Placing forced checkpoints in distributed real-time embedded systems

by Jane-Ferng Chiu and Ge-Ming Chiu

An efficient scheme for placing forced checkpoints in a distributed real-time embedded system so as to eliminate useless checkpoints is presented. The notion of primary non-causal intervals is introduced; these intervals are shown to be the only candidates that need to be considered for inserting a minimum number of forced checkpoints. An efficient algorithm is then used to identify the primary non-causal intervals where forced checkpoints should be inserted. The algorithm first converts the original problem to another problem on a directed graph, which may reflect the existence of useless checkpoints. The new problem can be efficiently solved using existing methods.

Control applications such as industrial process control, aircraft, robotics and automotive control have to maintain a very high level of safety, typically defined as the avoidance of unplanned events resulting in death, injury and damage, or loss of property. Fig. 1 illustrates the computing time of a real-time embedded system with horizontal lines. Fig. 1(*a*) shows that a process finished the computing before the deadline without failure, but Fig. 1(*b*) shows that the completed time will exceed the deadline when a process has to be repeated from the beginning because of failure. Thus, fault-tolerant techniques are often embedded to provide predictable performance in the presence of malfunction of safety-critical systems.[1] Work on fault masking achieves this goal by using spatial redundancy, which is expensive. A less expensive approach is that of using time redundancy, which typically does not require a large amount of extra resources and is amenable to embedded systems.[2]

A *checkpoint* at a process is the saved state of the process, which includes values of the data variables and contents of the system registers.[3] In Fig. 1, the short orange bars indicate checkpoints. If a failure occurs in the system, the failed processes may recover from their checkpoints without restarting from the beginning, as shown in Fig. 1(*c*) and (*d*). The more checkpoints there are in a process, the less time is required for repeating a process. However, checkpoint schemes depend on time redundancy, and they could also affect the correctness of the system by causing deadlines to be missed. One way to guarantee that all timing and resource constraints will be met is to statically schedule all communication and checkpoints to meet their deadlines.

In many cases and citations we need to statically schedule communication and checkpoints, such as reliability prediction of distributed embedded fault-tolerant systems, embedded diagnostic systems etc. Therefore, it is imperative to know the occurrences of all checkpoints and the communication patterns of all nodes prior to a distributed computation beginning in such systems. In particular, a set of checkpoints may be specified for each process that reflects the process's individual needs. These checkpoints are called basic checkpoints.[4] Basic checkpoints are normally determined independently and are inserted in the processes prior to their execution.

However, in distributed real-time embedded systems, dependency between the states of different processes may be caused by message communication. In Fig. 2, there are three processes, P_1, P_2, and P_3, with six

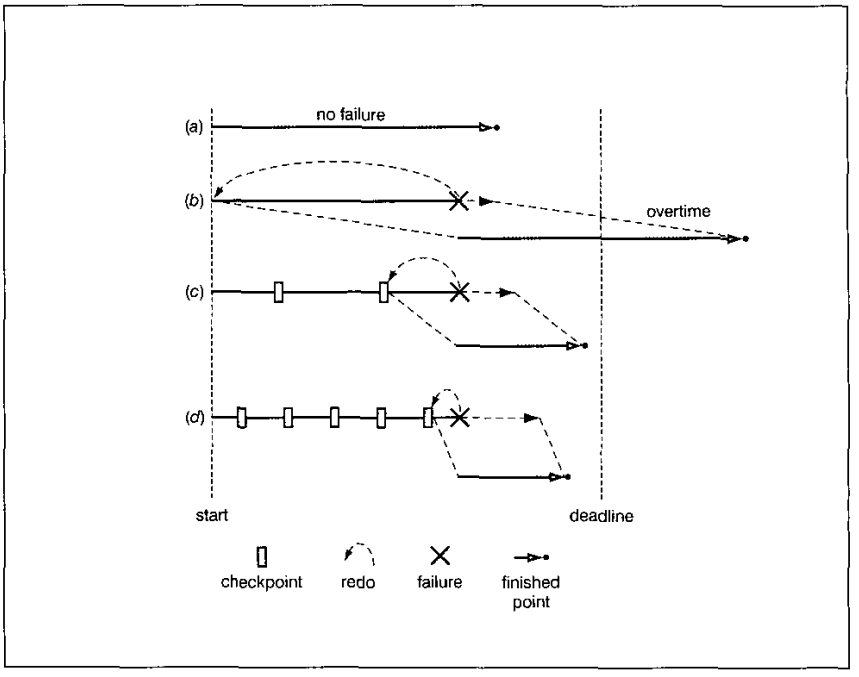

Fig. 1 Why processes need to take basic checkpoints

messages m_1, m_2, ..., and m_6 passed among them. As basic checkpoints of the processes are established independently as in Fig. 2, there is a risk that some basic checkpoints may never be included in any consistent global state.[2] Since $C_{2,1}$ happens before $C_{1,2}$ and $C_{1,2}$ happens before $C_{2,2}$, there is no consistent checkpoint at P_2. So, $C_{1,2}$ is a useless checkpoint. Similarly, $C_{3,2}$ and $C_{3,3}$ are useless checkpoints. Such checkpoints are useless[5] in the sense that they can never be used for rollback recovery, and thus are a waste of time and resources. In its worst case, useless checkpoints may cause the domino effect with unbounded rollback. In order to prevent basic checkpoints from becoming useless, additional checkpoints, called forced checkpoints,[2] must be added at the processes, as illustrated in Fig. 3 (two) and Fig. 4 (one). Forced checkpoints consume resources as well as incur time overhead on the system. Hence it is desirable to insert a minimum number of forced checkpoints, just enough to make all basic checkpoints useful,[6] as shown in Fig. 4 for a distributed real-time system.

