A New Roll-Forward Checkpointing / Recovery Mechanism for Cluster Federation

B. Gupta, S. Rahimi, and R. Ahmad

Department of Computer Science
Southern Illinois University
Carbondale, IL 62901-4511, USA

Summary
In this paper, we have addressed the complex problem of determining a recovery line for cluster federation and proposed an efficient checkpointing / recovery mechanism for it. The main objective of the proposed approach is to advance the recovery line in a cluster federation such that we can put a limit on the amount of rollback by the processes in all the clusters in case of failure(s) in the cluster federation; thereby in the worst case only limited domino effect is allowed in our work. In this approach, processes in different clusters are able to perform their responsibility independently and simultaneously. This inherent parallelism of the algorithm contributes to its speed of execution. We have shown that the proposed approach is superior to the existing works, because neither it suffers from any message storm, nor it takes any unnecessary checkpoints.

Keywords: *Cluster Computing, Checkpoints, Recovery*

1. Introduction

Cluster is a widely used term meaning independent computers combined into a unified system through software and networking. At the most fundamental level, when two or more computers are used together to solve a problem, it is considered a cluster. Clusters are typically used for high availability, for greater reliability or for high performance computing to provide greater computational power than a single computer can provide. Cluster computing environments have proved a cost-effective solution to many distributed computing problems by leveraging inexpensive hardware [1], [2], [9]. Processes in a cluster are often linked by a SAN (System Area Networks) while clusters are linked by LANs (Local Area Network) or WANs (Wide Area Networks) [1], [2]. A Cluster federation is a union of clusters, where each cluster contains a certain number of processes.

Checkpointing and rollback recovery [5] is one of the widely used techniques that allow systems to progress in spite of failure. The basic idea is to periodically record the system state as a checkpoint during normal system operation and, upon detection of faults, to restore one of the checkpoints and restart the system from there [5], [6]. Considering the characteristics of cluster federation architecture, different checkpointing mechanisms should be used within and between clusters.

A system is said to be consistent, if there is no message which is recorded in the state of its receiver but not recorded in the state of its sender [2], [5], [6]. Each cluster determines a consistent local checkpoint set that consists of one checkpoint from each process present in it. But this consistent local checkpoint set may not be consistent with the other clusters' consistent local checkpoint sets, because clusters interact through messages which result in dependencies between the clusters. Therefore, a collection of consistent local checkpoint sets does not necessarily produce a consistent federation level checkpoint set (i.e. a federation-wide recovery line) that consists of one checkpoint from each process present in the cluster federation. Consequently, rollback of one failed cluster may force rollback of the other clusters in order to maintain consistency of the cluster federation. In the worst case, consistency requirement may force the system to rollback to the initial state of the system, losing all the work performed before a failure. This uncontrolled propagation of rollback is termed as domino-effect [10]. There is, therefore, a need to have a second level of cluster-wide checkpointing algorithm that prevents the rollbacks in reaching a consistent federation level checkpoint set (i.e., a federation level recovery line). It means that, this algorithm has to ensure that the collection of all consistent local checkpoints sets always produces a consistent federation level checkpoint set.

<u>Problem Formulation:</u> A significant amount of research exists in the literature for recovery in distributed systems [3]-[8]. However, very few works [1], [2] have been reported for handling the problem of recovery in cluster federation computing so far. In this work, we address this problem in cluster federations. Our objective is to advance the recovery line in a cluster federation such that we can put a limit on the amount of rollback by the processes in all the clusters in case of failure(s) in the cluster federation; thereby in the worst case only limited domino effect is allowed in our work. The key concept used in this work is that it is the sending process that makes sure that none of its sent messages can remain an orphan. So any process that receives some messages has no responsibility to make the received messages non orphan. This is why the decision taken by a process P about whether to take a

Manuscript received November 10, 2006.
Manuscript revised November 25, 2006.

checkpoint or not, does not affect any other process' decision about the same. It results in all processes taking their respective check pointing decisions independently and simultaneously without the need for sharing of any information.

