Journal of Computing and Information Technology - CIT 19, 2011, 2, 127–139
doi:10.2498/cit.1001706

A Recovery Scheme for Cluster Federations Using Sender-based Message Logging

Bidyut Gupta, Ruslan Nikolaev and Raja Chirra

Department of Computer Science, Southern Illinois University, Carbondale, Illinois, USA

A cluster federation is a union of clusters and is heterogeneous. Each cluster contains a certain number of processes. An application running in such a computing environment is divided into communicating modules so that these modules can run on different clusters. To achieve fault-tolerance different clusters may employ different check pointing schemes. For example, some may use coordinated schemes, while some other may use communication-induced schemes. It may complicate the recovery process. In this paper, we have addressed the complex problem of recovery for cluster computing environment. The proposed approach handles both inter cluster orphan and lost messages unlike the existing works in this area. We first propose an algorithm to determine a recovery line so that there does not exist any inter cluster orphan message between any pair of the cluster level check points belonging to the recovery line. The main feature of the proposed algorithm is that it can be executed simultaneously by all clusters in the cluster federation. Next we apply the sender-based message logging idea to effectively handle all inter cluster lost messages to ensure correctness of computation.

Keywords: cluster federation, cluster level, checkpoint, recovery

1. Introduction

Cluster federation is a union of clusters and is heterogeneous. Each cluster contains a certain number of processes (nodes). Nodes in a cluster are often linked by a system area network and clusters are linked by local area networks or even by wide area networks [2]. An Application running in such a computing environment is divided into communicating modules so that these modules can run on different clusters. Clusters are usually deployed to improve speed over that provided by a single computer, while typically being much more cost-effective than single computers of comparable speed or reliability [1], [2], [9]. Some of the other reasons can be security, hardware/software constraints, or because the applications may need a very large number of nodes. Code coupling applications are usually run on such architectures. A cluster federation can also be viewed as follows. With the availability of large-scale global computing systems (grid computing, web services as examples) a computing system may consist of many geographically dispersed heterogeneous subsystems (clusters) for large scale resource sharing and problem solving [1]. Thus, a cluster federation can also be viewed as a hybrid distributed environment containing multiple heterogeneous subsystems (clusters).

Because of its growing importance, fault-tolerant aspect of cluster computing environment deserves significant attention. Check pointing and rollback recovery are widely used techniques that offer fault-tolerance in distributed systems [4], [5], [16]-[23]. 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 [4], [5].

In cluster computing, a cluster may employ either coordinated or independent check pointing scheme for its processes to take their local checkpoints. We term this checkpointing as the primary level of checkpointing. Note that in cluster computing failure of a cluster means failure of its one or more processes. It is also the responsibility of each cluster to determine its consistent local checkpoint set that consists

of one checkpoint from each process present in it. This consistency means that between every two checkpoints of this set there is no orphan message [1], [2], [4], [5]. (It may be noted that a message sent by a sender S is an orphan if its sending event has not been recorded in a checkpoint of the sender S, but its receiving event has been recorded in a checkpoint of the receiver R). But this consistent local checkpoint set (also known as cluster level checkpoint of the 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. Therefore, a collection of consistent local checkpoint sets, one from each cluster in the federation, does not necessarily produce a consistent federation level checkpoint (also known as federation level recovery line). Consequently, rollback of one failed cluster may force some other clusters to rollback in order to maintain consistency of operation by the cluster federation. This uncontrolled propagation of rollback is known as domino-effect [10], [24]. There is, therefore, a need to have a second level of checkpointing algorithm that helps in determining a consistent federation level checkpoint so that there is no inter cluster orphan message between any two cluster level checkpoints of the recovery line. Then, after recovery from a failure the individual clusters in the federation can restart their computation from their respective cluster level checkpoints belonging to the recovery line. In this context, note that after recovery from a failure even if the clusters restart from their respective cluster level checkpoints belonging to a federation recovery line, still it does not necessarily ensure correctness of computation. To achieve it, any inter cluster message that may become a lost message because of the failure must be identified and resent to the appropriate receiving cluster. The responsibility of the receiving cluster is that it must process all such lost messages following the order of their arrival before the occurrence of the failure [13]-[15]. An example of a lost message is shown in Figure 1. In this figure, after the system recovers from the failure f, if the two clusters C^i and C^j restart from their respective checkpoints CLC^i and CLC^j, then message m will be treated as a lost message. The reason is that cluster C^j does not have a record of the receiving event of the message as cluster C^i has the record of sending it in its checkpoint CLC^i. In such a situation, for correct computation this lost message m has to be identified and cluster C^i must resend it to cluster C^j after the system restarts.