There is a rich body of literature about checkpoint schemes for distributed computing systems. The class of *co-ordinated checkpoint protocols*[2] achieves domino-free recovery by sacrificing the processes' individual requirements and commonly incurs unnecessary extra overheads. This approach adopts passing additional control messages such as acknowledgments of all processes to synchronise their checkpoint activities. In this fashion many processes may trigger a large number of control messages and needless forced checkpoints simultaneously. Because of the excessive number of control messages on a single channel, such as CAN, each process not only has to wait for the decision of corresponding processes but also suffers from the collision of jamming messages.

In contrast, the *communication-induced protocols*[7] determine the insertion of forced checkpoints based on communication events occurring in the system. This approach is particularly interesting when the communication pattern of underlying application must not be changed. The co-ordination of this approach is implicit: each application message is allowed to piggyback a control message. A process uses this control information to know whether it must take a forced checkpoint. The local history is encoded for most general distributed computing systems and no protocol takes advantage of all the underlying information. Unfortunately, it is impossible to take the minimum number of forced checkpoints[2] due to lack of complete information. Some forced checkpoints may be wasted, while the absence of generation of a wasted checkpoint may be related, since processes take forced checkpoints based on only suspected failure modes. More than one forced checkpoint may be trigged concurrently in such a suspected situation. Furthermore, some forced checkpoints may become useless checkpoints in some conditions. Suspected failure modes may bring to a system a great number of unnecessary forced checkpoints. Due to the limited amount of communication information known to the processes, these protocols are likely to insert a number of forced checkpoints that are excessive.[7] Hence these protocols are not really provided for handling minimal forced checkpointing in distributed real-time embedded systems.

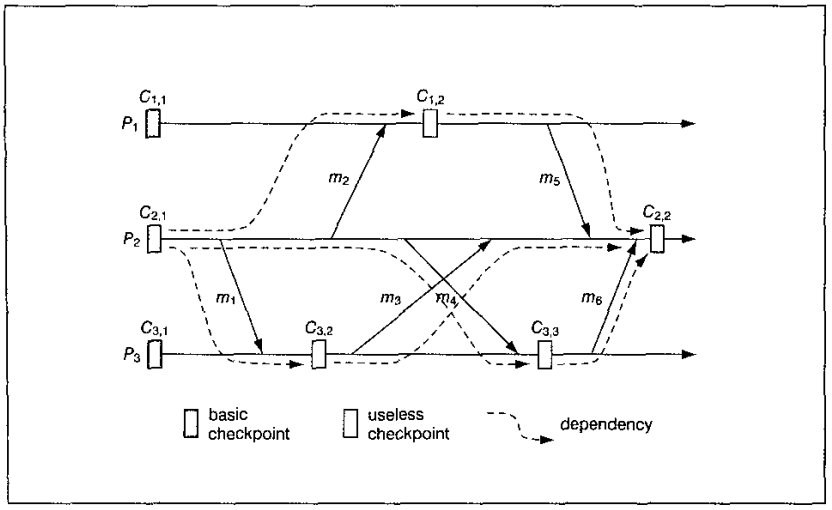

Fig. 2 Useless checkpoints

In this paper, we present a scheme for scheduling a minimum number of forced checkpoints for a static real-time embedded system so as to eliminate useless checkpoints. The notion of primary non-causal interval is first introduced. We show that these intervals are the only candidates that need be considered for inserting forced checkpoints. An efficient algorithm is then used to identify the primary non-causal intervals where a minimum number of forced checkpoints should be added.

System model and preliminaries

We consider a distributed real-time embedded system consisting of N processes, denoted by P_1, P_2, ..., P_N, which are interconnected by a communication network. The processes communicate with each other by message passing. In the system, processes take checkpoints based on individual fault-tolerance requirements. These checkpoints are highly application-dependent. Without elaborating further, we assume that a set of suitable checkpoints has already been determined for each process of the system. These checkpoints are called basic checkpoints. Moreover, our system is a statically scheduled one such that the communication pattern of the processes and the occurrences of basic checkpoints are known *a priori*.[1]

We can view the execution of each process in isolation as a sequence of events occurring in the process. In our model, three types of events—the send events of messages, the delivery events of messages and the checkpointing events—are considered. In the following, the send event and the delivery event of a message m are denoted by *send*(m) and *deliver*(m), respectively. In addition, $C_{i,j}$ represents the jth checkpoint at process P_i. The sequence of events occurring at P_i between two consecutive checkpoints $C_{i,x-1}$ and $C_{i,x}$ ($x > 0$) is called a checkpoint interval. The states of the processes may become dependent on one another due to inter-process communication. Hence the events of the processes can be ordered by the well-known relation (denoted by $\xrightarrow{h}$), proposed by Lamport.[6]

In a distributed system, a set of checkpoints, one from each process P_i, $1 \le i \le N$, form a consistent global checkpoint if and only if there exists no pair of checkpoints in the set that have prior relationship relation between them. A checkpoint of a process is said to be useless[5] if it cannot be included in any consistent global

