International Journal of Parallel Programming, Vol. 15, No. 4, 1986

An Optimal Distributed Solution to the Dining Philosophers Problem

S. P. Rana[1] and D . K. Banerji[2]

Received September 1986; Accepted February 1987

An optimal distributed solution to the dining philosophers problem is presented. The solution is optimal in the sense that it incurs the least communication and computational overhead, and allows the maximum achievable concurrency. The worst case upper bound for concurrency is shown to be n div 3, n being the number of philosophers. There is no previous algorithm known to achieve this bound.

KEY WORDS: Dining philosophers; distributed algorithms; concurrency; performance analysis; resource allocation.

1. INTRODUCTION

In this paper, we consider the well-known dining philosophers problem[1] extended for a distributed environment. The philosophers are seated around a table. In front of each philosopher is placed a fork and a plate of spaghetti. The philosophers engage themselves in thinking until they get hungry. A hungry philosopher, in order to eat, requires two forks, the fork in front and the fork in front of the neighbor to the right. After acquiring the forks, a philosopher eats the spaghetti, releases both the forks and starts thinking all over again. Two neighboring philosophers compete for the common fork. The problem is to devise an allocation strategy for the forks that is distributed, deadlock-free and starvation-free.

The underlying model for this problem is formally described in Section 2. In addition to the correctness of a distributed protocol, we are

[1] Department of Computer Science, Wayne State University, Detroit, MI 48202.
[2] Department of Computing and Information Science, University of Guelph, Guelph, Ontario N1G 2W1.

0885-7458/86/0800-0327$05.00/0

further interested in its performance in terms of computational and communication overhead and its potential for concurrency. In this case, concurrency is a measure of how many philosophers are able to eat simultaneously out of a set of hungry philosophers, in all reachable configurations.

We show in Section 3 that every protocol for the dining philosophers problem can reach a configuration in which at most n div 3 philosophers can eat.

Subsequently, in Section 4, we present a protocol that achieves this bound of n div 3 as well as incurs least computational and communication overhead. An earlier distributed solution[2] is also known to have least computational and communication complexity. However, in that solution, it is possible for the system to reach a configuration where at most n div 4 philosophers are able to eat.

2. MODEL

Let $P_1, P_2, ..., P_n$ be the set of philosophers such that $P_{(i+1) \bmod n}$ is the right neighbor of P_{ij}; $1 \leqslant i \leqslant n$. In a distributed environment, the philosophers are located at logically unique sites. The fork in the front of P_i is denoted by f_i and P_i and f_i are on the same logical site, say, S_i; $i \leqslant i \leqslant n$.

At any time, a philosopher is in exactly one of the three possible states: thinking, hungry or eating. A philosopher transits from thinking to hungry state arbitrarily and spontaneously but may remain in thinking state forever. Once in hungry state, a philosopher P_i attempts to acquire both forks f_i and $f_{(i+1) \bmod n}$. Upon getting the forks, the philosopher transits to eating state. It is assumed that a philosopher in eating state eventually comes out of this state, releases both the forks and goes to thinking state again. Thus, a philosopher cannot remain in eating state forever and, further, a fork cannot be held by or on behalf of a philosopher in thinking state.

There is no centralized controller for the allocation of forks. Also, nothing is assumed about the relative speeds of the philosophers. Each fork f_i has an associated arbiter A_i, also located at the site S_i. The arbiter A_i controls the allocation of fork f_i. In addition to controlling f_i, arbiter A_i also monitors the state of philosopher P_i. When P_i enters the hungry state, it is the arbiter A_i that acquires both the forks f_i and $f_{(i+1) \bmod n}$ on behalf of P_i and subsequently causes P_i to change its state from hungry to eating. When P_i finishes eating, the arbiter A_i marks f_i free and returns the fork $f_{(i+1) \bmod n}$ to the corresponding neighbor arbiter.

A correct protocol for arbiters, A_i, $i = 1 \cdots n$, ensures that when P_i becomes hungry, the arbiter A_i eventually succeeds in acquiring both the

forks and, consequently, P_i is able to transit from hungry to eating state. Since the arbiters are on logically distinct sites, the neighboring arbiters have to communicate for resolving the conflicts for the allocation of forks. We assume an asynchronous message based and perfectly reliable communication medium.

To facilitate the presentation of system configurations, we employ the following notations. We represent a site S_i by a labelled node in a graph. The label of a node is a triple enclosed in square brackets with first component as the index of the site. The remaining two components in the triple capture the information about the status of forks at the site in question. The second component lists the forks held at the site by, or on behalf of, the philosopher associated with the site. The last component in the triple lists the fork controlled at the site, if it is free. Note that any or both of the last two components in a label may be empty. To illustrate this notation, the label $[3; f_4;]$ conveys the information that it is associated with site S_3 and the fork f_4 is held for the philosopher P_4. Further, the empty third component indicates that the fork f_3 is not free and f_3 must be at site S_2, since it is not held at site S_3.

