Optimal Deadlock Detection in Distributed Systems Based on Locally Constructed Wait-for Graphs

Shigang Chen
Yi Deng
Paul C. Attie
Wei Sun

Univ. of Illinois, Urbana
Florida International University, Miami

1996
• Review of deadlocks
• Chen Algorithm
• Properties
• Algorithm Details + Example
• Conclusions
Deadlock Review 1

- A set of processes is deadlocked if each is waiting for an event that only another process can cause.

- Typically, the event is the release of some resource which the other process has exclusively acquired.

- Resources include peripherals such as printers, tape drives, CD-ROMs, scanners, database records and several other categories.
Deadlock Review II

• Required conditions:
  1. Mutual exclusion - given resource can only be assigned to one process
  2. Hold and wait - processes currently holding resources can request new resources
  3. No pre-emption - only the process can release its resources - they cannot be taken away by external processes
  4. Circular wait – there is a circular chain of 2 or more processes, each waiting for a resource held by the next member of the chain
Deadlock Review III

- Modeled via directed graphs
- Cycles (paths which connect back to starting node) are deadlocks.
- This invites graph theoretic analysis and algorithms
Deadlock Review IV - Solutions

- Ignore & Reboot - unix and windows both do this
- Detection and Recovery
  Detection
  - single type of resource
  - multiple resources
  - today's paper
  Recovery
  - pre-emption
  - rollback to checkpoint
  - kill processes
- Dynamic Avoidance – fancy algorithms
- Prevention – break one of the 4 required conditions
Chen Algorithm I

- No separate resources – only processes
- Nodes are distributed processes
- Graph arcs/edges – REQUEST made but not filled
- No arcs for allocated resources – when a process allocates a resource, the edge is removed
- Outgoing arc indicates a blocked process

Assume:

- message passing - no shared memory
- no communication problems – messages arrive in order, error free, no duplicates
Chen Algorithm II - Properties

- One node gathers all the data in a centralized fashion, and produces a complete analysis of the existing processes and will locate any deadlocks. Similar to centralized algorithms.
- Any node can run the algorithm, multiple nodes can start and run it independently and concurrently. There is no central controller, all nodes are symmetric and equivalent. Similar to distributed algorithms.

- Message complexity – optimal
- Time complexity – very competitive
Chen algorithm III

- Generalized requests – p out of q - adds a lot of complexity
- A cycle is a path with the same start and end node
- If all requests are 1 out of 1 then a cycle indicates a deadlock
- With p out of q requests a cycle is necessary but not sufficient for deadlock
- The full condition for deadlock is the generalized tie - formal definition on page 6 – compare with generalized deadlock set on page 5
● Node Data – maintained for every node:

<table>
<thead>
<tr>
<th>t</th>
<th>t.block</th>
<th>in</th>
<th>out</th>
<th>p</th>
<th>q</th>
</tr>
</thead>
</table>

in, out are lists of edges directed in and out

● Node data is created/updated by

REQUEST - node i requests resource from node j and updates its data

ACK - node j updates its data and sends ACK (but no data) to node i

node i is now waiting for node j

REPLY - node j allocates resource to node i, begins processing and the data for the REQUEST are deleted at both nodes
Node data is created/updated by

CANCEL  - if p REPLY's come in, the remaining REQUESTS are sent this message and removed from the node data

Possible sequences

i                j
REQUEST  --->
<---    ACK

node i waiting on j,
algorithm can run here

<---    REPLY
request is granted, data is removed
Another Possible sequence

i                        j

REQUEST  --->

<--- ACK

node i waiting on j,
algorithm can run here

CANCEL  --->

request data deleted
Another Possible sequence

\[\text{REQUEST} \rightarrow \text{ACK} \]

\[\text{node } i \text{ waiting on } j, \]
\[\text{algorithm can run here} \]

\[\text{FORWARD} \rightarrow \text{BACKWARD} \]
\[\text{request to send node data} \]
\[\text{includes all node } j \text{ data} \]
\[\text{continue processing} \]
Deadlock Detection Algorithm I