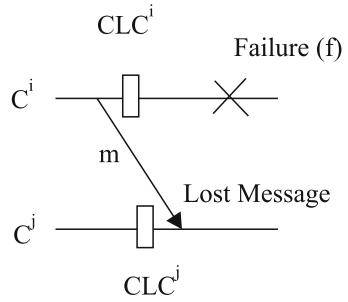

Figure 1. Message m is a lost message.

Although several recovery approaches exist for distributed systems [3]-[8], [11], [12], [25], [28], due to the complexity involved in finding a federation level recovery line, very few works exist in the area of cluster computing [1], [2].

In this work, we present a recovery approach for cluster computing environment that considers both determination of a federation level recovery line and resending of all inter cluster lost messages. It is a two phase approach. First, we propose a fast recovery algorithm to determine a federation level recovery line that guarantees the absence of any inter cluster orphan message with respect to the cluster level checkpoints belonging to the recovery line. Then we apply the existing idea on sender-based message logging approach for distributed computing [13] to cluster computing to identify and resend inter cluster lost messages. It helps a receiving cluster, after it restarts, to process these messages following the order of their arrival before the occurrence of the failure.

2. Relevant Data Structures

Before we state the relevant data structures and their use in our proposed algorithm we need to define the following. A *cluster level checkpoint* (*CLC*) of a cluster is defined as a set of local checkpoints, one from each process belonging to the cluster, such that these checkpoints are mutually consistent. In other words, a CLC represents a recovery line for the cluster; however this CLC may not be consistent with CLCs

of other clusters. As in [1] and [2], we assume that inside a cluster processes take these local checkpoints periodically in a coordinated way [12]. A CLC taken in this way is termed in this paper as *regular cluster level checkpoints*. Besides, in our approach a cluster also takes a cluster level checkpoint in a coordinated way if it receives an inter cluster application message from another cluster. We call it a *forced cluster level checkpoint*. Therefore, a forced CLC may be considered as a communication-induced one [6], [26], [27]. As in [2], we assume that the two events of receiving an inter cluster application message and taking a forced CLC occur together atomically. *A consistent federation level checkpoint* (i.e. *a federation level recovery line*) is a set of the CLCs, one from each cluster, such that these CLCs are mutually consistent; that is, there is no orphan message in the system with respect to this set of the CLCs.

Below we justify the motivation for using hierarchical check pointing approach.

We assume that inside a cluster processes take local checkpoints periodically in a coordinated way. This ensures that these checkpoints inside a cluster are consistent. The assumption that the number of inter cluster messages is low justifies the use of communication induced check pointing scheme between two communicating clusters. In our approach a cluster takes a cluster level checkpoint in a coordinated way if it receives an inter cluster application message from another cluster which we call a forced cluster level checkpoint and it is in fact a communication-induced checkpoint. This means that each cluster takes its CLC independently, but information is added to each inter-cluster communication that results in taking a forced CLC by its receiving cluster. Therefore, we propose a hierarchical protocol combining coordinated and communication-induced check pointing.