Using this notation, a global state, where all philosophers are in thinking state, is depicted as follows:

$$\begin{array}{cccccc} [\mathring{1}; & [\mathring{2}; & \cdots & [\mathring{n}-1; & [\mathring{n}; \\ ; & ; & & ; & ; \\ f_1] & f_2] & & f_{n-1}] & f_n] \end{array}$$

We now introduce some other definitions to be used in later sections. Suppose that an arbiter A_i holds a fork, on behalf of P_i of course, and has requested another fork from arbiter $A_{(i+1) \bmod n}$, whose fork is currently not free. Such a situation is denoted by us graphically by a directed arc from node i to node $(i+1) \bmod n$. We call this directed arc as a "hold-and-wait link." This link disappears as soon as the requested fork is free.

Consider a collection of nodes $\{i_1, i_2, ..., i_k\}$; $k \leqslant n$. These nodes are said to be on a "hold-and-wait chain" at a time instant if and only if for all j; $i \leqslant j < k$; there is a "hold-and-wait link" from node i_j to node i_{j+1}.

The nodes form a "hold-and-wait cycle" at a time instant if all nodes in the collection $\{1, 2, ..., n\}$ are on a "hold-and-wait chain" and there is a "hold-and-wait link" from node n to node 1. Obviously, the existence of a "hold-and-wait cycle" implies a deadlocked system state.

3. WORST CASE UPPER BOUND ON CONCURRENCY

The upper bound on the number of philosophers who could eat simultaneously is obviously n div2. However, no solution is known to

exhibit such a behavior in all reachable configurations where all philosophers want to eat.[2–4] In this section, we establish the worst case upper bound on concurrency.

Theorem 1. Every protocol for the dining philosophers problem can reach a configuration in which at most n div 3 philosophers eat simultaneously. In other words, n div 3 is the worst case upper bound on concurrency.

Proof. Suppose P is a protocol for the dining philosophers problem in which in all configurations at least n div 3 philosophers are able to eat simultaneously when all philosophers are wanting to eat.

Consider the execution of P in which every third philosopher in turn becomes hungry and picks up both forks. Pictorially, this situation is depicted as follows:

[$\overset{\circ}{1}$;	[$\overset{\circ}{2}$;	[$\overset{\circ}{3}$;	[$\overset{\circ}{4}$;	[$\overset{\circ}{5}$;	[$\overset{\circ}{6}$;
;	;	f_3, f_4;	;	f_5;	f_6, f_7;
]	f_2]	]	]	]	]

From this configuration, no other philosopher is able to eat until one of the n div 3 eating philosophers transits from eating phase and thereby returns a fork. To complete the proof, we show that such a configuration may be reached irrespective of the allocation strategy in P.

For example, consider the scenario where, except every third philosopher in turn, all other philosophers remain in thinking state during the interval when the previous hungry philosophers are in the process of acquiring forks. Since a fork can never be held for a thinking philosopher in our model, in any correct protocol P, each of the previous hungry philosophers will eventually succeed in picking up both the forks because of no competition from the neighbors.

4. AN OPTIMAL DISTRIBUTED SOLUTION

We first describe the common part of all protocols discussed in this section and also state the common assumptions under which these protocols operate. The presented protocols differ only in one aspect viz. the order in which the forks are requested by the arbiters.

(i) Each arbiter meets the fork requests in order of their arrival. If at the time of a request, the fork is not available, it is reserved and is made available to the requestor as soon as it is released.

(ii) An arbiter acquires the forks, one as a time, on behalf of the associated philosopher. That is, unless the first requested fork is

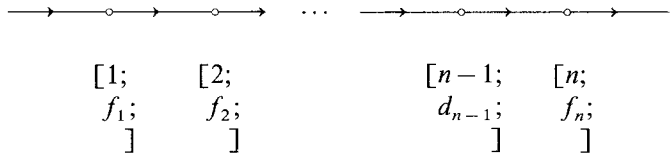
not acquired, the arbiter does not acquire, reserve or send a request for the other fork. An arbiter does not have to communicate for acquiring the fork controlled by it. It simply reserves or acquires the fork by marking the fork reserved or busy, as the case may be.