The system architecture considered in this paper is similar to that considered in [2], where multiple coordinated checkpointing subsystems are connected with a single independent checkpointing subsystem. An example of such architecture is shown in Fig. 1. In this figure each coordinated subsystem represents a group of smaller coordinated subsystems which have frequent message exchanges among them and multiple independent subsystems are combined into one larger independent subsystem. Also note that according to this architecture the coordinated subsystems CO^1, CO^2, CO^3, and CO^4 interact with each other very infrequently and only via the larger independent subsystem. Thus, in the figure we have only one independent subsystem and multiple coordinated subsystems. From now on we will call the coordinated checkpointing subsystem as coordinated cluster subsystem and independent checkpointing subsystem as independent cluster subsystem.

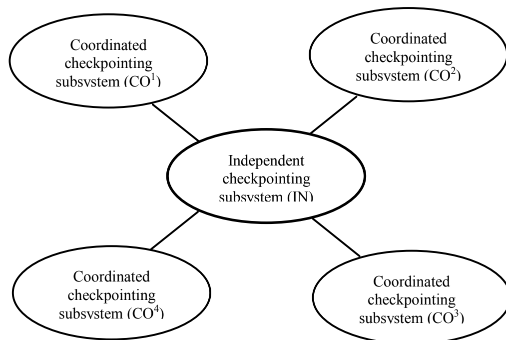

Fig.1 System architecture

This paper is organized as follows. In Section 2 we have stated the necessary data structures used in our algorithm. Section 3 describes the working principle of the algorithm. In Section 4, we have stated some simple observations necessary to design the algorithm. In Section 5, we have presented our roll-forward federation wide checkpointing algorithm along with a comparison with existing works. Finally, Section 6 concludes this paper.

2. Relevant Data Structures

Before we state the relevant data structures and their use we need to define the following:

Regular checkpoint: Inside the coordinated cluster subsystem, processes use the single phase non-blocking checkpointing protocol reported in [3] to take checkpoints. Inside an independent cluster subsystem, processes take checkpoints independently according to their respective time periods.

Forced checkpoint: Besides the regular checkpoints, a cluster subsystem may also have to take forced checkpoints. The conditions for taking a forced checkpoint are explained in detail in Section 3.

In our approach, communication between two clusters means communication between two processes belonging to the two clusters respectively. Failure of a cluster means failure of its one or more processes.

Let us consider a set of n coordinated cluster subsystems, $\{CO^1, CO^2, \ldots, CO^n\}$ and an independent cluster subsystem denoted by IN. Each cluster subsystem consists of several processes. We use the following notations to represent a cluster subsystem and its processes. The jth process of ith coordinated cluster subsystem CO^i is denoted as p_j^i, and the kth process of the independent cluster subsystem IN is denoted as p_k. Each process p_k in independent cluster subsystem (IN) maintains a flag c_k (Boolean). The flag is initially set at zero. Flag c_k is set at 1 when process p_k sends its first intercluster or intracluster application message after its latest checkpoint. Flag c_k is reset to 0 again after process p_k takes a checkpoint. Note that the flag c_k of process p_k is set to 1 only once independent of how many messages process p_k sends after its latest checkpoint. In addition, process p_k maintains an integer variable S_k, which is initially set at 0 and is incremented by 1 each time the roll-forward federation wide checkpointing algorithm is invoked. When process p_k of the independent cluster subsystem takes its xth checkpoint, then this checkpoint is represented as $CN_{k(x)}$.

Similarly, each process p_j^i in CO^i (for i =1 to n) also maintains a boolean flag c_j^i which is initially set at zero. Flag c_j^i is set at 1 only when process p_j^i sends its first intercluster or intracluster application message after its latest checkpoint. It is reset to 0 again after process p_j^i takes a checkpoint. Note that the flag c_j^i of process p_j^i is set to 1 only once independent of how many messages process p_j^i sends after its latest checkpoint. In addition, process p_j^i maintains an integer variable S_j^i, which is initially set at 0 and is incremented by 1 each time the roll-forward federation wide checkpointing algorithm is invoked. When process p_j^i of the coordinated cluster subsystem takes its xth checkpoint then this checkpoint is represented as $CN_{j(x)}^i$.