It may be noted that taking a forced CLC every time a cluster receives an inter cluster application message is not an overhead because, in general, such type of message exchange between any two clusters occurs quite infrequently [1], [2]. We use the following notations in this work to represent a cluster and its processes.

Let the cluster federation under consideration consist of N clusters, where each cluster consists of a number of processes. The j^{th} process of the i^{th} cluster is denoted as p_j^i and i^{th} cluster as C^i. For cluster C^i consisting of r processes, its m^{th} cluster level checkpoint is represented as $CLC_m^i = \{cp_1^m, cp_2^m, \ldots, cp_{r-1}^m, cp_r^m\}$, where cp_j^m is the m^{th} local checkpoint taken by process p_j of cluster C^i. Note that all these m^{th} local checkpoints are taken following the coordinated checkpointing approach and so are mutually consistent. That is, CLC_m^i represents a recovery line for cluster C^i. In this context, note that by the statement, 'a process p_j^i in C^i stores the corresponding CLC_m^i in the stable storage', we mean that process p_j^i stores its local m^{th} checkpoint cp_j^m that belongs to CLC_m^i. Also in cluster computing environment, communication between two clusters means communication between two processes belonging to these two clusters respectively and failure of a cluster means failure of its one or more processes.

Corresponding to every cluster level checkpoint, for example say CLC_m^i, every process p_j^i in cluster C^i maintains the following three vectors at its m^{th} local checkpoint, which are same for all processes in the cluster at their respective m^{th} local checkpoints. Since CLC_m^i is the set of these m^{th} local checkpoints of the processes in C^i and these vectors are same for all processes in C^i, hence for simplicity we will assume that as if cluster C^i maintains these three vectors at CLC_m^i. These three vectors are initialized with 0s at the initial state (starting state) of a cluster (i.e. at the starting states of the processes in it). These vectors are stated below.

1. $V_{m(sent)}^i = [v_m^{i,0}, v_m^{i,1}, \ldots, v_m^{i,N-1}]$, where $|V_{m(sent)}^i| = N$=Number of clusters in the cluster federation and $v_m^{i,j}$ represents the number of inter cluster application messages sent from cluster C^i to any cluster C^j. Initially $v_m^{i,j} = 0$, for $0 \leq j \leq N-1$.
2. $V_{m(recv)}^i = [r_m^{i,0}, r_m^{i,1}, \ldots, r_m^{i,N-1}]$, where $|V_{m(recv)}^i| = N$=Number of clusters in the cluster federation and $r_m^{i,j}$ represents the number of inter cluster application messages received by cluster C^i from cluster C^j. Initially $r_m^{i,j} = 0$, for $0 \leq j \leq N-1$.
3. $CIC_m^i = [c_0^{i,0}, c_1^{i,1}, \ldots, c_{m-1}^{i,m-1}]$, and $|CIC_m^i| = m$ =Number of CLC's taken by C^i.

$$\text{where } CIC^i_m(m) = \begin{cases} CIC^i_m(m-1)+1 & \text{if the } CLC^i_m \text{ is a forced checkpoint} \\ CIC^i_m(m-1) & \text{if the } CLC^i_m \text{ is a regular checkpoint.} \end{cases}$$

For example, at the initial checkpoint CLC^i_0, the vector $CIC^i_0 = [c^{i,0}_0] = [0]$. And at the second checkpoint CLC^i_1, the corresponding vector $CIC^i_1 = [c^{i,0}_0, c^{i,1}_1] = [0, c^{i,1}_1]$, where $c^{i,1}_1 = c^{i,0}_0 + 1 = 1$, if the checkpoint CLC^i_1 is a forced cluster level checkpoint; and $c^{i,1}_1 = 0$ if the checkpoint CLC^i_1 is a regular cluster level checkpoint. In a similar way all other entries in the vector CIC^i_m are updated. In this work, note that when we do not need to specify a particular checkpoint number, we will simply use the notations $V^i_{(sent)}$, $V^i_{(recv)}$, and CIC^i to represent the three vectors.

