリズムを提案する．今までの研究と比べて，本研究は，複数のグループがあるプロセスを共有する場合でも有効であるという特徴を持つ．

1 Introduction

A fundamental problem in distributed systems is the resource allocation problem, which could be briefly summarized as follows.

In a distributed system, there are a set of processes and a set of resources. Every process requires a subset of the resources and has to acquire all required resources for the process to perform its task. Every resource can be allocated to at most one process at a time. That is, the processes which require the resource have to access the resource mutual exclusively [1][2][3][4][6].

There are many variations of and extensions to the resource allocation problem. For example, the k-out of-M model [9][10] which requires arbitrary k instances from M instances of a resource and models [5][9] which considers types of resources, are proposed.

When we consider the solution the resource allocation problem, the states that no process can use resources and some process can never use resources have to be avoided. The former is called deadlock and the latter is called starvation. Both of them are important problems in development of distributed systems. Many distributed algorithms for avoiding them have been developed. Some examples of them are the methods which employ techniques such as time stamps [9][10][6], acyclic dericted graph [1], and coterie [6], etc.

On the other hand, with the progress of computer networks, many cooperative activities of groups happen in a network environment. If

these groups' activities are performed around the same period of time, these groups may compete for resources of the network. In such a case, the deadlock among groups and/or starvation of a group may happen.

For example, there are 4 groups g_a, g_b, g_c, and g_d consisting of $\{p_1, p_2, p_3\}$, $\{p_4, p_5\}$, $\{p_6, p_7, p_9\}$, and $\{p_7, p_8, p_9\}$, respectively as shown in Fig. 1. A process may be an agent, a human, an object, or a UNIX process depending on different applications. The members of a group may be resident at different sites far from each other.

Processes in different groups may be geographically closed to each other and compete for resources. In the example, p_1 and p_6 compete for r_1, p_2 and p_4 compete for r_2, p_3 and p_8 compete for r_3, p_5 and p_7 compete for r_4, and p_5 and p_9 compete for r_5.

To make the story more realistic, suppose every group wants to hold a group meeting around the same time, they compete for meeting rooms equipped with high quality presentation tools such as shared white board, screens and cameras for eye contact, and so on. Since the meeting rooms are limited, allocation of rooms to groups is important. Careless allocation, such as r_1 to p_6, r_3 to p_8, r_2 to p_2, and r_4 and r_5 to p_5, may lead to deadlock among groups, even some process may acquire the access privilege to some room, no group meeting can be held, since other members of the group can't acquire meeting rooms.

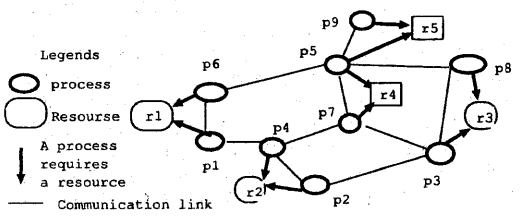

Figure 1:

Notice there is processes p_7 and p_9 belonging to both groups g_c and g_d. However, each of p_7 and p_9 can participate in only one group's meeting at a time. We have to consider such a situation, to arrange meeting time satisfying the condition, while we allocate the meeting rooms to groups. Careless arrangment may also lead to a deadlock among groups. For example, if a room is allocated to each of p_6, p_7, p_8, and p_9 but arranging p_9 to attend the meeting of g_c, and p_7 to attend the meeting of g_d, both the meetings can't be held.

Previous resource allocation model and its variations and extensions couldn't reflect the relations among groups explicitly, and didn't give the definition of deadlock among groups and starvation of a group. Moreover, previous algorithms could only deal with the deadlock among processes and starvation of a process, but not deadlock among groups and starvation of a group. Our previous works [13][14] didn't consider the case where a processes may belong to more than one groups and participate in only one group's activity at a time.