3. Working Principle

In this paper, we consider the complex problem of determining a federation wide recovery line (i.e. globally consistent checkpoints for all process of all clusters). This federation wide recovery line can be defined as a set of checkpoints, one from each process, such that these checkpoints are mutually consistent; that is, there is no orphan message in the system with respect to this set of

checkpoints. Our main objective is to make this recovery algorithm roll-forward in nature [7], [8], that is, we will allow only limited domino-effect. In our approach, processes in a coordinated cluster subsystem takes their respective local checkpoints periodically following the checkpointing protocol reported in [3]; where as processes in the independent cluster subsystem take their respective local checkpoints independently (asynchronously) [10] according to their respective time periods. However, as mentioned earlier that the consistent local checkpoint set of one cluster may not be consistent with the other clusters' consistent local checkpoint sets, because clusters interact through messages which result in dependencies between the clusters. Hence, to determine a federation wide recovery line (i.e. the federation wide globally consistent checkpoints) we design a roll-forward federation wide checkpointing algorithm which is executed periodically (say the time period is T) by a process in the independent cluster subsystem. Now if a failure occurs in any cluster (s), then after the system recovers from the failure, all processes of all the clusters can restart from their respective federation wide globally consistent checkpoints as determined by the last execution of the roll-forward algorithm. Therefore, in the worst case, the effect of the domino phenomenon is limited by the time period T. Hence we call it roll forward recovery. Note that the recovery becomes as simple as in the synchronous approach [10]. Also, it is understood that the time period T used by the proposed algorithm is much larger than the individual time periods used by any process in the independent cluster system. It is also much larger than the time periods used in the coordinated cluster subsystems.

We now briefly outline the situations about when a process needs to take a checkpoint while the algorithm is executed. Assume that process p_k of cluster IN initiates periodically the roll-forward federation wide checkpointing algorithm. It first checks its local flag c_k. If it is set to 1, then it takes a forced checkpoint and increments its integer variable S_k by 1; otherwise, it only increments S_k by 1. It then sends a control message M_c piggybacked with its updated S_k value to all processes in its subsystem as well as to all coordinated subsystems. A process p_q (or p_j^i) in IN (or CO^i) on receiving this control message M_c checks its local flag c_q (or c_j^i). If it is set to 1 then it takes a forced checkpoint and updates its integer variable S_q (S_j^i); if it is set to 0 then it skips taking a forced checkpoint, but it updates its local integer variable S_q (S_j^i) with S_k which is piggybacked with the control message M_c sent by the process p_k. If process p_q (p_j^i) skips taking forced checkpoint then its latest checkpoint will be its federation wide globally consistent checkpoint; otherwise the forced checkpoint it takes becomes a federation wide globally consistent checkpoint.

Any process p_k (p_j^i) after implementing its decision whether to take a forced checkpoint or not, must piggyback its S_k (S_j^i) value with its first application which is sent to any other process before the next invocation of the algorithm. However, the process p_k (p_j^i) does not need to piggyback S_k (S_j^i) value if it sends any other message to the same process in the system before the next invocation of the proposed algorithm [3]. The advantage of sending this piggybacked application message is that, if a process p_q (or p_r^i) of cluster IN (or CO^i) receives this application message before the control message M_c, it compares its S_q (S_r^i) value with the piggybacked value of the received application message. If S_k (S_j^i) value is greater than S_q (S_r^i) then process p_q (p_r^i) comes to know that the algorithm has already been initiated, but it has not yet received the control message from the initiator process p_k. Therefore instead of waiting for control message M_c to arrive, the process p_q (p_r^i) of cluster IN (CO^i) checks its flag c_q (c_r^i). If it is set to 1, it takes a forced checkpoint, updates its S_q (S_r^i) value with the piggybacked S_k (S_j^i) value, and then processes the received application message. On the other hand, if flag c_q (c_r^i) value is 0, it just updates its S_q (S_r^i) value and then processes the application message.

<u>*An illustration*</u>: Fig. 2 shows a sample execution of our algorithm with different situations. Without any loss of generality let us assume that process p_2 of cluster IN is the initiator of the algorithm to determine a consistent global checkpoint of the cluster federation. Process p_2 first checks its flag c_2 and finds that it is set to 1 because it has sent two application messages m_2 and m_4 since its last checkpoint $CN_{2(2)}$. So process p_2 takes a forced checkpoint $CN_{2(3)}$, updates its integer variable S_2 to 1 and resets its flag c_2 to 0. It now sends the control message M_c piggybacked with S_2 value to processes in its cluster and to other clusters. Process p_2^1 of cluster CO^1 on receiving the control message M_c takes the forced checkpoint $CN_{2(3)}^1$ because flag c_2^1 is set to 1and updates its integer variable S_2^1 to 1.

