An Enhanced Distributed Deadlock Detection and Recovery in Process Networks

Mohammad H. Al Shayeji, Abbas Fairouz, and M. D. Samrajesh

Abstract—The Process Network (PN) model consists of multiple concurrent processes communicating over a unidirectional First-In-First-Out (FIFO) queue. Process networks are widely used for functional parallelism in digital signal processing and streaming data applications such as MPEG encoding and decoding. However, the bounded-memory scheduling policy of process networks can lead to process deadlock. The current deadlock detection and recovery algorithms in process networks can be enhanced by incorporating the role of a coordinator.

In this paper we propose an enhanced distributed deadlock detection and recovery algorithm using blocked-lists and a coordinator to detect and recover from deadlocks. The distributed blocked-list algorithms detect the blocked processes and merge them with the existing blocked-lists. When a cycle is detected in the blocked-list, the coordinator is invoked to recover the system from the deadlock by either increasing the size of the channel for artificial deadlocks or by terminating the lowest priority processes from the blocked-lists. Our evaluation of the algorithm shows that the proposed blocked-list algorithm outperforms other algorithms by requiring fewer messages to detect and recover from deadlock.

Index Terms—Deadlock detection, deadlock recovery, distributed systems, kahn process networks (KPN).

I. INTRODUCTION

The Process Network (PN) model is a type of formal dataflow model that is highly useful for modeling and exploiting functional parallelism in data streaming applications such as signal processing, MPEG encoding and decoding [5], [9]. The model consists of concurrent processes communicating via unidirectional channels. It maps easily to multi-threaded and/or multi-processor systems [10], [16].

The Process Networks model was first proposed by Kahn in 1974 [4], and was termed as the Kahn Process Network (KPN). It consisted of concurrent processes communicating over the FIFO unidirectional channel with bounded memory. Processes read and wrote atomic data or tokens from/to channels. A process that read from an empty channel was suspended and could only continue when the channel contained sufficient data items. Tokens were received in the order they were transmitted. Process networks can be represented by a directed graph where nodes represent processes and arcs represent infinite FIFO queues that connect the processes as shown in Fig. 1 [4].

Some of the interesting properties [11], [19] of process networks that make them a suitable model for data streaming applications are:

- Each process is a sequential program that consumes tokens from its inputs queues and produces tokens to the output queues.
- Each queue has one source and one destination.
- The network has no global data.

Since unbounded channels are impractical in reality, bounded memory is proposed in [6], this leads to process blocks while attempting to write data in a full channel and consecutively introduces deadlocks in process networks. The challenge is to have an efficient algorithm that detects & recovers from deadlocks. Henceforth, the reference to PN refers to a bounded channel Process Network.

There are two types of deadlocks in Process Networks [17]. The first type is real deadlock that occurs when all the processes are blocked for reading from a full channel. The second type is artificial deadlock that occurs when at least one of the processes is blocked for write on a full channel. An efficient deadlock detection and recovery algorithm in PN is needed to minimize the recovery time and increase the degree of concurrency in processes.

![Fig.1. A typical layout of process network](image)

We present an enhanced distributed algorithm for detecting a deadlock using a coordinator utilizing a distributed blocked-list. The coordinator recovers from deadlocks by terminating the lowest priority process for real deadlock or by increasing the size of the channel in the case of an artificial deadlock.

The paper is structured as follows: Section II discusses related work, the enhanced distributed deadlock detection and recovery algorithm are presented in section III, our evaluation of algorithm is presented in section IV, discussion on the evaluation is presented in section V and our conclusion and future work are in section VI.

II. RELATED WORK

Deadlock detection and recovery in distributed networks
has been studied in [1], [14], [15]. However, not much has been done in the area of deadlock detection in PN.

A local deadlock detection and resolution was proposed in [5], here the deadlock is resolved by increasing only the size of bottlenecked channels so as to avoid unnecessary large channels. However, this lacks dedicated monitoring of deadlocks thus leading to a delay in detection and resolution of deadlocks.

A distributed deadlock detection algorithm was proposed in [3]. The algorithm is a single resource algorithm in which at any given time a process could only wait on a single resource. It is similar to a node that could be blocked on only a single other process. The model could detect a deadlock however it could not locate and/or resolve it.

A runtime mechanism for handling artificial deadlocked situations in process networks was proposed in [8]. The detection of artificial deadlocks was done early using extended CPU architecture; however, this approach did not deal with real deadlocks.

Hierarchical approach for deadlock detection at run time is proposed in [7]. Here the PN is partitioned; the partitions are given local deadlock detection observers that are supervised by a global observer that leads to a hierarchical deadlock detection approach. However, the overhead of partitioning and the criteria for partition are some of the challenges in this model.