The problem presented in the paper can be described as an extension of dining philosophers problem, called dining philosophers party problem. Suppose in a dining room, there are some tables, each of which is surrounded by some chairs. For each table, when each of the chairs surronding the table is taken by a philosopher and each philosopher holds a glass of wine, all philosophers surrounding the table toast (drink) to the health of each other using the glass of wine, and then begin to eat. If there is a philosopher who has not acquired a glass of wine, other philosophers surrounding the table will wait until the philosopher acquires that. A glass is put on a shelf between two neighboring tables. So a philosopher may compete for the glass with another philosopher sitting by a neighboring table.

In order to solve the new problem, we firstly represent clearly the relation among groups of processes, and then define the deadlock among groups and starvation of a group explicitly. In addition, we propose a solution which guarantees that deadlock among groups and starvation of a group never happen, based on extension of acyclic graph approach.

Recently, mutual exclusion method considering the relations of inter-groups and intra-group attracts some researchers' attention, and an algorithm for resolving the resource competition by using K logical ring was proposed [11]. However, in the model, for every group, a resource is competed for by all processes in the group, which can't be applied to the situation where every process in a group has to be ensured to acquire its required resources in the same time, and do their cooperative group work together. So far, there is no such model as ours, which deals with that a set of processes in a group compete for resources with other groups, and acquire all required resources for them to do their cooperative group work.

The rest of the paper is organized as follows. In Section 2, our resource allocation model and the definition of DARG are given. In Section 3, we propose our new solution to DARG. Section 4 concludes the paper.

2 Model and Problem

Definition 1 Resource allocation model

The resource allocation model is represented by a tuple (B, F). $B = (V, E)$ is a bipartite graph, where $V = \mathbf{P} \cup \mathbf{R}$ is a set of nodes consisting of a set of processes denoted with $\mathbf{P} = \{p_1, p_2, \cdots, p_i, \cdots p_n\}$ and a set of resources denoted with $\mathbf{R} = \{r_1, r_2, \cdots, r_j, \cdots, r_m\}$. E is a set of edges $e = (p_i, r_j)$ between p_i and r_j. F is a family of subsets of processes which belong to $\mathbf{P}$. That is $F = \{g_1, g_2, \cdots, g_k, \cdots, g_h\}$, where $g_k \subseteq \mathbf{P}$. g_k is called "process group" or "group".

If a process may require a resource, there is an edge between them in B. We also say the process is adjacent to the resource. Each process is adjacent to one or more resources. The set of resources, to which a process p_i is adjacent, is denoted with $R_i \subseteq \mathbf{R}$. The set of processes, which requires a resource r_j, is denoted with $P_j \subseteq \mathbf{P}$. If $R_{i1} \cap R_{i2} \neq \{\}$, two processes p_{i1} and p_{i2} are said to be in competition. In other words, there is a resource r_j, such that $p_{i1} \in P_j$ and $p_{i2} \in P_j$. The set of processes, the set of resources, the adjacent relation between processes and resources, and the set of groups don't change dynamically during an execution of our algorithm.

The following properties have to be guaranteed when allocating resources to processes of groups.

(1) Each process has to acquire all resources adjacent to it.

(2) A resource can be allocated to only one process at a time.

(3) In order for a group to perform their group cooperative work, the group has to acquire all resources adjacent to the group (all resources adjacent to any process in the group).

(4) Two or more groups can't work in the same time, if they share a common process, (i.e. the process belongs to these groups).

Remark 1

The conditions (1) and (3) could be relaxed to a more general model, since in some applications, that a part of processes acquire a part of required resources is enough for a group to perform its group work.

Definition 2 Deadlock, Starvation, Group Deadlock, and Group Starvation

Deadlock means that no process can acquire all resources adjacent to it.

Group deadlock (deadlock among groups) means that there is no group such that every process of the group can acquire all resources adjacent to it.

Starvation means that there is some process that never acquire all resources adjacent to it in the same time.

Group starvation (starvation of a group) means that there is some group such that some process of the group never acquire all resources adjacent to it in the same time.