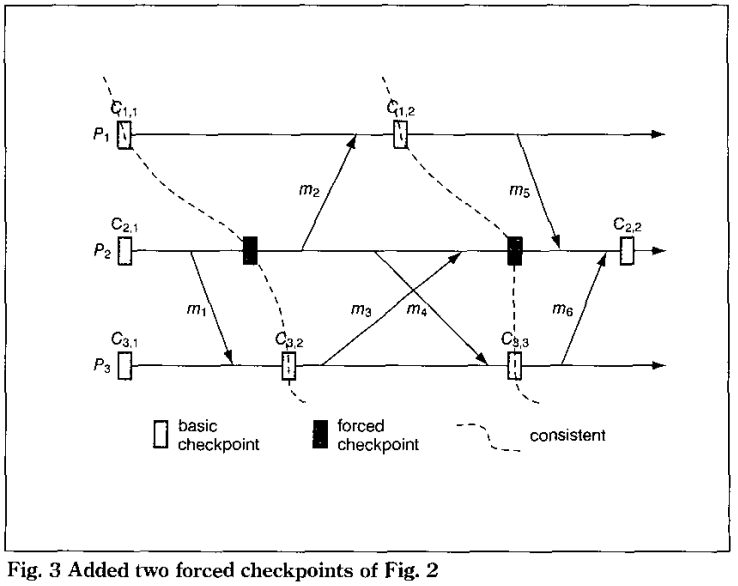

Fig. 3 Added two forced checkpoints of Fig. 2

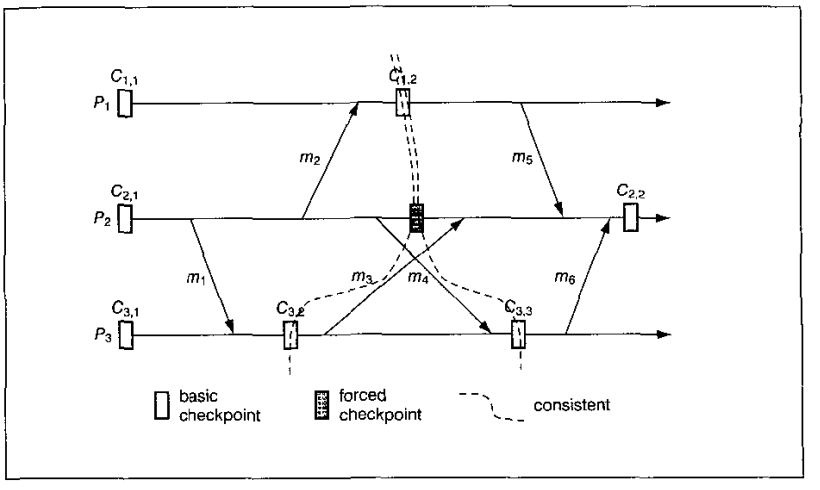

Fig. 4 Added one forced checkpoint of Fig. 2

checkpoint. Essentially, useless checkpoints indicate a complete waste of time and resources and may consequently cause uncontrollable rollback recovery, which should be avoided for a real-time embedded system. Useless checkpoints can be characterised by the existence of zigzag cycles (or *Z*-cycles),[8] as in Fig. 5.

Definition 1: *A Z-cycle for a checkpoint* $C_{i,x}$ *is a logical path from* $C_{i,x}$, *to itself that involves a sequence of messages* $m_1, m_2, \ldots, m_k$, $k \leq 2$, *such that:*
1 m_1 *is sent by* P_i *and* m_k *is received by* P_i;
2 $C_{i,x} \xrightarrow{h}$ *send(m1);*
3 deliver$(m_j) \xrightarrow{h}$ *send*(m_{j+1}) *or deliver*(m_j) *and send*(m_{j+1}) *occur in the same checkpoint interval (although send*$(m_{j+1}) \xrightarrow{h}$ *deliver*(m_j)*), for all* $1 \leq j \leq k$;
4 deliver$(m_k) \xrightarrow{h} C_{i,x}$.

It is evident that there must be at least two messages associated with a *Z*-cycle. Netzer and Xu[8] showed that a checkpoint is useless if and only if there exists a *Z*-cycle for the checkpoint. Hence a system with no useless checkpoints must be free of *Z*-cycles.

Fig. 5 illustrates an example of the distributed computing of a real-time embedded system with a given communication pattern and a predetermined set of basic checkpoints. In the Figure, there are four processes, P_1, P_2, P_3 and P_4, with 21 messages m_1, m_2, m_3, ..., and m_{21} passed among them. The short orange bars indicate basic checkpoints. It is easy to see that $C_{2,4} \xrightarrow{h} C_{1,5}$. Hence $C_{2,4}$ and $C_{1,5}$ cannot be included in the same global consistent checkpoint. Similarly, we have $C_{1,5} \xrightarrow{h} C_{2,5}$, thus $C_{1,5}$ and $C_{2,5}$ cannot be included in the same global consistent checkpoint either. In fact, $C_{1,5}$ is a useless checkpoint in this system. There is a *Z*-cycle involving a message chain of m_{15} and m_{12} for $C_{1,5}$. Another *Z*-cycle for $C_{1,5}$ includes a sequence of messages m_{15}, m_{19}, m_{21}, m_{17}, m_{16} and m_{12}. There may be more than one *Z*-cycle associated with a useless checkpoint. Similarly, $C_{1,2}$, $C_{4,3}$, $C_{4,4}$, $C_{1,5}$, $C_{2,3}$ and $C_{3,4}$ are also useless checkpoints.

In order to eliminate useless checkpoints we need to add extra checkpoints in the processes. These extra checkpoints are called forced checkpoints. It is desirable that the number of forced checkpoints can be made small so that the incurred overhead is reduced.

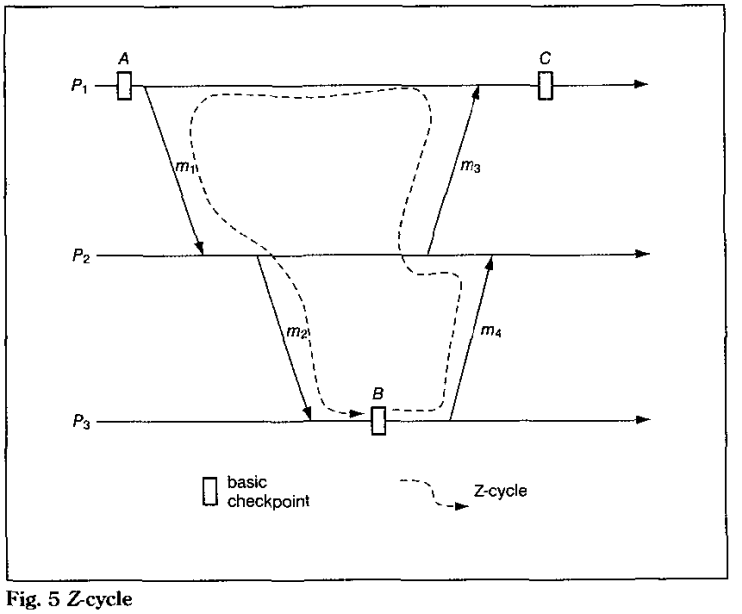

Fig. 5 Z-cycle

Prime non-causal interval (PNCI)