Now consider the next situation. Processes p_1^2 and p_2^2 of cluster CO^2 on receiving this control message finds that their respective flags c_1^2, c_2^2 are set to 0. So, they skip taking forced checkpoints, but update their S_1^2 and S_2^2 values to 1. So federation wide globally consistent checkpoints for processes p_1^2 and p_2^2 are $CN_{1(2)}^2$, $CN_{2(2)}^2$.

Now assume that process p_2 of cluster IN sends an application message m_7 piggybacked with S_2=1 to process p_3^1 after its checkpoint $CN_{2(3)}$. Process P_3^1 of cluster CO^1 receives this application message before the control message M_c. Process p_3^1 learns from the piggybacked application message <m_7, S_2=1> that the algorithm to determine the federation wide recovery line has already

been invoked, because its own S_3^1 value is 0. So instead of waiting to receive the control message M_c, process p_3^1 checks its flag c_3^1, which is set to 1. So it takes a forced checkpoint $CN_{3(3)}^1$, increments its S_3^1 value to 1, and then processes the application message m_7. Process p_3^1 receives the control message M_c after it has already implemented its decision whether to take a forced checkpoint or not. So it just neglects this message.

Now assume the following situation. Process p_2^2 of cluster CO^2 sends a piggybacked application message $<m_8, S_2^2=1>$ to process p_1 of cluster IN after its latest checkpoint $CN_{2(2)}^2$, which is received by process p_1 before receiving the control message M_c. Process p_1 learns from the piggybacked application message $<m_8, S_2^2=1>$ that the algorithm to determine the federation wide recovery line has already been invoked, because its own S_1 value is 0. So process p_1 checks its flag c_1, which is set to 1 and takes the forced checkpoint $CN_{1(3)}$, increments its S_1 value to 1 and then processes the application message m_8.

Next consider that process p_2^1 of cluster CO^1 sends a piggybacked application message $<m_9, S_2^1=1>$ to process p_1^1 which belongs to its own cluster after its latest checkpoint $CN_{2(3)}^1$. Process p_1^1 has not yet received the control message M_c. As $S_2^1 > S_1^1$ process p_1^1 knows that the algorithm to determine the federation wide recovery line has already been invoked. So instead of waiting for this control message, process p_1^1 checks its flag c_1^1, which is set to 1 and takes a forced checkpoint $CN_{1(3)}^1$, increments its S_1^1 value to 1 and then processes the application message m_9. Therefore, federation wide globally consistent checkpoints for the given example are $CN_{1(3)}^1$, $CN_{2(3)}^1$, $CN_{3(3)}^1$, $CN_{1(3)}$, $CN_{2(3)}$, $CN_{1(2)}^2$ and $CN_{2(2)}^2$.

Now, suppose if process p_1 of cluster IN fails, it rolls back to its federation wide globally consistent checkpoint $CN_{1(3)}$ and initiates the recovery algorithm by broadcasting a control message to all processes in its cluster and to other clusters telling them to rollback to their respective latest federation wide globally consistent checkpoints. Processes on receiving this control message rollback to their respective latest federation wide globally consistent checkpoints and restart their computations.

4. Relevant Observations

Below, we state some simple but important observations used in the proposed algorithm.

Theorem 1: If at any given time t, if c_k (c_j^i) = 0 for process p_k (p_j^i) with $CN_{k(x+1)}$ ($CN_{j(x+1)}^i$) being its latest checkpoint, none of the messages sent by p_k (p_j^i) remains an orphan at time t.