Assumption 1 We assume that processes belonging to the same group do not compete for any resource. That is because if they compete for a resource, not all processes in the group can acquire their required resources at the same time. That means it is impossible to solve the problem with group starvation free.

However, the assumption could be removed under the relaxed condition mentioned in **remark 1**.

Definition 3 Distributed problem DARG:

The problem DARG (Distributed Allocation of Resources to process Groups) is how to devise an algorithm which allocates the resources to the processes in the above model without group deadlock, group starvation, deadlock, and starvation under the above assumption.

Initially, each process knows its own identifier, the identifiers of groups to which the process belongs, identifiers of all processes in these groups, identifiers of the processes which compete for some resource(s) with it. We assume that a total order among group identifiers denoted with integers is predefined. Similarly, another total order among process identifiers is predefined also.

As output, each process in a group acquires all required resources and begins to work by using these resources, with Deadlock, Starvation. Group Deadlock and Group Starvation never happening.

Assumption 2 Underlying network

In this paper, we consider that the resource allocation model will be implemented in a network environment. A process is mapped to a site of the network. Each pair of processes in the same group are connected with a communication channel. In addition to these channels, there is a communication channel between each pair of processes which are in competition.

Each process executes the same algorithm, which consists of sending massages on incident edges, waiting for incoming messages, and processing them. Messages are transmitted independently in both directions in a communication channel, and arrive after a finite but unpredictable delay, without errors and in FIFO order.

Each process has 3 state: *thinking*, *hungry*, and *acquired* similar to the 3 states of a philosopher in dining philosophers problem. In *thinking* state, the process does not require any resource. A process spontaneously makes transition from *thinking* to *hungry* state in finite time. "Spontaneously" means that the transition is not dependent on other processes, but the process itself. Generally, some input from a user or an application of the process initiates the transition. In *hungry* state, the process requires all resources adjacent to it. When all resources required by a process are acquired, its state is changed to

acquired. Though a process in *acquired* state can access the resources, in order to use these resources for group work, the process may have to wait for other processes in the group, due to the properties (3) and (4) in **definition 1**.

3 An Outline of Our Solution

The traditional distributed resource allocation without considering competition among groups can be described as the distributed dining philosophers problem, illustrated by a connected undirected graph, in which a vertex (a process) represents a philosopher, and an edge represents a fork between the pair of processes that compete for a set of resources.

A famous solution for distributed dining philosophers problem is based on an acyclic directed graph technique [1][2]. The technique points of the approach are summarized below. We assume that readers are familiar with the traditional dining philosopher problem. We use the terms such, thinking, hungry, eating, fork, etc. as in the problem without explanation.

(1) An arc (p_i, p_j) oriented away from p_i to p_j means that p_j has the privilege to use the fork competed by the two processes.

(2) Initially, the direction of every arc is arranged such that the directed graph formed by the arcs are acyclic, so that there is at least a sink vertex (philosopher), which holds all privileges of forks incident to it and is able to acquire all required forks in order to eat.

(3) After eating, the philosopher reverses all incident arcs simultaneously. So that the directed graph is still acyclic and some other vertex becomes sink.

For example, 3 processes compete for 3 forks as shown in Figure 2 (a). Initially, p_1 is a sink, and eventually can acquire all required forks and use them. Then it reverses the arcs, see Figure 2 (b), so that p_2 can acquire all required forks. The same will happen to p_3.

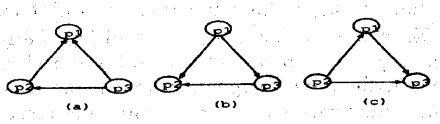

Figure 2:

We extended the traditional resource allocation problem to the distributed resource allocation among process groups, in which all processes of a group have to acquire all required resources in order to perform their cooperative works by using the resources, assuming there is no more than one group shares a common process [13][14].

A solution for (DRAG) was also based on the acyclic graph approach. However, the above points is not enough to solve the DRAG. For example, in Figure 3 (a), the graph is acyclic, but still a deadlock among groups happens, since there is no such a group that all processes in the group can acquire all required resources.