As stated above, the interval for taking a forced checkpoint must be part of a *Z*-cycle. There are two kinds

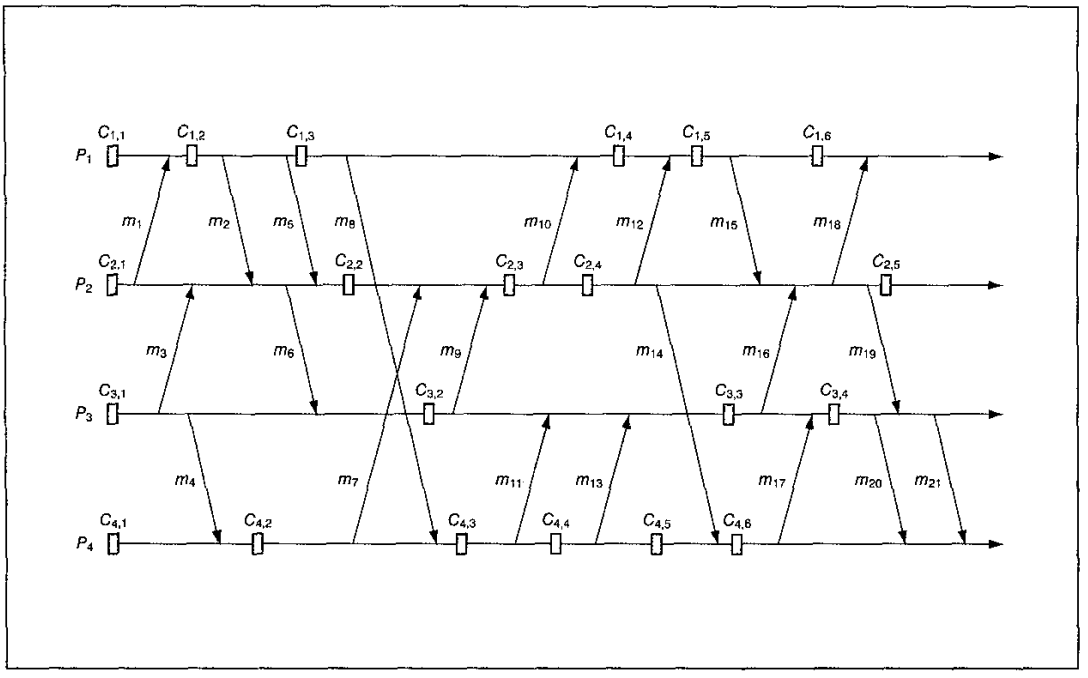

Fig. 6 Distributed computation and a set of prearranged basic checkpoints

of interval in a *Z*-cycle. Consider some process P_i in the system. Suppose that message m_x is sent by P_i, and message m_y is received by P_i. If we have $deliver(m_y) \xrightarrow{h} send(m_x)$, we call the interval between $deliver(m_y)$ and $send(m_x)$ a *causal interval*. We call the interval between $send(m_x)$ and $deliver(m_y)$ at P_i a non-causal interval (*NCI* for short) if $send(m_x)$ and $deliver(m_y)$ are included in the same checkpoint interval, i.e. there exists no checkpoint between $send(m_x)$ and $deliver(m_y)$ at P_i. For example, consider the system in Fig. 6. At P_2, the interval between $deliver(m_2)$ and $send(m_6)$ is a causal interval by definition. The interval between $send(m_9)$ and $deliver(m_{11})$ at P_3 is a non-causal interval by definition, so is the interval between $send(m_9)$ and $deliver(m_{13})$. However, at P_4, the interval between $send(m_{13})$ and $deliver(m_{14})$ is not a non-causal interval because checkpoint $C_{4,5}$ exists between the two events.

Definition 2: *A non-causal interval is called a prime non-causal interval (PNCI for short) if there exists no other send or delivery event in the interval.*

For example, in the system of Fig. 6, the non-causal interval between $send(m_9)$ and $deliver(m_{11})$ at P_3 is a *PNCI*. However, the non-causal interval between $send(m_9)$ and $deliver(m_{13})$ is not a *PNCI* as $deliver(m_{11})$ exists in the interval. It is easy to see that any non-causal interval must contain at least one *PNCI*. As described previously, any *Z*-cycle includes at least one non-causal interval.

From our Theorem 1 (details of the theorem can be found in Reference 2), the problem of placing a minimum number of forced checkpoints to achieve *Z*-cycle free property can be solved by identifying a proper set of *PNCI*s in the system.

Identifiying locations for forced checkpoints

In this section we present an algorithm that identifies the *PNCI*s where forced checkpoints may be placed to make the system *Z*-cycle free. Consider a given distributed embedded system with N processes $P_1, P_2, \ldots, P_n$, and a set β of basic checkpoints. Again, let $\Gamma\beta$ denote the system. Our algorithm is implemented in two steps. In the first step, we generate a directed graph from $\Gamma\beta$, which illustrates all possible paths that may be involved in *Z*-cycles. The original problem is converted to another problem on this directed graph. We then solve the new problem in the second step.