Besides the above mentioned vectors, each process in a cluster maintains a Boolean flag. The use of this flag has been stated in the following section.

Observation 1: *At any cluster level checkpoint CLC^i_r in cluster C^i, the value of the last element of the CIC^i_r vector denotes the total number of forced checkpoints taken by cluster C^i till its checkpoint CLC^i_r.*

Observation 2: *At any cluster level checkpoint CLC^i_r in cluster C^i, the length of the CIC^i_r vector (i.e. the number of elements in it) denotes the total number of cluster level checkpoints, including both regular and forced ones taken by the cluster C^i till its checkpoint CLC^i_r.*

The updating of the vectors will become clearer in the following section.

3. Working Principle

In this section we first present how different vectors are updated. We then briefly outline how the proposed recovery mechanism works, followed by an illustration.

The updating of the vectors will become clear from the following example. Consider the two cluster system as shown in Figure 2. Two inter cluster application messages, m_1 and m_2, are sent from C^i to C^j. Initially, the two clusters take their respective initial cluster level checkpoints CLC^i_0 and CLC^j_0. The CIC vectors at the two clusters are given in Table 1.

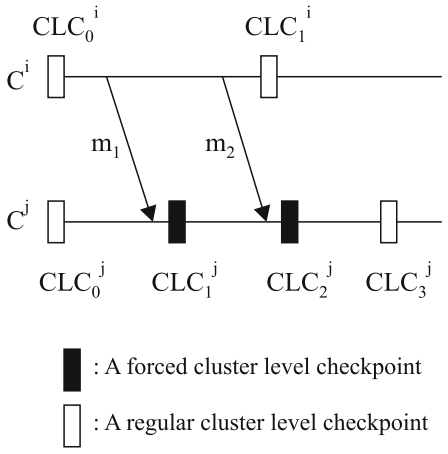

Figure 2. An example of updating the vectors.

In Table 1 consider CIC^j at the cluster level checkpoint CLC^j_3. It is [0 1 2 2]. In this vector, total number of elements $(= 4)$ represents the total number of CLCs (including both regular or forced ones) taken by cluster C^j and the value of the last element $(= 2)$ in the vector represents the total number of forced CLCs taken.

Cluster	Checkpoint	$V_{(sent)}$	$V_{(recv)}$	CIC
C^i	CLC^i_0	[0 0]	[0 0]	[0]
	CLC^i_1	[0 2]	[0 0]	[0 0]
C^j	CLC^i_0	[0 0]	[0 0]	[0]
	CLC^j_1	[0 0]	[1 0]	[0 1]
	CLC^j_2	[0 0]	[2 0]	[0 1 2]
	CLC^j_3	[0 0]	[2 0]	[0 1 2 2]

Table 1. Vectors at different checkpoints.

For a clear understanding of our approach, throughout this paper we will use the following interpretations needed to design the proposed recovery algorithm: (1) by the statement 'a cluster C^k rolls back to its r^{th} cluster level checkpoint CLC^k_r' we mean that all processes in cluster C^k rollback to their respective local checkpoints which form together the cluster level checkpoint CLC^k_r; (2) by 'initiator cluster' we mean the cluster that contains the initiator process. In fact, in our work a failed process

inside the initiator cluster actually initiates the recovery mechanism after this process recovers from the failure; (3) by the statement 'a cluster C^k receives a request from the initiator cluster C^i and sends its vector and its Boolean flag to it', we mean that the process ($\in C^k$) receiving the request from the initiator process ($\in C^i$) sends its vector and its flag to the initiator; (4) by the statement 'the initiator cluster sends/receives a message' it means that the initiator process in this cluster actually sends/receives the message; (5) if any of the processes in a cluster rolls back, the respective Boolean flags of all processes in that cluster are set at 1; otherwise these flags are set at 0 each; (6) finally by 'a computation done or an action taken by the initiator cluster associated with the recovery scheme' it means that it is actually performed by the initiator process belonging to this cluster. Similarly, by 'a computation done or an action taken by any other cluster associated with the recovery scheme' it means that it is performed by a process of this cluster.