One reason of the group deadlock is because the arcs between g_a and g_b is inconsistent. Namely, arc from p_3 to p_1 is oriented away from g_b to g_a, but arc from p_2 to p_4 is in the opposite direction. Therefore, g_a and g_b may never acquire all resources required. Even though the direction of the arc from p_3 to p_1 is reversed to remove the inconsistency, as shown in Figure 3 (b), still group deadlock exists, since no group of g_a, g_b, and g_c can acquire all resources adjacent to the group. This is because for every group, some arc of a process belonging to the group is directed to a process of another group, which means the group doesn't hold all privileges. If reversing the arc from p_2 to p_4 in Figure 3 (a), the deadlock is avoided, because all processes p_1 and p_2 can acquired all required resources, (see Figure 3 (c).)

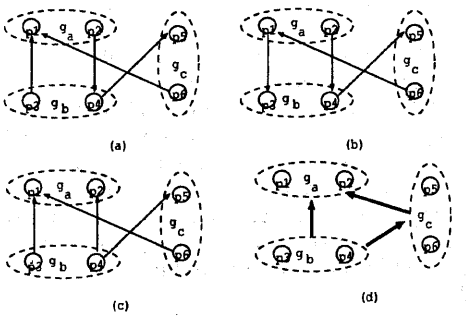

Figure 3:

Therefore in additional to above points, the following are required.

(4) All arcs between a pair of groups have to direct to the same direction, i.e. from one group to another group.

Taking all arcs between a pair of groups as an meta-arc and a group as a meta-vertex, we obtain another meta-graph. (See Figure 3 (d).)

(5) The meta-graph should be kept to be acyclic to guarantee group deadlock and group starvation.

Comparing with our previous papers[13][14], we consider *DARG* in a more general model, in which a process may belong to more than one group, in this paper. The above (4) and (5) don't work in such a case. For example, in Figure 4,

all arcs are directed from group g_d to g_a, however, it is not true for g_b and g_c. Whatever, you change the initial direction of the arcs, the above (4) can't be satisfied by all groups. The above

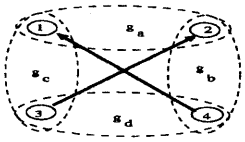

Figure 4:

example tells us that we need more tricky and elegant control method for the problem.

Our basic ideas for solving the new problem are as follows.

(1) Let $G_i \subseteq F$ be a set of groups, to each of which p_i belongs, and $|G_i| = q$.

Every process p_i belonging to $q > 1$ groups holds a $q \times q$ array A_q. An element $x_{ig_k}^{ig_l}$ $(g_k \neq g_l)$ of A_q is a boolean variable, where $g_k, g_l, \in G_i$.

$x_{ig_k}^{ig_l} = 1$ means that p_i would like to participate in the group work of g_k but not g_l. In other words, $x_{ig_l}^{ig_k} = \neg x_{ig_k}^{ig_l}$. For the case of $g_k = g_l$, $x_{ig_k}^{ig_l} = null$.

Remark 2 It seems to be redundant that i is used in both the subscript and superscript of $x_{ig_l}^{ig_k}$. However, it is for specifying variables in an uniform manner, which is convenient to specify the initial conditions of the solution. (See the latter part of the section.)

If p_i participates in activity of group g_k, then $\forall g_l \in G_i, x_{ig_k}^{ig_l}$ must be equal to 1. After p_i finishes g_k's activity, $x_{ig_k}^{ig_l}$ is set to 0 and $x_{ig_l}^{ig_k}$ is set to 1.

(2) In the solution for the dining philosophers problem [1][2], an arc between a pair of processes (p_i, p_j) could be implemented by a pair of boolean variables x_i^j and x_j^i held by p_i and p_j, respectively, under the condition $x_i^j = \neg x_j^i$.