Recently a lightweight deadlock avoidance for streaming computations was presented in [12] by adding extra dummy tokens to the data streams and this did not require global run-time coordination among nodes or dynamic resizing of buffers. However, this method takes considerable time for computing the dummy token intervals.

Our approach of deadlock detection and recovery is different from the earlier algorithms. We try to detect the deadlock using a coordinator utilizing blocked lists that are more efficient and guarantee the detection of deadlock in process networks with fewer messages, furthermore in the case of recovery; the coordinator terminates the lowest priority process using the blocked list or increases the channel capacity.

III. PROPOSED SOLUTION

A. Introduction

Blocked-List Deadlock Deduction algorithm (BLD) is based on the concept of a message passing through the entire blocked processes during the execution. This algorithm is an extension of [5] which is actually presented for distributed systems that consists of processes and resource managers. The deadlock would be detected by one of the blocked processes and that process sends the list of all blocked processes on the deadlock cycle to the coordinator for recovery.

B. Assumptions and Definitions

The algorithm is based on the following assumptions:

- The communication between processes is 1:1 connection, i.e. each channel has one sender and one receiver.
- Each process has a system-wide unique identifier represented by positive integer.
- The coordinator is an entity and is always up and available for managing the system and dealing with the deadlocks.

<table>
<thead>
<tr>
<th>Algorithm 1: Blocked-List Deadlock Detection Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>if B-message, ε Blocked-List, ( GBL = B-message, ε B-message ), for all blocked processes related to ( P_i ) (B-messages, ε GBL (Call Coordinator-Send Blocked List to BLD) else ( GBL = OBL ) V B-messages, ε GBL end for</td>
</tr>
<tr>
<td>( Blocked-List = Blocked-List, ε Blocked-List, ε B-message(( ch_i, size_i, block-type, id )) ) for all blocked processes related to ( P_i ) else Blocked-List = Blocked-List, ε B-message, ε GBL end for</td>
</tr>
</tbody>
</table>

B-message is sent from the blocked process to the requested process and stored in the requested process’s blocked-list. This message consists of: \( ch_i, size_i, block-type \), and \( id \) (‘i’ represents the requesting process and ‘j’ represents the requested process), \( ch_i \) refers to the channel that connects process i and process j and \( size_i \) represents the size of that channel. block-type refers to the type of blocked process (i.e. read or write), \( id \) refers to the unique id of process, and this identifies the process’s priority in the system.

C. Role of Coordinator

In our design, when process, enters the process network, the coordinator assigns process id. The id of a process determines its priority in the system. Once the blocked list deadlock detection algorithm detects a deadlock, it sends the blocked-list to the coordinator. The coordinator checks whether it is real deadlock (i.e. all the processes are blocked for reading from a full channel) or artificial deadlock (i.e., at least one of the processes is blocked for write on a full channel). Artificial deadlocks are handled in a manner similar to that in [5] wherein the size of the channel is increased to resolve the deadlock. On the other hand, for real deadlocks the coordinator terminates the process with the lowest priority (i.e. highest id) in the blocked-list.

D. Blocked list Deadlock Detection(BLD) Algorithm

The foundation of the algorithm is based on [13] where a node is used as the organizer of certain tasks that is distributed among several computers in a network. In our model, the coordinator plays an important role; each process’s priority is based on the process id that is assigned by the coordinator. The id is used for terminating the process with the lowest priority in case of real deadlock detection in the process network.

The Blocked list algorithm is invoked when a process is unable to get its request. When process, attempts to send a message to process, on a full channel, process, blocks process, and process, creates a new B-message to be sent to process. Before sending the B-message, process, checks its blocked-list. If it has a message sent from process, then a
cycle exists and the blocked-list is sent to the coordinator. The coordinator determines whether the deadlock is real or artificial and deals with it accordingly.

```
Algorithm 2: Blocked-List Deadlock Recovery Algorithm

count = size(Blocked-List)
for all B-message, i in Blocked-List
   if (Blocked-List.B-message.type = read)
      continue
   else
      break
   end for
if (i == count)
   Terminate the lowest priority in Blocked-List, clear Block-list
else
   Increase the size of ch_i
endif

Fig. 3. Deadlock recovery algorithm-Coordinator
```

If a cycle does not exist, then blocked-list, is merged with blocked-list of process, and it sends the B-message of process, to all the blocked processes related to process.

