A Modification to the Chandy-Misra Dining Philosophers Algorithm to Support Dynamic Resource Conflict Graphs

Kenneth Goldman and Joe Hoffert  
{kjg,joeh}@cs.wustl.edu  
Distributed Programing Environments Group  
Department of Computer Science,  
Washington University, St. Louis, MO. 63130, U.S.A.

Abstract

The Chandy-Misra dining philosophers algorithm is based on a static undirected resource conflict graph $H$ in which each vertex represents a process and each edge represents potential resource conflict. The algorithm creates a distributed priority graph by implicitly assigning a direction to each edge in the static graph based on priority of the incident processes for using a resource. The priorities change over time to prevent starvation and the algorithm maintains acyclicity of the priority graph to prevent deadlock. In this paper, we present some simple modifications to the algorithm that allow the potential resource conflict graph $H$ to vary over time, with the addition and deletion of both processes (vertices) and potential conflicts (edges) dynamically. Our algorithm guarantees that no waiting process will be bypassed more than once by any other process in the system.

Keywords: distributed algorithms; distributed computing; mutual exclusion; dining philosophers; on-line algorithms; dynamic graph algorithms

1.0 Introduction

Mutual exclusion is a long-standing, important problem in distributed systems. An early formulation of the problem, dining philosophers (DP) [2], dealt with resource sharing by neighbors in a ring. Chandy and Misra later described a generalized dining philosophers problem (GDP) [1] in which resource conflicts are specified in an arbitrary undirected graph. Vertices in the graph represent processes and edges represent resources shared between two processes. Chandy and Misra provided an algorithm to solve GDP in the case where the conflict graph is static. They also present a drinking philosophers problem and solution in which processes may need only subsets of resources to enter their critical states.

This paper presents an extension of GDP which allows nodes and edges to be added to the resource conflict graph dynamically. We call this the dynamic generalized dining philosophers problem (DGDP). With DGDP, the resource conflict graph is allowed to have nodes and edges added and removed arbitrarily. The algorithm presented in this paper for DGDP maintains the same properties of the GDP algorithm. Namely, it guarantees that the resource conflict priority graph remains acyclic with the addition of new nodes and edges (i.e., no deadlock) and it guarantees that no node will be denied needed shared resources indefinitely (i.e., no starvation). DGDP can be used to replace GDP [4].
The Chandy-Misra algorithm can be described as follows. Each shared resource between a pair of nodes is represented as a fork that may reside at either node or may be in transit from one node to another. The nodes or processes are designated as philosophers. Philosophers can be in one of four states: trying (hungry), critical (eating), exit (finished eating), and remainder (thinking). A philosopher must have all his incident forks in order to eat. Each philosopher notes the state of each incident fork as clean (which means the fork was used by the neighboring philosopher and given to this philosopher), dirty (which means the philosopher has used the fork or the fork was initialized to be dirty), or not present (which means the fork is either at the neighboring philosopher or in transit between them).

The behaviors of the philosophers in each of the four states are summarized in Figure 1. The direction of an edge in the priority graph relates to which philosopher has priority for that fork and is implicitly defined as follows. An edge in the priority graph is directed from A to B if any of the following holds: (1) the fork is dirty and at A, (2) this fork is clean and at B, or (3) the fork is in transit from A to B. If an edge is directed from philosopher A to philosopher B, then philosopher B has priority over philosopher A.

If a philosopher is thinking he freely gives up a fork. If a philosopher is hungry he will only give up a fork if the fork is dirty. (Forks are clean when they are received and become dirty when a philosopher eats.) A philosopher holds on to a clean fork knowing that the neighboring philosopher has already used it. A brief summary of the Chandy-Misra algorithm is presented in Figure 1.

<table>
<thead>
<tr>
<th>State</th>
<th>Behavior</th>
</tr>
</thead>
<tbody>
<tr>
<td>Trying (Hungry)</td>
<td>Request all forks that the philosopher doesn’t currently have.</td>
</tr>
<tr>
<td></td>
<td>Grant all requests for dirty forks. (Forks will be cleaned when sent.)</td>
</tr>
<tr>
<td></td>
<td>Defer all requests for clean forks.</td>
</tr>
<tr>
<td></td>
<td>May move to Critical state when all forks are present.</td>
</tr>
<tr>
<td>Critical (Eating)</td>
<td>Defer all requests for forks clean or dirty.</td>
</tr>
<tr>
<td></td>
<td>Make all forks dirty.</td>
</tr>
<tr>
<td></td>
<td>May move to Exit state at any time.</td>
</tr>
<tr>
<td>Exit (Finished Eating)</td>
<td>May move to Remainder state when all deferred requests have been granted.</td>
</tr>
<tr>
<td>Remainder (Thinking)</td>
<td>Grant all requests. (Forks will be cleaned when sent.)</td>
</tr>
<tr>
<td></td>
<td>May move to Trying state at any time.</td>
</tr>
</tbody>
</table>

**FIGURE 1. A brief outline of the Chandy-Misra algorithm**

Deadlock prevention depends upon the acyclicity of the priority graph. Since the priority graph is acyclic, one philosopher will always be able to get all his shared forks and eat. When that philosopher finishes eating, he will lower himself in the priority hierarchy by making all of his forks available. Eventually, each hungry philosopher will be able to eat since they will move closer to the top of the priority hierarchy with each philosopher that finishes eating.