Proof: Flag c_k (c_j^i) can have the value 1 between two successive checkpoints, say $CN_{k(x)}$ ($CN_{j(x)}^i$) and $CN_{k(x+1)}$ ($CN_{j(x+1)}^i$), of a process p_k (p_j^i) if and only if process p_k (p_j^i) has sent at least one message m_n between these two checkpoints. It can also be 1 if p_k (p_j^i) has sent at least a message after taking its latest checkpoint. It is reset to 0 at each checkpoint. On the other hand, it will have the value 0 either between two successive checkpoints, say $CN_{k(x)}$ ($CN_{j(x)}^i$) and $CN_{k(x+1)}$ ($CN_{j(x+1)}^i$), if process p_k (p_j^i) has not sent any message between these checkpoints, or p_k (p_j^i) has not sent any message after its latest checkpoint. Therefore, c_k (c_j^i) = 0 at time t means either of the following two: (1) c_k (c_j^i) = 0 at $CN_{k(x+1)}$ ($CN_{j(x+1)}^i$) and this checkpoint has been taken at time t. It means that any message m_n sent by p_k (p_j^i) (if any) to any other process between $CN_{k(x)}$ ($CN_{j(x)}^i$) and $CN_{k(x+1)}$ ($CN_{j(x+1)}^i$) must have been recorded by the sending process p_k (p_j^i) at the checkpoint $CN_{k(x+1)}$ ($CN_{j(x+1)}^i$). So message m_n can not be an orphan; (2) c_k (c_j^i) = 0 at time t and p_k (p_j^i) has taken its latest checkpoint $CN_{k(x+1)}$ ($CN_{j(x+1)}^i$) before time t. It means that process p_k (p_j^i) has not sent any message after its latest checkpoint till time t. Hence at time t, there does not exit any orphan message sent by p_k (p_j^i) after its latest checkpoint. ■

Theorem 2: If flag c_k (c_j^i) = 1, where $CN_{k(x)}$ ($CN_{j(x)}^i$) is the latest checkpoint of process p_k (p_j^i), some message(s) sent by p_k (p_j^i) to other processes may become an orphan.

Proof: The flag c_k (c_j^i) is reset to 0 at every checkpoint. It can have the value 1 only between two successive checkpoints of any process p_k (p_j^i) if and only if process p_k (p_j^i) sends at least one message m_n between the checkpoints. Therefore, c_k (c_j^i) = 1 means that p_k (p_j^i) is yet to take its next checkpoint following $CN_{k(x)}$ ($CN_{j(x)}^i$). Therefore, the sending event of the message(s) sent by p_k (p_j^i) after its latest checkpoint $CN_{k(x)}$ ($CN_{j(x)}^i$) is not yet recorded. Now if some other process receives one or more of these messages sent by p_k (p_j^i) and then takes its latest checkpoint before process p_k (p_j^i) takes its next checkpoint $CN_{k(x+1)}$ ($CN_{j(x+1)}^i$), then this received message(s) will become orphan. Hence the proof follows. ■

Theorem 3: If S_k (= x) > S_j^i (= x-1), the piggybacked application message $< m_n, S_k >$ sent by process p_k and received by a process p_j^i can never be an orphan.

Proof: When S_k (= x) > S_j^i (= x-1), process p_j^i knows that the xth execution of the roll-forward federation wide checkpointing algorithm has already begun and so very soon it will also receive the message M_c from the initiator process associated with this execution. So instead of waiting for M_c to arrive, it decides if it needs to take a checkpoint and implements its decision, and then processes the message m_n. This means that the receiving event of the message m_n is not recorded at the receiver. Therefore, message m_n can never be an orphan. ■

Theorem 4: If $S_k = S_j^i = x$, process p_j^i can immediately process the received piggybacked application message <

m_n, S_k > and this application message sent by process p_k can never be an orphan.

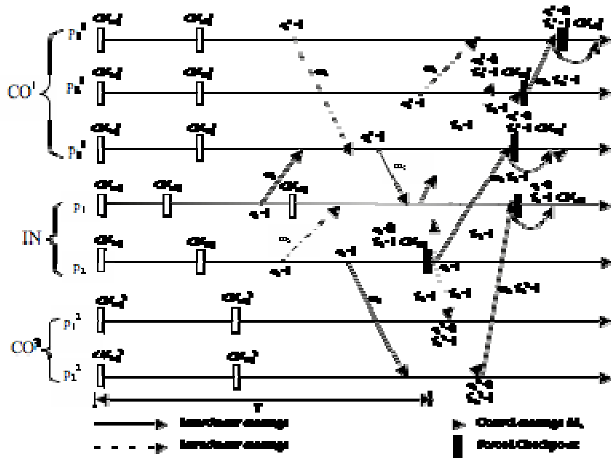

Fig.2 Process P_1 initializes the roll-forward algorithm

Proof: If $S_k = S_j^i = x$, like process p_k, process p_j^i has already received the control message M_c associated with the latest execution (x^{th}) of the roll-forward federation wide checkpointing algorithm and has taken its checkpointing decision and has already implemented that decision. Therefore, process p_j^i now processes the message m_n. It ensures that message m_n can never be an orphan, because both the sending and the receiving events of message m_n have not been recoded yet by the sender p_k and the receiver p_j^i respectively. ■