Constructing ZC-digraph

We generate, from $\Gamma\beta$, a directed graph $G(V, E)$, called the *ZC-digraph*, to facilitate the identification of *Z*-cycles in the system. In the following treatment, we only consider the send, the delivery and the checkpoint events. For $\Gamma\beta$, the directed graph G is constructed by having each event correspond to a distinct vertex in G. Directed edges in G reflect all possible links that may be included in *Z*-cycles of the system. Let $e_{i,j}$ represent the jth event in process P_i. To facilitate our description, the notations for events are also used to represent the corresponding

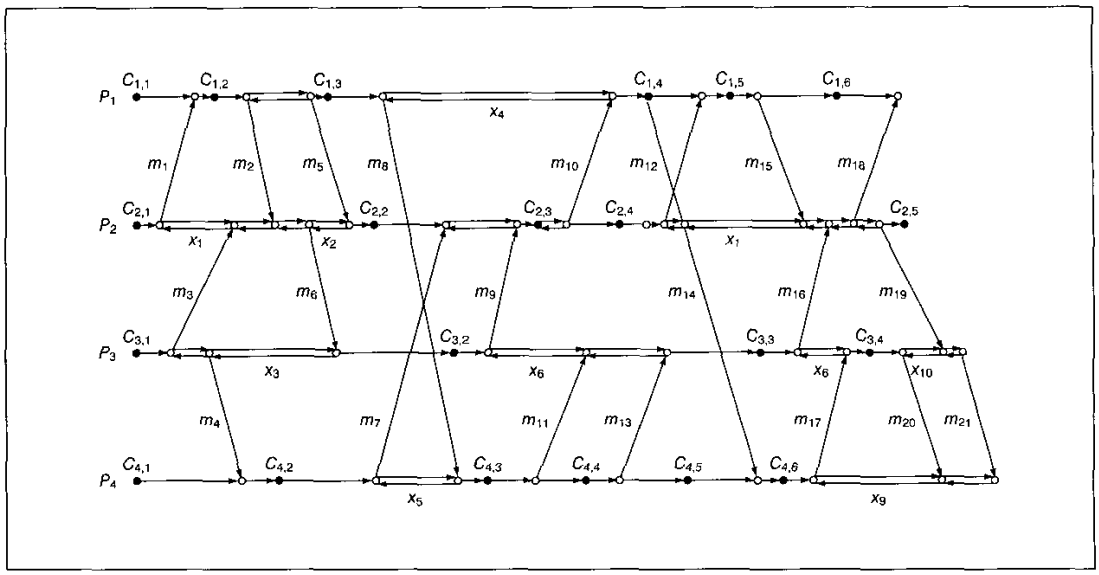

Fig. 7 *ZC*-digraph for the system of Fig. 6

vertices in V of G. The construction of $G(V, E)$ is formally defined as follows:

- $V = \{e_{i,j} | e_{i,j}$ is an event in process P_i, for all $1 \le i \le N\}$
- A directed edge $<e_{i,x}, e_{j,y}> \in E$ if any one of the following conditions is met:

– There exists a message m such that $e_{i,x} = send(m)$ and $e_{j,y} = deliver(m)$
– $i = j$ and $y = x + 1$
– $i = j$, $x = y + 1$, $e_{i,x} \notin \beta$ and $e_{j,y} \notin \beta$.

In the above definition, there are three conditions specified for the existence of the edges in G. The first two conditions are given to reflect message transmissions and causal relations in a process. The third one is used to address the existence of non-causal intervals. In the following treatment, we call edges of this class backward edges, indicating the fact that their directions are against causal directions.

Since no checkpoint may exist in a non-causal interval, no backward edges may direct to or from a checkpoint. Fig. 7 shows the *ZC*-digraph for the system of Fig. 6. In the Figure, notations P_1, P_2, P_3 and P_4 are retained for clarity purpose. Vertices that correspond to the basic checkpoints in β are illustrated with filled circles and are labelled in the Figure. Also labelled are edges that represent message transmissions.

The backward edges that correspond to the *PNCI*s in $\Gamma\beta$ are labelled by x_i, $1 \le i \le 10$. In a *ZC*-digraph, we define a *ZC*-cycle as a directed path from a checkpoint vertex to itself.

Consider adding a forced checkpoint to a *PNCI* in $\Gamma\beta$. Let this *PNCI* be the interval between $send(m_x)$ and $deliver(m_y)$. The *ZC*-digraph of the resulting system can be obtained by transforming that of $\Gamma\beta$ as follows: (1) add an extra vertex that corresponds to the added forced checkpoint; (2) remove the backward edge associated with the original *PNCI*, and (3) replace the original edge from $send(m_x)$ to $deliver(m_y)$ by

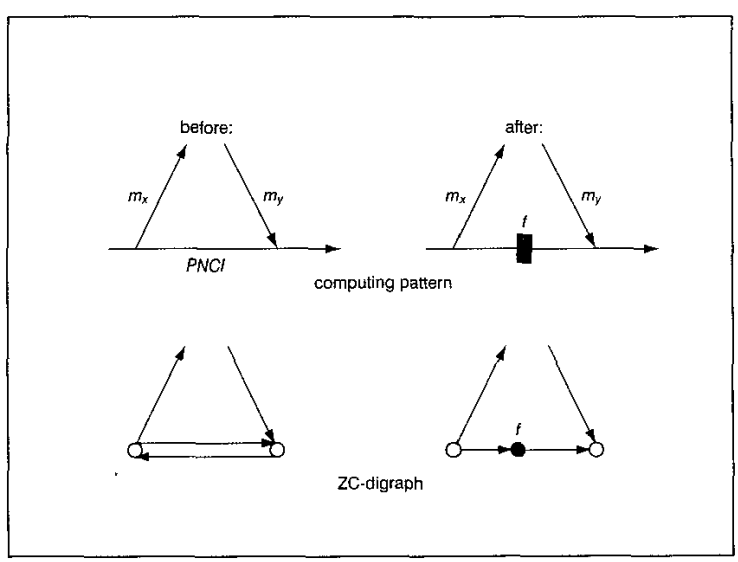

Fig. 8 ZC-digraphs before and after adding forced checkpoint f

two edges, one from *send*(m_x) to the forced checkpoint and another from the forced checkpoint to *deliver*(m_y). This transformation is illustrated in Fig. 8. The problem of locating the *PNCI*s for placing forced checkpoints so that the system may become *Z*-cycle free is converted to finding places in the corresponding *ZC*-digraph for performing the aforementioned operations so that the resulting *ZC*-digraph is free of *ZC*-cycles.