In this paper, if p_i and p_j belongs to q and r groups respectively, p_i holds a $q \times r$ array A_{qr} and p_j holds a $r \times q$ array A_{rq} of variables, for the edge (fork) between p_i and p_j. Element $x_{ig_k}^{jg_l}$ in A_{qr} is equal to $\neg x_{jg_l}^{ig_k}$ in A_{qr}.

The following three initial conditions have to be satisfied.

$\forall i, j \in P, g_k, g_l \in F \ s.t. \ g_k \neq g_l,$

$x_{ig_k}^{jg_l} \wedge \neg x_{jg_l}^{ig_k} \vee \neg x_{ig_k}^{jg_l} \wedge x_{jg_l}^{ig_k} = 1 \qquad (1)$

The condition is for the mutual exclusion property. That is, it guarantees no two processes use the same fork in the same time and no two groups sharing a common process perform their group works in the same time.

$\exists g_k, g_l \in G, \ (\bigwedge_{\forall i \in g_k, \ j \in g_l, \ s.t. \ x_{ig_k}^{jg_l} \ exists} x_{ig_k}^{jg_l}) \vee$

$(\bigwedge_{\forall i \in g_k, \ j \in g_l, \ s.t. \ x_{ig_k}^{jg_l} \ exists} \neg x_{ig_k}^{jg_l}) = 1 \qquad (2)$

The condition is for the mutual exclusion between two groups. All privileges should be held by one group to guarantee group deadlock-freedom.

If processes $p_i \in g_k$ and $p_j \in g_l$ compete for a resource, respectively, we say group g_k and g_l are adjacent. A path among groups is defined as a sequence of distinct groups $g_1, g_2, \cdots, g_s$, such that g_{h+1} is adjacent to g_h, where $1 \leq h \leq s$. A cycle of groups is defined as a closed path such that all groups are distinct except $g_1 = g_s$.

For any cycle of groups, $g_1, g_2, \cdots, g_s$,

$(\bigwedge_{\forall i_1, j_1, l_1} x_{i_1 g_1}^{j_1 l_1})$

$\bigvee (\bigwedge_{\forall i_2, j_2, l_2} x_{i_2 g_2}^{j_2 l_2})$

$\bigvee \cdots$

$\bigvee (\bigwedge_{\forall i_{s-1}, j_{s-1}, l_{s-1}} x_{i_{s-1} g_{s-1}}^{j_{s-1} l_{s-1}}) = 1 \qquad (3)$

The condition is for group deadlock-free also. Intuitively, it means for any cycle of groups at least there exists one group, all variables of which are equal to 1, i.e. the group can acquire all resources it required.

Such a group will use the resources. After that, it does not release the resources at once, but only on request of other processes as in the solution for the distributed dining philosophers problem [2]. If other adjacent groups don't require the resources, i.e. the processes of these groups are in *thinking* state, the former group may use again the resources if it makes transition from *thinking* to *hangry* again. However, to guarantee group starvation-free, the privileges of using the resources should be sent to adjacent processes.

To this end, variable $x_{ig_k}^{jg_l}$ is set to 0, and a $turn(g_k)$ message is sent to $p_j \in g_l$. On receiving the message, p_j sets its variable $x_{jg_l}^{ig_k} = 1$. During the transmission of the message, variables $x_{ig_k}^{jg_l}$ and $x_{jg_l}^{ig_k}$ may be both equal to 0, but they will never be 1 in the same time, which guarantees the mutual exclusive access to resources.

A group of processes will begin to use the resources, when all $x_{ig_k}^{jg_l} = 1$ and all its processes acquire all their required resources. To check if the condition is satisfied, communication among processes in a group is needed. Message $inform$ is employed for the purpose.

Assume that every group has an unique identifier represented by an integer, a simple method could be used to set the initial values of the variables such that all the above 3 conditions are satisfied.

For the pair of variables $x_{ig_k}^{ig_l}$ and $x_{ig_l}^{ig_k}$, both of which are held by process p_i, p_i sets $x_{ig_k}^{ig_l} = 1$

and $x_{ig_l}^{ig_k} = 0$, if $id(g_k) < id(g_l)$.