5. Algorithm Recovery

We now state the algorithm. It is a single phase algorithm because the initiator process interacts with the other processes only once via the control message M_c. Also a process after its participation in the algorithm does not wait for the algorithm to terminate before resuming its normal operation. That is, it is a non-blocking algorithm. The responsibilities of the initiator process and any other process are stated separately.

<u>Process p_k in cluster IN:</u>

if p_k is the initiator
if $c_k = 1$ */* it has sent at least one message after its latest checkpoint */*
takes a forced checkpoint;
$c_k = 0$;
$S_k = S_k + 1$;
sends <M_c, S_k> to all processes in cluster IN as well as to all other cluster subsystems;
continues its normal operation;
else
$S_k = S_k + 1$;
sends <M_c, S_k> to all processes in cluster IN as well as to all other cluster subsystems; continues its normal operation;
else if p_k receives < M_c, S_q > */* p_q in cluster IN is the initiator*/*
if $c_k = 1$
takes a forced checkpoint;
$c_k = 0$;
$S_k = S_k + 1$;
continues its normal operation;
else
$S_k = S_k + 1$;
continues its normal operation;
else if p_k receives < m_n, S_q > && p_k has not yet received < M_c, S_q >
if $S_k < S_q$
if $c_k = 1$
takes a forced checkpoint without waiting for <M_c, S_q>;
$c_k = 0$;
$S_k = S_k + 1$;
processes the received message m_n and ignores M_c when received later;
else
$S_k = S_k + 1$;
processes the received message m_n and ignores M_c when received later;
else
processes the received message m_n and ignores M_c when received later;
else if p_k receives < m_n, S_j^i > && p_k has not yet received <M_c, S_q>
if $S_k < S_j^i$
if $c_k = 1$
takes a forced checkpoint with out waiting for <M_c, S_q>;
$c_k = 0$;
$S_k = S_k + 1$;
processes the received message m_n and ignores M_c when received later;
else
$S_k = S_k + 1$;
processes the received message m_n and ignores M_c when received later;
else
processes the received message m_n and ignores M_c when received later;
else
continues its normal operation;

<u>Process p_j^i in cluster CO^i (for i = 1 to n):</u>

if p_j^i receives < M_c, S_k >
if $c_j^i = 1$
takes a forced checkpoint;
$c_j^i = 0$;
$S_j^i = S_j^i + 1$;
continues its normal operation;
else
$S_j^i = S_j^i + 1$;
continues its normal operation;
else if p_j^i receives < m_n, S_k > && p_j^i has not yet received <M_c, S_k>
if $S_j^i < S_k$
if $c_j^i = 1$
takes a forced checkpoint with out waiting for <M_c, S_k>;
$c_j^i = 0$;
$S_j^i = S_j^i + 1$;
processes the received message m_n and ignores M_c when received later;
else
$S_j^i = S_j^i + 1$;
processes the received message m_n and ignores M_c when received later;
else
processes the received message m_n and ignores M_c when received later;

else if p_j^i receives $< m_n, S_r^i >$ && p_j^i has not yet received $<M_c, S_k>$
  if $S_j^i < S_r^i$
    if $c_j^i = 1$
      takes a forced checkpoint with out waiting for $<M_c, S_k>$;
      $c_j^i = 0$;
      $S_j^i = S_j^i + 1$;
      processes the received message m_n and ignores M_c when received later;
    else
      $S_j^i = S_j^i + 1$;
      processes the received message m_n and ignores M_c when received later;
  else
    processes the received message m_n and ignores M_c when received later;
else
  continues its normal operation;

Proof of Correctness: Each process p_k of cluster IN in the first 'if else' and 'else if' blocks of its pseudo code, decides whether to take a checkpoint based on the value of its flag c_k. If it has to take a checkpoint, it resets c_k to 0. Therefore, in other words, each process p_k of cluster IN makes sure using the logic of Theorem 1 that none of the messages, if any, it has sent since its last checkpoint can be an orphan. On the other hand, if p_k does not take a checkpoint, it means that it has not sent any message since its previous checkpoint. In the second and third 'else if' blocks each process p_k of cluster IN follows the logic of Theorems 3 and 4, which ever is appropriate for a particular situation so that any application message received by p_k before it receives the control message M_c can not be an orphan. Besides none of its sent messages, if any, since its last checkpoint can be an orphan as well (following the logic of Theorems 1 and 2).