(iii) When a philosopher goes from eating to thinking state, the corresponding arbiter immediately marks its fork free. If the fork was marked reserved, it is marked busy and granted to the requestor. Further, the second fork is immediately returned to the neighboring arbiter to the right. The fork is returned by an intersite message and it is assumed that this return fork message is eventually sent and received, thereby making the fork eventually available to the receiving site.

(iv) Further, it is assumed that if an arbiter is expected to send a request to the arbiter to its right, it does so in a finite time and its request is eventually received. Similarly, when a fork is granted to a left neighbor, the message to that effect is eventually received by the latter.

Thus, in the class of protocols previously described, three messages are exchanged between arbiters per eating phase of a philosopher. These messages are request fork from right neighbor, grant fork to left neighbor and return fork to right neighbor. Note that these are the minimum number of messages required by any protocol. Further, the computational overhead is least, since the allocation of a fork requires checking a fork and marking it busy or reserved.

We now discuss the protocols further. Consider the strategy in which every arbiter A_i, on P_i's becoming hungry; $1 \leqslant i \leqslant n$, grabs fork f_i first. It is easy to see that such a strategy may lead to the following configuration:



This configuration has a "hold-and-wait cycle" and thus, the previous strategy does not give a deadlock-free solution. However, it can easily be modified to give a deadlock-free solution by using the well-known deadlock prevention scheme of ordering resources and restricting the generation of resource requests in the prespecified order. In the present context, the resources (forks) are already indexed. Furthermore, all arbiters except A_n

do request forks in order of increasing fork indexes in this strategy. By modifying the protocol for A_n such that A_n requests fork f_1 first, we get our first deadlock-free solution.

Protocol 1. All arbiters, on behalf of the associated philosophers, acquire forks in order of increasing fork indexes.

Protocol 1 exhibits a bad concurrency behavior because of the potential of forming long "hold-and-wait chains." In the worst case, as shown here, as much as $(n-1)$ nodes may be on a "hold-and-wait chain":

$[n;$ $[1;$ $[2;$ $[n-1;$
$;$ $f_1;$ $f_2;$ $f_{n-1}, f_n;$
$]$ $]$ $]$ $]$

In order to improve the concurrency behavior, we modify Protocol 1 further. To simplify the ensuing presentation, we use the following definition.

An arbiter A_i is called to be of "O-type" if it grabs fork f_i and then requests for the other fork; otherwise, it is called to be of "R-type." The letter "O" and "R" are acronyms from Own-fork-first and Right-fork-first, respectively. Accordingly, Protocol 1 may be restated as follows:

"The arbiter A_n is of R-type while all others are of O-type."

We now prove two lemmas:

Lemma 1. If the right neighbor of an O-type arbiter is of R-type, then a hold-and-wait link between them can exist if and only if the associated philosopher of R-type arbiter is in eating state.

Proof. Let us assume that philosopher of R-type arbiter is not in eating state. In such a case, the R-type arbiter cannot hold its forks, because holding it implies it has both the forks and the associated philosopher is in eating state. Since R-type arbiters fork is free, the hold-and-wait link between O-type and R-type cannot exist in this case. The proof is complete.

We say that this pair of arbiters form an ··OR·· pattern.

Lemma 2. In a pattern ··OR·· of arbiters, i.e., an R-type arbiter surrounded by O-type arbiters, a hold-and-wait chain cannot cross the node of R-type arbiter.

Proof. By Lemma 1, if there is a hold-and-wait link between the first two nodes, the philosopher of R-type must be eating and consequently there will not be any hold-and-wait link between the last two nodes at that time. This implies that there can be at most one hold-and-wait link either between the first two or between the last two nodes in the ··ORO·· pattern. The proof is complete.

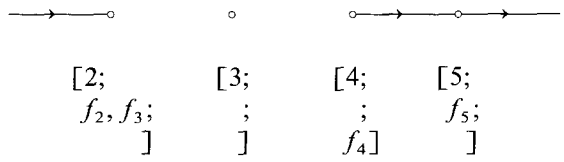
The deadlock-freedom of Protocol 1 is a direct consequence of Lemma 2, since arbiters A_{n-1}, A_n and A_1 form the ··ORO·· pattern. To improve the concurrency behavior of Protocol 1, we use the insight from Lemma 2, not only to prevent the formation of a hold-and-wait cycle but also to prevent the formation of long hold-and-wait chains. The trick is to repeat the pattern ··ORO·· more often by introducing more R-type arbiters.

We have a second solution now.

Protocol 2. An Arbiter A_i is of R-type if i is a multiple of 2; otherwise, it is of O-type.