E. Blocked list Deadlock Recovery (BLR) Algorithm

Once the coordinator receives the blocked list, it checks the list to determine the type of process block. If the type is read for all the messages in the blocked list, then the deadlock is identified as a real deadlock, and the coordinator terminates the process with the lowest priority in the blocked list. If at least one of the blocked messages type is write, then the deadlock is an artificial deadlock and the coordinator increases the size of the channel that caused the write block, thus resolving the deadlock. All other processes in the blocked-list become active again, and the terminated process is cleared from the respective blocked-lists.

IV. ALGORITHM EVALUATION

A. Algorithm Complexity

The algorithm has a single loop for sending the blocked list messages through the whole link. The message complexity is \(O(n)\) which is comparatively better than [5]. The algorithm has a better performance since the blocked-list messages are not recreated but merged with the existing blocked-list.

Once the last process enters the blocked link (i.e. which cause of cycle), it checks the processes that relate to the cycle and then sends the blocked-list message for detecting the deadlock cycle. The recovery algorithm on the coordinator side has a complexity of \(O(n)\). Once a process detects a cycle in the system, it sends the blocked-list to the coordinator.

B. Case Study

The three most popular cases that represent most of the process scenarios are 1) forward link, 2) backward link, and 3) random link deadlocks. The movement of the B-messages is traced at each step for each of these cases.

1) Case : Forward Link

The forward link describes forward-directional sequences of process requests. One process requests another process, and the requested process will request another process, etc.

For the forward link deadlock, as shown in Fig. 5, assuming that \(P_1\) tries to send a message on read to \(P_2\). \(P_2\) is blocked due to a full channel. Now, when \(P_2\) blocks \(P_1\) (step 1), \(P_1\) sends a B-message \((P_2, ch_{1,2}, size_{1,2}, read)\) to \(P_2\) and stores this message in Blocked-List. Before sending B-message from \(P_2\), \(P_1\) checks if B-message exists in Blocked-List. If B-message does not exist on Blocked-List, then \(P_1\) sends B-message to the blocked-list. Now, Blocked list = \{B-message\}. Next, when \(P_2\) blocks \(P_1\) on a read, \(P_1\) checks its own message on Blocked-List. \(P_2\) then sends the Blocked list contents and its own message, B-message, message to \(P_3\), and stores it in Blocked-Lists. Hence, Blocked-List = \{B-message, B-message\}.

The same is repeated for steps 3, 4, and 5. Finally, in step 6, \(P_1\) blocks \(P_6\) on a read and \(P_1\) detect its own message in Blocked-List, that shows the presence of a deadlock. Thus, \(P_1\) creates a new Blocked list called Global Blocked List (GBL) that is used to check all the process related to the current deadlock. B-message_6 and B-message_1 are added to GBL. Next, \(P_1\) passes GBL to the next process that blocked \(P_1\) which was \(P_2\). \(P_2\) records B-message_2 in GBL and passes it to the next blocked process. Each process in the cycle records its B-message in GBL and passes it to the next process until it reaches the first process that detects the deadlock that is \(P_6\). Finally, \(P_6\) sends BL to the coordinator to recover from the deadlock.

When the coordinator receives GBL from \(P_6\), it first checks if the deadlock is real or artificial. It checks the type of
deadlock since all processes are blocked on read. The coordinator terminates the lowest priority process that is $P_6$.

2) Case 2: Backward Link

The backward link also describes sequential process requests but in a backward direction. A process requests another process, and the requesting process may be requested by another process and so on. As shown in Fig. 6, initially $P_6$ tries to send a message on read to $P_1$ but it is blocked according to full channel. When $P_1$ blocks $P_6$ (for step 1), $P_6$ sends a $B$-message$_6$ ($P_6$, ch$_{6,1}$, size$_{6,1}$, read) to $P_1$ and stores this message in Blocked-List$_1$. Before sending the B-message from $P_6$, $P_6$ checks if B-message$_1$ is available on Blocked-List$_1$. If B-message$_1$ does not exist in Blocked-List$_1$, then $P_6$ sends the blocked-list contents and its own B-message$_6$. Hence, Blocked list$_1$ = {B-message$_6$}. In step 2, $P_3$ tries to send a message on write to $P_6$. Before receiving the blocked message from $P_3$, $P_6$ checks if its B-message$_6$ is available on Blocked list$_1$. If B-message$_1$ does not exist in Blocked list$_1$, then $P_3$ sends Blocked list contents and its own B-message$_3$. Thus, Blocked list$_6$ = {B-message$_3$}. Blocked-List$_1$ = {B-message$_6$, B-message$_3$}.