Each process p_j^i of cluster CO^i in the first 'if else' block of its pseudo code, decides whether to take a checkpoint based on the value of its flag c_j^i. If it has to take a checkpoint, it resets c_j^i to 0. Therefore, in other words, each process p_j^i of cluster CO^i makes sure using the logic of Theorem 1 that none of the messages, if any, it has sent since its last checkpoint can be an orphan. On the other hand, if p_j^i does not take a forced checkpoint, it means that it has not sent any message since its previous checkpoint. In the first and second 'else if' blocks each process p_j^i of cluster CO^i follows the logic of Theorems 3 and 4, which ever is appropriate for a particular situation so that any application message received by p_j^i before it receives the control message M_c can not be an orphan.

Since Theorems 1, 2, 3, and 4 guarantee that no sent or received messages by any process p_k (p_j^i) of any cluster since its previous checkpoint can be an orphan and since it is true for all participating clusters, therefore, the algorithm guarantees that the latest checkpoints taken during the current execution of the algorithm and the previous checkpoints (if any) of those processes which did not need to take forced checkpoints during the current execution of the algorithm are globally consistent checkpoints. ■

5.1 Advantages of the proposed algorithm

The algorithm offers the following important advantages. In our work only those processes that have sent some message(s) after their last checkpoints, take checkpoints during checkpointing; thereby reducing the number of forced checkpoints to be taken. It also helps processes of different clusters to take their respective checkpointing decisions independently and simultaneously without the need for sharing of any information. This inherent parallelism contributes to the speed of execution of the algorithm. Also, the proposed algorithm is a single phase algorithm with only one control message (M_c) and very simple data structures are maintained by the processes. Another advantage of the proposed algorithm is that it is non-blocking which means that application processes are not suspended during checkpointing. This single phase non-blocking nature of the algorithm definitely contributes to its speed of execution. Finally, the recovery is as simple as in the synchronous approach.

5.2 Comparisons

Comparison with [1]: In [1], a cluster takes two types of checkpoints; processes inside a cluster take checkpoints synchronously and a cluster takes a communication induced checkpoint whenever it receives an intercluster application message. Each cluster maintains a sequence number (SN). SN is incremented each time a cluster level checkpoint is committed. Each cluster maintains a DDV (Direct dependency vector) with a size equal to the number of clusters in the cluster federation. Whenever a cluster (i.e. a process in it) fails, after recovery it broadcasts an alert message with the SN of the failed cluster. This alert message triggers the next iteration of the algorithm. All other clusters, on receiving this alert message decide if they need to roll back by checking the corresponding entries in the DDV vectors. Each time there is a rollback, a new iteration starts. The rolled back clusters further broadcast the alert messages with their SN. This algorithm has the following advantage; simultaneous execution of the algorithm by all participating clusters contributes to its speed of execution. Our proposed algorithm also offers similar advantage.

However, the main drawback of the algorithm is that if we consider a particular message pattern where all the clusters have to roll back except the failed cluster, then all the clusters have to send alert messages to every other cluster. This results in a message storm. But in our approach when a process of a cluster fails it broadcasts just one control message.

In the algorithm [1], whenever a coordinated cluster subsystem of size n has to take a consistent local

checkpoint set, it requires 3*(n-1) control messages as it follows a three phase synchronous approach [10]. However in our approach, number of control messages required to take a consistent local checkpoint set is only (n-1) as our approach follows the single-phase non blocking checkpointing protocol [3].