For the pair of variables $x_{ig_k}^{jg_l}$ and $x_{jg_l}^{ig_k}$ held by different processes p_i and p_j belonging to different groups, p_i and p_j send their group identifiers to each other, and then the one holds smaller group identifier sets its corresponding variables to 1, another one sets its corresponding variables to 0.

A *fork* competed for by p_i and p_j can be initially held by any one of p_i and p_j. We use a variable $y_i^j = 1$ to represent p_i holds the fork. Obviously, $y_j^i = \neg y_i^j$.

4 Conclusion

In this paper, we have presented a generalization of the resource allocation problem and a solution to the generalized problem.

Recently, opportunities for groups of people or agents to work together on a computer network are increasing. The demand for applications that support such works are growing now and in the near future. The problem of distributed resource allocation among process groups can be considered as a building block for such applications.

Comparing with our previous work, we removed the restriction that no more than one group may share a common process, so that the new solution will have more possible applications.

Some future works are summarized as follows:

1. For a given set of processes, how to partition the processes into groups is an interesting problem and needs to be further studied.
2. The time complexity of our solution needs to be analyzed.

References

[1] K. M. Chandy and J. Misra, "The Drinking Philosophers Problem," ACM Trans. Prog. Lang. Syst. , Vol. 6, 4, pp. 632-646, 1984.

[2] V. C. Barbosa, "An Introduction to Distributed Algorithms," The MIT Press, 1996.

[3] Nancy A. Lynch, "Distributed Algorithms," Morgan Kaufmann Publishers, 1996.

[4] Injong Rhee, "A fast Distributed Modular Algorithm for Resource Allocation," Proceedings of the 15th International Conference on Distributed Computing Systems, pp. 161-168, May, 1995.

[5] D. Ginat, A. U. Shankar and A. K. Agrawala, "An Efficient Solution to the Drinking Philosophers Problem and its Extensions," Proceeding of the 3rd International Workshop on Distributed Algorithms, Lecture Notes in Computer Science, Vol. 392, pp. 83-93. 1989.

[6] H. Kakugawa and M. Yamashita, "Local Coteries and a Distributed Resource Allocation Algorithm," Transactions of Information Processing Society of Japan, Vol. 37, 8, pp. 1487-1496, Aug. 1996.

[7] L. Lamport, "Time, Clock , and Ordering of Events in a distributed System," Communications of the ACM, Vol. 21, No. 7, pp. 558-565, July 1978.

[8] M. Maekawa, "A $\sqrt{N}$ algorithm for mutual exclusion in decentralized systems," ACM Trans. Comput. Syst., Vol. 3, No. 2, pp. 145-159, May. 1985.

[9] M. Raynal, "A Distributed Solution to the k-out of-M Resources Allocation Problem," LNCS 497, Springer-Verlag, pp.599-609.

[10] R. Baldoni, "An $O(N^{M/M+1})$ distributed algorithm for the k-out of-M resources allocation problem," Proc. of 14th ICDCS, pp.81-88, 1994.

[11] A. KHIAT and M. NAÏMI, "Distributed Mutual Exclusion In K-Groups Based on K-Logical Rings," Proceedings of the 11th Annual International Symposium on High Performance Computing Systems, HPCS'97, pp.269-282, Jul. 1997.

[12] Judit Bar-Ilan and David Peleg, "Distributed Resource Allocation Algorithms," LNCS 647, pp.277-291, Nov. 1992.

[13] Zixue Cheng, Tongjun Huang, and Norio Shiratori, "A Distributed Algorithm for Resource Allocation among Process Group," Proceedings of The 9th International Conference on Information Networking, pp. 443-448, Dec. 1994.

[14] Y. Wada, Z. Cheng, and T. Huang, "A distributed Algorithm for Allocation of Resources to Process Groups with Acyclic Graphs," Proc. of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'97), pp.801-805, Jul. 1997