The same is repeated for steps 3, 4, and 5. Finally, in step 6, $P_2$ blocks $P_1$ on a read and $P_2$ checks its own blocked message availability on Blocked list$_1$. $P_2$ records its own message in Blocked list$_1$ that shows the presence of a deadlock. Thus, $P_2$ creates a new Blocked list GBL. B-message$_2$ and B-message$_3$ are added to GBL. Next, $P_2$ passes the BL to the next process that blocked it that was $P_5$. $P_5$ records B-message$_1$ in GBL and passes it to the next blocked process. Finally, $P_1$ sends GBL to the coordinator to recover from the deadlock.

When the coordinator receives BL from $P_6$, it first checks whether the deadlock is real or artificial. It detects that it is an artificial deadlock because $P_6$ blocked $P_5$ on write, hence coordinator increases ch$_{5,6}$.size.

3) Case 3: Random Link

The random link is the most common deadlock in real applications. The procedure for sending the blocked-lists and B-messages through the link is a combination of forward and backward links. A random link deadlock is shown in Fig. 7. Initially, $P_3$ tries to send a message on read to $P_1$ but it is blocked due to a full channel. When $P_3$ blocks $P_4$ (step 1), $P_4$ sends a B-message$_3$ to $P_3$ and stores this message in Blocked-List$_3$. Thus, Blocked list$_3$ = {B-message$_3$}. Next, in step 2, $P_1$ tries to send a message on read to $P_2$. $P_2$ blocks $P_1$ and it receives the Blocked-List$_1$ and B-message$_3$ from $P_1$. Hence, Blocked-List$_1$ = {B-message$_3$}.

The same is repeated for steps 3, 4, and 5. Next, in step 6, $P_4$ blocks $P_3$ on a read and $P_4$ detects its own message in Blocked-list$_3$ that shows a deadlock occurrence. Thus, $P_4$ creates a new global blocked-list. B-message$_3$ and B-messages is added to GBL. $P_4$ then passes GBL to the next process that blocked $P_4$ that was $P_5$. $P_5$ records B-message$_1$ in GBL and passes it to the next blocked process which is $P_2$. $P_2$ records its own message on GBL and passes to $P_3$. $P_3$ then detects its own message in GBL.

Finally, $P_1$ sends GBL to the coordinator to recover from the deadlock. The coordinator receives GBL from $P_1$ and checks if the deadlock is real or artificial. In case the coordinator detects a real deadlock, it terminates the lowest priority process in the list that is $P_5$.

V. DISCUSSIONS

We compared our algorithm with [5] and [2] considering the scenario in Case-3. Specifically, we focused on the number of messages passed since this has a significant impact on the performance of a process network [18]. Fewer messages result in faster deadlock detection and recovery, thus this provides better overall performance and a higher degree of concurrency among processes in process networks.

We traced the number of messages passed in Case-3 Random link, which is a combination of both forward and backward links, using our proposed Blocked List Deadlock detection (BLD) algorithm and against Wei Huang [2], Prieto[5]. Our comparison showed that the proposed approach required fewer messages to detect deadlock in process networks because the blocked list was distributed and the blocked process was merged with the existing list.

The deadlock detection in each case was done once the process in the link was blocked for its requests on read/write and formed a cycle. Moreover, our algorithm terminated only the lowest priority process in the blocked-list instead of terminating the whole link as in [5], thus avoiding the cost of restarting other processes. In addition, in artificial deadlock recovery, unlike in [9], only the corresponding blocked channel capacity was increased and resulted in better utilization of channel capacity.

VI. CONCLUSION & FUTURE WORK

We have presented an enhanced distributed deadlock detection and recovery algorithm by detecting a deadlock using a coordinator and blocked-list. The distributed blocked-list detects the blocked processes and merges it with the existing blocked-list. When a cycle is detected in the blocked-list, the coordinator is invoked to recover from the deadlock by means of terminating the lowest priority process for real deadlock or by increasing the channel capacity in case of an artificial deadlock.

Our comparative evaluation of the algorithm shows that the proposed blocked-list algorithm requires less number of messages for deadlock detection resulting in faster deadlock detection and recovery on the performance of a process network [18]. Fewer messages result in faster deadlock detection and recovery, thus this provides better overall performance and a higher degree of concurrency among processes in process networks.
detected and recovered, and provides better performance and a higher degree of concurrency among processes in process networks. In the case of artificial deadlock recovery, only the required channel’s size is increased thereby providing better utilization of channel capacity. Further, in case of real deadlock, it terminates only the lowest priority process in the blocked-list instead of terminating the whole list, thereby reducing the recovery and restart time of processes.

As a part of future work, we plan to compare the message passing delay and its impact on the performance of deadlock detection and recovery in process networks.

REFERENCES