Problem reduction

There are several observations that are useful for reducing the complexity of the problem. Consider the *ZC*-digraph $G(V, E)$ of $\Gamma\beta$. There exists a *ZC*-cycle for a checkpoint vertex C if and only if the strongly connected component in G that contains C includes more than one vertex. In fact, a strongly connected component that contains a checkpoint vertex C consists exactly of all the vertices that are located on some *ZC*-cycles associated with C. For any checkpoint vertex for which no *ZC*-cycle exists, the strongly connected component that contains the vertex would be formed entirely by the checkpoint vertex itself.

Henceforth, we first decompose G into its strongly connected components. We ignore those strongly connected components that contain a single vertex or contain no checkpoint vertex. No *ZC*-cycles may possibly exist for vertices in these components. The rest of the strongly connected components are called *meaningful strongly connected components* (abbreviated as *MSCC*s).

There is at least one *ZC*-cycle associated with each checkpoint vertex in an *MSCC*. Strongly connected components of G can be found in $O(|V|^3)$ time.[4] The original problem is thus divided into several sub-problems, one for each of the *MSCC*s of G. A solution for the original problem is obtained by combining the solutions to all of the sub-problems. For example, we obtain three *MSCC*s for the *ZC*-digraph of Fig. 9. These *MSCC*s are illustrated in Fig. 9, in which they are identified as H_1, H_2 and H_3.

Consider an *MSCC* $H(V_h, E_h)$ of G. We are to find a set of backward edges in E_h, each corresponding to some *PNCI* of $\Gamma\beta$, whose removal makes H free of *ZC*-cycles. Obviously, it would be desirable to find the minimum number of such edges. This problem can be modelled as a *generalised weighted feedback edge* set problem, called SUBSET-FES,[3] which is stated as follows: 'Consider a directed graph with weighted edges. A cycle is considered *interesting* if it intersects a subset of *special* vertices and edges. Find a subset of edges with minimum total weight that intersects every interesting cycle in the graph.' Our problem may be transformed to a SUBSET-FES problem by setting the weights of all of the backward edges in E_h that correspond to *PNCI*s to one and the weights of the rest of the edges to infinity. All the checkpoint vertices in V_h are indicated as the special vertices and there is no special edge. In this case, the interesting cycles correspond to the *ZC*-cycles in H. A solution for this SUBSET-FES problem directly provides a solution for our problem. The general SUBSET-FES problem is NP-hard with only two special vertices. An approximation algorithm has been presented in Reference 9. Their polynomial-time approximation factor is $O(\log^2 k)$, where k is the number of special vertices (checkpoint vertices).

Although we resort to an approximation method to find a near-optimal solution, in general, optimal solutions may be obtained in polynomial time in many cases. If there is only one checkpoint vertex in V_h, we can find the optimal solution using max-flow min-cut algorithm. Moreover, many *MSCC*s can be characterised as

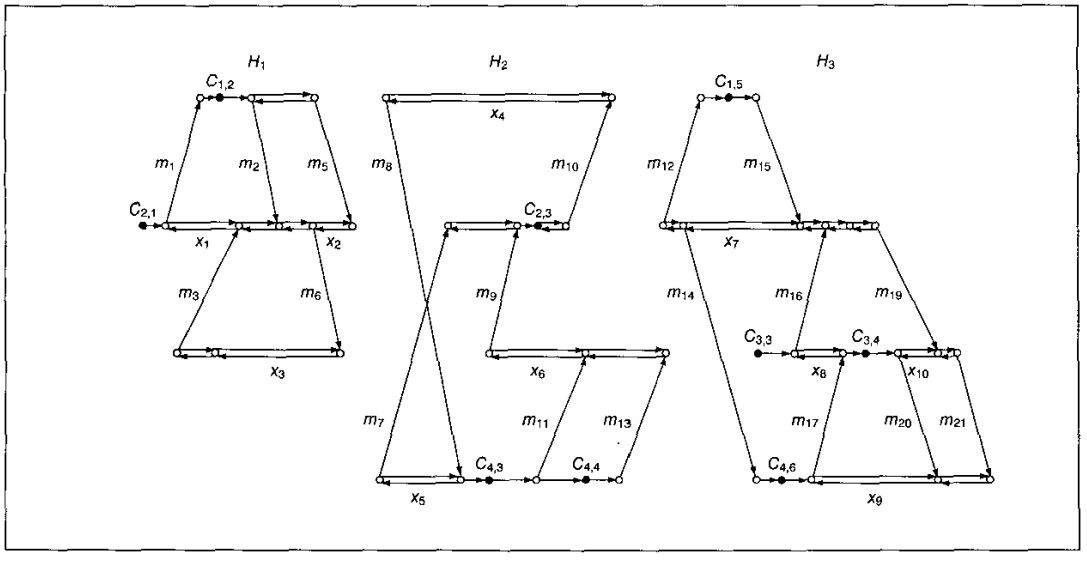

Fig. 9 *MSCC*s for the *ZC*-digraph of Fig. 7

reducible flow graphs.[10] Optimal solutions for reducible flow graphs may also be obtained in polynomial time by using Ramachandran's algorithm.[10] For example, in Fig. 9, the *MSCC* H_1 contains only one checkpoint vertex $C_{1,2}$ and *MSCC* H_2 is a reducible flow graph. The solutions for the *MSCCs* H_1, H_2 and H_3 of Fig. 9 are $\{x_1\}$, $\{x_4\}$ and $\{x_7, x_9\}$, respectively. All three solutions are optimal in this case.