Comparison with [2]: In [2], the authors have addressed the need of integrating independent and coordinated checkpointing schemes for applications running in a hybrid distributed environment containing multiple heterogeneous subsystems. This algorithm mainly works on two rules. Rule 1 states that, independent checkpoint subsystem takes a new coordinated checkpoint set if it sends an intercluster application message. Rule 2 states that, a process P_i of independent checkpointing subsystem takes a new independent checkpoint before processing an already received intercluster application message, if P_i has sent any intracluster application message after taking its last checkpoint. So, if the independent checkpointing subsystem has sent k number of intercluster application messages in a time period T, then it has to take k number of coordinated checkpoint sets besides the regular local checkpoints taken asynchronously by its processes. In our approach, if we consider the same situation, a process in the independent cluster subsystem (not all processes in the subsystem) has to take only one forced checkpoint, if at all needed, besides the regular local checkpoints taken asynchronously by processes of the independent system. So we reduce drastically the number of checkpoints to be taken by the independent cluster subsystem.

6. Conclusions

In this paper, we have presented a simple non-blocking roll-forward checkpointing / recovery mechanism for cluster federation. The effect of domino phenomenon is limited by the time interval between successive invocations of the algorithm and recovery is as simple as that in the synchronous approach. The noteworthy point of the presented approach is that a process receiving a message does not need to worry whether the received message may become an orphan or not. It is the responsibility of the sender of the message to make it non-orphan. Thus, processes in different clusters are able to perform their responsibility independently and simultaneously by just testing their local flags. This inherent parallelism of the algorithm contributes to its speed of execution. Our approach also reduces the number of forced checkpoints to be taken by forcing only those processes which have sent some message(s) after their last checkpoints. We have shown that the proposed approach is superior to the existing works, because neither it suffers from any message storm, nor it takes any unnecessary checkpoints.

References

[1] S. Monnet, C. Morin, R. Badrinath, "Hybrid Checkpointing for Parallel Applications in cluster Federations", In 4th IEEE/ACM International Symposium on Cluster Computing and the Grid, Chicago, IL, USA, pp 773-782, April 2004.

[2] J. Cao, Y. Chen, K. Zhang and Y. He, "Checkpointing in Hybrid Distributed Systems", Proc. of the 7th International Symposium on Parallel Architectures, Algorithms and Networks (ISPAN'04), pp 136-141, Hong Kong, China, May 2004.

[3] B. Gupta, S.Rahimi, R. A. Rias, and G. Bangalore, "A Low-Overhead Non-Blocking Checkpointing Algorithm for Mobile Computing Environment", Springer Verlag Lecture Notes in Computer Science, vol 3947, pp 597-608, 2006.

[4] B. Gupta, A. Thakre and D. Chhillar, "A Fast and Efficient Recovery Scheme for Distributed Programs", Proc. ISCA 20th Intl. Conf. Computers and their applications, New Orleans, pp. 459-464, March 2005.

[5] R. Koo and S. Toueg, "Checkpointing and Rollback-Recovery for Distributed Systems", IEEE trans. Software Engineering, vol. SE-13, no. 1, pp.23-31, Jan 1987.

[6] Y. Wang, "Consistent Global Checkpoints that contain a Given Set of Local Checkpoints", IEEE trans. Computers, vol. 46, no. 4, pp. 456-468, April 1997.

[7] B. Gupta, S.K. Banerjee and B. Liu, "Design of new roll-forward recovery approach for distributed systems", IEE Proc. Computers and Digital Techniques, vol. 149, issue 3, pp. 105-112, May 2002.

[8] D.K. Pradhan And N.H. Vaidya, "Roll-Forward Check pointing Scheme: A Novel Fault-Tolerent Architecture", IEEE Transactions on Computers, vol. 43, no.10, pp. 1163-1174, 1994.

[9] Xin Qi , G. Parmer , R. West, "An efficient end-host architecture for cluster communication", Proc. 2004 IEEE Intl. Conf. on Cluster Computing, San Diego, California, pp.83-92, September 20-23, 2004

[10] Mukesh Singhal, Niranjan G. Shivaratri, *Advanced Concepts in Operating Systems,* McGraw-Hill, Inc., 1994.

Bidyut Gupta received his PhD in Computer Science and his MTech degree in Electronics Engineering from the University of Calcutta, India. Currently, he is a professor of computer science and the graduate director for Computer Science department at the Southern Illinois University Carbondale.

Shahram Rahimi received his PhD in Scientific Computing from the University of Southern Mississippi in 2002, and his BS degree from National University of Iran (Tehran) in 1992. Currently, he is an assistant professor at Southern Illinois University and the Editor-in-Chief of the International Journal of Computational Intelligence Theory and Practice.