Protocol 2 arranges the arbiters in the pattern ORORORO···. Protocol 2 is essentially the same as given by Cargill.[2] Intuitively, it appears that Protocol 2 must exhibit the best concurrency behavior, since hold-and-wait situation does not propogate beyond three arbiters. Unfortunately, that is not true, and as pointed in Ref. 2, in Protocol 2, one may reach a configuration where at most n div 4 philosophers can eat simultaneously.

Such a configuration results from the scenario shown as follows for the remaining quadruples:



In this scenario, out of a sequence of four hungry philosophers, only the first one is able to eat while all others are blocked. Observe that the node with index 4 has a free fork but no arbiter can pick it up in this scenario. This is so because fork f_4 is their second fork, i.e., it cannot be acquired unless they have acquired their first fork. Thus, in the worst scenario, n div 4 forks remain free in the system even though all philosophers want to eat . This is perhaps the reason for less than optimal concurrency behavior of Protocol 2.

We give a final protocol that is conceptually similar to Protocol 2 but exhibits an optimal concurrency behavior.

Protocol 3. (Optimal solution) An arbiter A_i is of R-type if i is a multiple of 3; otherwise, it is of O-type.

In Protocol 3, n div 3 arbiters are of R-type and are scattered among O-types as in the pattern OOROOROOR···. We now prove the correctness and performance results for Protocol 3.

Result 1. Protocol 1 is starvation- and deadlock-free.

Proof. Since there are ··ORO·· patterns of arbiters, as dictated by Protocol 3, deadlock freedom is proved by application of Lemma 2. Further, as described in the beginning of this section, each request for a fork succeeds in either getting the fork or reserving the fork. Since a fork is eventually released, the reserved fork is eventually granted to the requesting arbiter. Thus, starvation can not occur in any of the protocols presented in this section.

Lemma 3. In a pattern OOR of three consecutive arbiters, in every reachable configuration when all philosophers want to eat, at least one of these three arbiters will be able to acquire both the forks and thereby will cause the corresponding philosopher to transit to eating state.

Proof. Consider the pattern OOR.

If the first O-type has both the forks, the proof is complete.

If the first O-type arbiter is waiting for a fork, then there are two cases

(i) waiting for the fork in front.

(ii) waiting for the fork in front of neighbor to the right, in which case the fork in front must have already been acquired.

Consider case (i). Here, the second O-type can always have the fork in his front and compete for the second fork. If R-type arbiter is not competing, the second O-type will get the second fork. However, if R-type is also competing for the same fork, then both second O-type and R-type are competing for their second fork. Whosoever succeeds in the competition will have both the forks. Thus, in case (i), either the second or the third arbiter is able to acquire both the forks in all reachable configurations.

Now, let us consider case (ii). If the second O-type is not competing, the first O-type will get the second fork. However, if both the first and second O-type are competing for the same fork, then either of them may succeed. If the second O-type succeeds in getting the fork, this would be his first fork. As in case (i), second O-type and the R-type now or may not compete for the same fork. In either case, as shown earlier, exactly one of them is able to acquire both its requisite forks.

This complete the proof of Lemma 3.

Result 2. In protocol 3, in every reachable configuration when all philosophers want to eat, at least n div 3 philosophers will be able to eat simultaneously.

Proof. Since in protocol 3, the pattern OOR is repeated n div 3 times, by using Lemma 3 the proof is complete.

5. CONCLUSION

We have presented a new deadlock- and starvation-free distributed solution to the dining philosophers problem in a distributed environment. The solution has optimal concurrency behavior in the sense that maximum possible number of philosophers are able to eat concurrently in all configurations when all philosophers are wanting to eat. Further, the solution is also robust like the solution in Ref. 2 because the effect of a failure is not propagated beyond the immediate neighbors.

ACKNOWLEDGMENTS

We are thankful to an anonymous referee for providing valuable suggestions for improving the presentation of the results in this paper. This work has been supported in part by The Natural Sciences and Engineering Research Council of Canada under grant numbers A0087 and A0956 and also from a grant from The Institute of Manufacturing Research at Wayne State University.

REFERENCES

1. E. W. Dijkstra, Hierarchical ordering of Sequential Processes, *Acta Informatica* **1**(2):115–138 (1971).
2. T. A. Cargill, A Robust Distributed Solution to the Dining Philosophers Problem, *Software Practice and Experience*, **12**(10):965–969 (1982).
3. E. Chang, *n*-Philosophers: An Exercise in Distributed Control, *Computer Networks* **4**:71–76 (1980).
4. H. Wedde, A Starvation-Free Solution for the Dining Philosophers Problem by use of Interaction Systems, *Proc. MECS '81 Symposium*, Lecture Notes in Computer Science 118, Springer, Berlin, pp. 534–543 (1981).