Minimal solution set

The set of backward edges that are located in the previous step assures that their removal makes the *ZC*-digraph of the original system $\Gamma\beta$ free of *ZC*-cycles (with respect to the basic checkpoint vertices). Actually, each of these backward edges corresponds to a forced checkpoint to be inserted. However, the previous operation does not take the existence of *ZC*-cycles for the vertices that correspond to the forced checkpoints into consideration. We have to ensure that no new *ZC*-cycles may be created for the forced checkpoint vertices in the resulting *ZC*-digraph. Let E_b represent the set of backward edges located in the previous step for some *MSCC H*. As shown later, this property can be achieved, for *H*, as long as the set E_b of backward edges located is minimal in the sense that no proper subset of E_b would satisfy the requirement that the *ZC*-digraph of $\Gamma\beta$ remains free of *ZC*-cycles if the edges of the subset are removed. Apparently, if E_b is a minimum set for *H*, then E_b is minimal. Suppose that the approximation method[3] is used to find E_b; E_b is not guaranteed to be a minimal set. In this case, we may apply a greedy method to the edges in E_b as follows. First remove all edges in E_b from the *ZC*-digraph of $\Gamma\beta$. The resulting graph must be *ZC*-cycle free. We then take an edge, say e_b, from E_b, put the edge back to the graph and examine whether the graph remains *ZC*-cycle free. If it does, then delete e_b from E_b, and go on to the next edge. We perform this operation for all the edges in E_b, one by one. It is easy to see that E_b finally becomes a minimal set. Finding the transitive closure of the graph requires examining whether a graph is *ZC*-cycle free. Forced checkpoints are inserted according to E_b. We conclude this section by proving that no new *ZC*-cycle would be created for any added checkpoint vertices as long as the set E_b is minimal.

From our Theorem 3 (details of the theorem can be found in Reference 2) and the fact that no *ZC*-cycles exist for basic checkpoints, the resulting *ZC*-digraph, with forced checkpoints inserted according to our scheme, is *ZC*-cycle free. For the previous example of Fig. 9, the sets of backward edges located for the *MSCCs* H_1, H_2 and H_3 are minimum ones, and hence they are minimal sets. A total of four forced checkpoints should be added at the *PNCIs* that correspond to the edges labelled by x_1, x_4, x_7 and x_9. It is easy to see that the resulting system, which is shown in Fig. 10, is *Z*-cycle free.

Discussion

In Fig. 11, there are six processes, with some messages passed among them. The short orange bars indicate basic checkpoints and the short blue bars indicate forced checkpoints. Owing to the history of messages m_2, m_5 and m_3, there is a possibility for a *Z*-cycle to be formed.

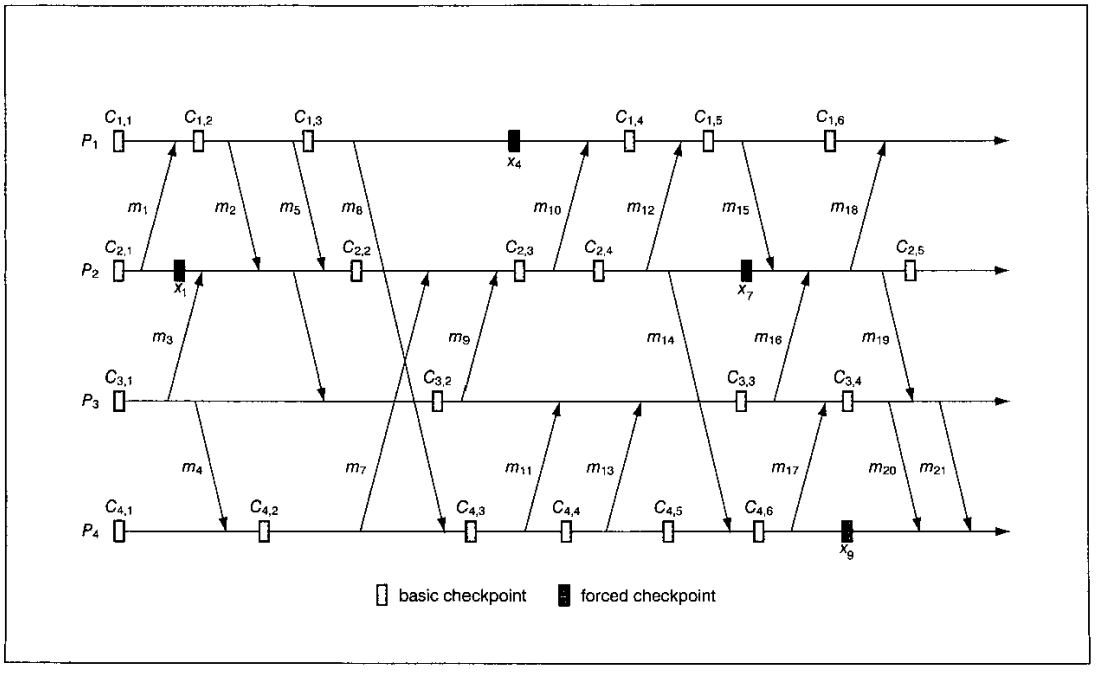

Fig. 10 Final system with forced checkpoints added to the original system of Fig. 6