- Node i sends q REQUESTS and needs p to continue, thus it blocks
- ACKs come back rather than REPLYs, \( p > 0 \) REPLYs are still needed
- Node i begins iteratively building the wait-for graph
- Node i sends FORWARD message to the nodes in its out list
- Those nodes respond with a BACKWARD message containing their node data
- The wait-for graph is built for these nodes and analyzed for a generalized tie condition
Deadlock Detection Algorithm II

- If no deadlock is found, then iterate by sending FORWARD messages to all nodes which are 2 steps from node i and repeating the analysis.
- Then analyze the graph including nodes 3 steps out, etc.
- Terminate if:
  a. Deadlock found
  b. All edges out from node i are removed by the analysis
  c. All nodes reachable from i have been included and no tie/deadlock found
Deadlock Detection Algorithm III

- More Details of Analysis Phase
  1. Remove unmatched edges – these seem to be data errors or garbage collection
  2. Remove reducible edges – the request is grantable because node j has no outbound arcs and thus is not blocked
  3. Remove invalid edges – no path remains from i to j
  4. Search for tie – see previous page
Deadlock Detection Algorithm IV

- Notes on numerical example
- Generalized deadlock set
- Generalized tie subgraph
- If arc 8->7 is removed the whole tie collapses
- Demonstrate/argue that tie == deadlock
- Show iterative sets
- Explain 2n
- Explain q – p + 1
<table>
<thead>
<tr>
<th>Node</th>
<th>t</th>
<th>t.block</th>
<th>out</th>
<th>in</th>
<th>p</th>
<th>q</th>
<th>(q-p+1)</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>16</td>
<td>(2,3,4)</td>
<td>&lt;5,12&gt;&lt;9,15&gt;</td>
<td>2</td>
<td>3</td>
<td>2</td>
<td>(3, 4)</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td>&lt;1, 16&gt;</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>10</td>
<td>(5,6,7)</td>
<td>&lt;1, 16&gt;</td>
<td></td>
<td>2</td>
<td>3</td>
<td>2</td>
<td>(5,7)</td>
</tr>
<tr>
<td>4</td>
<td>11</td>
<td>(8,9)</td>
<td>&lt;1, 16&gt;</td>
<td></td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>(8,9)</td>
</tr>
<tr>
<td>5</td>
<td>12</td>
<td>(1)</td>
<td>&lt;3, 10&gt;</td>
<td></td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>(1)</td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td>&lt;3, 10&gt;</td>
<td></td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>14</td>
<td>(4)</td>
<td>&lt;3, 10&gt;</td>
<td></td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>(4)</td>
</tr>
<tr>
<td>8</td>
<td>13</td>
<td>(7)</td>
<td>&lt;4, 11&gt;&lt;9, 15&gt;</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>(7)</td>
</tr>
<tr>
<td>9</td>
<td>15</td>
<td>(1,8,10)</td>
<td>&lt;4, 11&gt;</td>
<td></td>
<td>2</td>
<td>3</td>
<td>2</td>
<td>(1, 8)</td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
<td>&lt;9, 15&gt;</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[ S = (1, 3, 4, 5, 7, 8, 9) \]  
\[ \text{tie} = S \]
Deadlock Detection Algorithm V - Summary

- Theoretical proof of correctness sketched in paper
- Time, message, data-traffic, and space complexities are better than or equal to existing algorithms
- First distributed algorithm which:
  1. detects every deadlock and no false deadlocks
  2. collects the whole structure of the deadlock set – thus easing resolution
Tannenbaum: If ever there was a subject that was investigated mercilessly during the early years of operating systems, it is deadlocks. The reason is that deadlock detection is a nice little graph theory problem that one mathematically inclined graduate student can get his jaws around and chew on for 3-4 years. All kinds of algorithms were designed, each one more exotic and less practical than the previous one.
Future Directions:

- Essentially, all this research has died out...
  When an operating system wants to do deadlock detection or prevention, which few of them do, they use one of the methods discussed in this chapter.
  There is still a little research on distributed deadlock detection, however. ... none of it is even remotely practical in real systems. Its main function seems to be keeping otherwise unemployed graph theorists off the streets.