The Chandy-Misra algorithm for GDP ensures that given an initially acyclic priority graph, the acyclicity of the graph is preserved throughout the algorithm. The algorithm maintains the acyclicity of the graph by ensuring that when a philosopher eats, it gives its neighbors priority for all the resources simultaneously. In the resource graph this means that the node making its resources available now has all it directed edges going out. This ensures that the graph is still acyclic since no cycles can propagate through this node.
The algorithm also ensures fairness by maintaining that a node must allow all its neighboring nodes to use the shared resources after it has used them. This means that within finite time a node will be able to acquire all the resources needed from its neighboring nodes. That is, since a philosopher immediately grants all pending requests when it finishes eating and a hungry philosopher never gives up a clean fork, we know that a hungry philosopher cannot be bypassed more than once by any neighbor. By transitivity, there is no starvation, although the algorithm does admit long waiting chains.

2.0 Modifications for Dynamic Graphs

The Chandy-Misra algorithm applies to static graphs. We have developed some simple modifications to extend the algorithm to work in systems where the set of processes and resource conflicts among them may change over time. In such systems, the conflict graph is dynamic with edges and nodes being added and removed as the system evolves. We need only concern ourselves with new edges being added and not with old edges being deleted, as an acyclic graph can not be made cyclic by removing edges.

When adding an edge to the resource conflict graph, it is necessary to consider the direction of the edges since this allows for the possibility of cycles in the priority graph. (See Figure 2.) To handle this problem, we make all the edges of one of the newly connected nodes be directed outwardly. Therefore, none of its incident edges can be involved in a cycle. This is similar to what happens when a philosopher eats in the Chandy-Misra algorithm. However, the relevant node may not possess all the resources in order to give priority to its neighboring nodes. In the Chandy-Misra algorithm, this would be analogous to making a fork dirty but not having the fork!

This situation is addressed by adding an additional resource state. In addition to the three states of a resource or fork relative to a node being 1) clean, 2) dirty, or 3) not present, we have added a fourth state: 4) not present but make dirty when it arrives. In the priority graph, if a resource is in states 2 or 4 the edge will be directed away from the node. If a resource is in state 1 the edge will be directed toward the node. (See Figure 3.)

Based on the above ideas, our solution to DGDP is summarized in Figure 4. The solution differs from the Chandy-Misra algorithm as follows. New edges are processed when adjacent nodes are in the Exit or Remainder states. When edges are added between nodes, the node with the higher ID gives priority to all its neighboring nodes for all its resources. This is the case even if the node doesn’t possess the resource. When the resource finally comes into the possession of the higher ID node, it will give priority for that resource to its neighboring node.
In the Chandy-Misra algorithm a receiving node has priority over its neighbor for an incoming resource (i.e., a fork arrives clean). That is, in the priority graph the edge is directed towards the receiving node. For a node with new edges we need to ensure that the neighboring nodes have priority over the node for all the resources (i.e., all forks are made dirty). That is, in the priority graph all the edges are directed away from the node. This needs to be facilitated even if the resource is not currently held by the node (i.e., make the forks immediately dirty when they arrive). With these simple modifications, the original Chandy-Misra algorithm can be made to handle resource conflict graphs with edges and nodes being added and removed over time.

3.0 Properties of the Algorithm

Listed below are the major properties of the algorithm.

New Edge Processing: In order to simplify the modifications, some restrictions have been added. Edges between nodes cannot be added when a node is in the process of acquiring all the
needed resources (state T) or when all the resources have been acquired and the appropriate functionality is being performed (state E). It may be possible to lift this restriction so that edges can be added while existing nodes are in states T and E but it is not a trivial matter, nor is it a pressing issue unless processes spend a lot of time in those states.

**Locality of Contention:** The algorithm provides locality of contention. That is, a philosopher that is hungry only needs to contend with its neighboring philosophers for shared forks. Disjoint sections of philosophers can eat simultaneously. However, just as in the original Chandy-Misra algorithm, it is possible for a long path to exist in the priority graph, resulting in a long waiting chain.

**No Starvation/Deadlock:** The algorithm shares with the Chandy-Misra algorithm the property of no starvation for neighboring nodes that are trying to acquire resources. Additionally, this algorithm ensures that all edge requests will be processed eventually by requiring that deferred edge requests be processed before a node transitions from the Exit state to the Remainder state. At this transition all neighboring nodes are giving priority for resources (i.e., all the node’s forks are dirty) and therefore the node must satisfy all requests for resources at this time. This guarantees that progress will be made and no deadlock will occur. The algorithm avoids starvation even though the resource graph can be constantly changing.

**Reconfiguration Progress:** A node must process all deferred edge requests as a condition of transitioning from the Exit state to the Remainder state. A node must also complete the processing of any current edge requests before it can transition from the Remainder state to the Trying state. (New edge requests may be deferred.) These two requirements ensure that edge requests will eventually be processed.

### 4.0 Practical Application - Module Migration

The modifications to the Chandy-Misra algorithm are useful in the context of work being done in the Playground distributed programming environment [3]. In Playground, modules, autonomous processes in a distributed computation, are connected by links that define their communication pattern. The links may change dynamically and modules are able to migrate from one process or machine to another. However, a module is not allowed to migrate while other modules connected to it are migrating. Thus, the shared resource between connected nodes is the exclusive migrate permission fork. A module must have all the forks from all its neighboring nodes to migrate.

If several connected Playground modules are wanting to migrate at the same time they must negotiate an order to perform migrations. The DGDP algorithm is used to resolve this ordering. Once a module wishing to migrate has all the forks from its neighbors it proceeds to migrate. In Playground, dynamic resource conflict graphs are required because new modules and connections can be added while the system is running.

### References