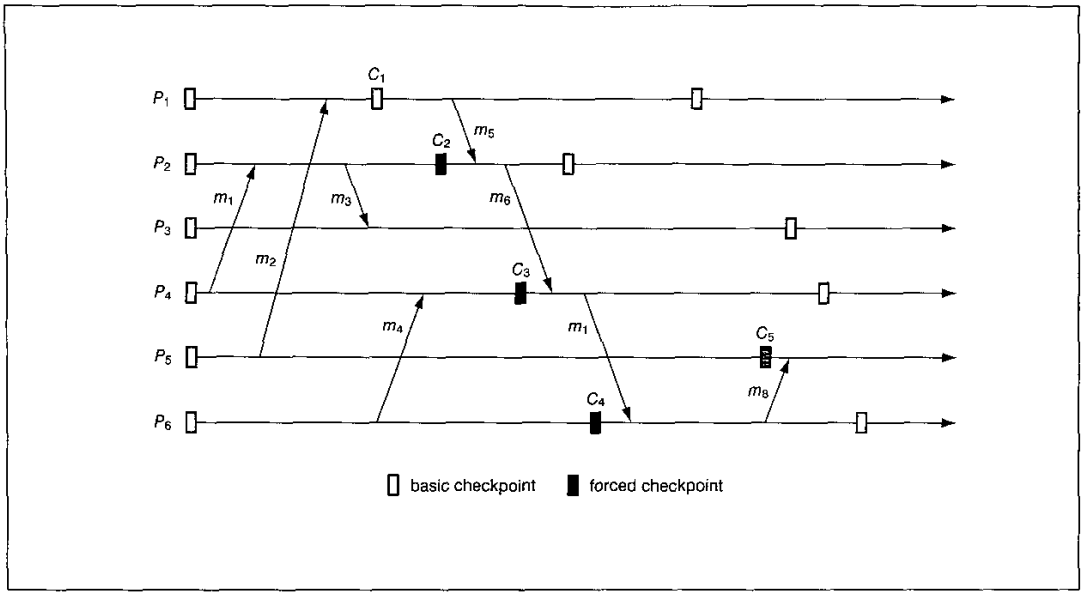

Fig. 11 Unneeded forced checkpoints

The forced checkpoint C_2 is taken because of the suspicion that the basic checkpoint C_1 will become useless according to some existing methods.[7] Unfortunately, it is easy to see that C_2 was in a Z-cycle (m_6 and m_2) if C_3 wasn't taken. Similarly, C_4 is triggered by C_3 since C_3 was also in a Z-cycle (m_4 and m_7) without C_4. In fact C_2, C_3 and C_4 are needless checkpoints in this system. Hence, many forced checkpoints not only may be triggered in such suspected situation, but also some of those unnecessary forced checkpoints may become useless checkpoints in the same conditions. The propagation of the needless forced checkpoints is attributed to the incomplete information for detecting Z-cycles. However, in Fig. 11, only the forced checkpoint C_5 is necessary. Apparently, it is desirable that the number of forced checkpoints can be made small so that the incurred overhead is reduced. Consequently, the number of forced checkpoints must be minimal only with our algorithm.

Conclusion

In this paper, we have presented a scheme for placing forced checkpoints in a distributed real-time embedded system so as to make all checkpoints useful for rollback recovery. Using the notion of *PNCI*, we are able to reduce the problem to identifying a set of *PNCIs* in the system where forced checkpoints should be inserted. An efficient algorithm has been presented in the paper. Although our algorithm offers a near-optimal solution, in general optimal solutions could be obtained in many cases using reduction techniques. Timing constraints are normally imposed upon real-time embedded systems. The addition of the forced checkpoints may possibly result in having some deadlines that are missed. In this case, basic checkpoints need to be re-scheduled. The efficiency of our algorithm is useful in this regard. Moreover, our scheme is also useful for debugging purposes.

References

1 ZUBERI, K. M., and SHIN, K. G.: 'Design and implementation of efficient message scheduling for controller area network', *IEEE TC*, February 2000, **49**, (2), pp. 182-188
2 CHIU, J.-F., and CHIU, G.-M.: 'Placing forced checkpoints in distributed real-time embedded systems', IEEE/IEE Real-Time Embedded Systems Workshop London, 3rd December 2001, available at: http://rtds.cs.tamu.edu/chiu.pdf
3 ELNOZAHY, E. N., JOHNSON, D. B., and WANG, Y. M.: 'A survey of rollback-recovery protocols in message-passing systems', Technical Report CMU-CS-96-181
4 HELARY, J. M., *et al.*: 'Communication-based prevention of useless checkpoints in distributed computations', *Distributed Computing*, 2000, **13**, (1) pp. 29-43
5 MANIVANNAN, D., and SINGHAL, M.: 'Quasi-synchronous checkpointing: models, characterization, and classification', *IEEE TPDS*, July 1999, **10**, (7), pp.703-717
6 CHANDY, K. M., and LAMPORT, L.: 'Distributed snapshots: determining global states of distributed systems', *Communication of ACM*, February 1985, **3**, pp. 63-75
7 ALVISI, L., *et al.*: 'An analysis of communication induced checkpointing', IEEE FTCS-99 pp. 242-249
8 NETZER, R. H. B., and XU, J.: 'Necessary and sufficient conditions for consistent global snapshots', *IEEE TPDS*, 1995, **6**, (2), pp.165-169
9 EVEN, G., NAOR, J., SCHIEBER, B., and SUDAN, M.: 'Approximating minimum feedback sets and multicuts in directed graphs', *Algorithmica*, 1998, **20**, pp. 151-174
10 RAMACHANDRAN, V.: 'Finding a minimum feedback arc set in reducible flow graphs', *Journal of Algorithms*, 1988, pp. 299-313

Jane-Ferng Chiu is with the Department of Electrical Engineering, National Taiwan University of Science and Technology, Taipei, Taiwan, jfchiu@totoro.ee.ntust.edu.tw. Ge-Ming Chiu is with the Department of Computer Engineering, University of the National Taiwan University of Science and Technology, Taipei, Taiwan, chiu@optimal.ee.ntust.edu