Recovery mechanism: Unless otherwise needed, we will simply use the notations $V^i_{(sent)}$, $V^i_{(recv)}$, and CIC^i to represent the three vectors. A failed process p^i_j inside a cluster C^i initiates the recovery mechanism after it recovers from the failure. Therefore, cluster C^i acts as the initiator cluster. To start with, this initiator cluster first rolls back to its latest cluster level checkpoint and then sends a request message to each cluster C^k, for $0 \leq k \leq N-1$, $k \neq i$ asking it to send its $V^k_{(sent)}$ vector corresponding to its latest cluster level checkpoint. After receiving the vector $V^k_{(sent)}$ from all clusters, the initiator cluster C^i forms a two dimensional array V^N.

$$V^N = \begin{vmatrix} v^{0,0} & v^{0,1} & \cdots & v^{0,N-1} \\ v^{1,0} & v^{1,1} & \cdots & v^{1,N-1} \\ \vdots & \vdots & \cdots & \vdots \\ v^{k,0} & v^{k,1} & \cdots & v^{k,N-1} \\ v^{N-1,0} & v^{N-1,1} & \cdots & v^{N-1,N-1} \end{vmatrix}$$

where the k^{th} row represents $V^k_{(sent)}$ corresponding to cluster C^k, for $0 \leq k \leq N-1$. The initiator cluster then computes the column sums to create the following vector.

$$V^c = [v^0_c, v^1_c, v^2_c, \ldots, v^k_c, \ldots, v^{N-1}_c]$$

where v^k_c = column sum of the entries of the k^{th} column of V^N and is given by $v^k_c = \Sigma V^N(l,k)$, for l= 1 to N. Therefore, v^k_c represents the total number of inter cluster messages sent to cluster C^k from all other clusters.

The initiator cluster then unicasts $v^k_c (= V^c(k))$ to each corresponding cluster C^k, for $0 \leq k \leq N-1$, $k \neq i$. After receiving v^k_c from the initiator, each cluster C^k adds the elements of its $V^k_{(recv)}$ vector (actually as mentioned earlier this computation is performed by the process $p^k_x (\in C^k)$ which has received the unicast information v^k_c). Let the sum be v^k_r. Therefore, v^k_r represents the total number of inter cluster messages received by the processes in cluster C^k from all other clusters.

Cluster C^k (i.e. Process p^k_x) now computes $D^k = v^k_r - v^k_c$. The difference D^k (if > 0) between v^k_r and v^k_c gives the exact number of inter cluster orphan messages received by a cluster C^k from all other clusters. Process p^k_x now checks the last element (let it be X) present in CIC^k vector at its latest checkpoint; this element is the number of forced CLCs taken so far by cluster C^k. Process p^k_x rolls back to its latest checkpoint (say, it is the l^{th} checkpoint) where the last element in its corresponding CIC^k vector is equal to $X - D^k$. It also unicasts a message to all other processes in its cluster to rollback to their respective l^{th} checkpoints. Observe that all these l^{th} checkpoints of the processes of cluster C^k form the cluster level checkpoint CLC^k_l. Thus, effectively, it can be said that the cluster C^k rolls back to its cluster level checkpoint CLC^k_l. Observe that all these l^{th} checkpoints of the processes of cluster C^k are assumed to have been taken during the l^{th} execution of the coordinate checkpointing protocol.

We have already mentioned that if any of the processes in a cluster C^k rolls back (i.e. $D^k > 0$), the Boolean flags of all processes in C^k are set at 1. The effect of this rollback is that the corresponding cluster C^k (i.e. actually process p^k_x) sends this flag value ($= 1$) along with its $V^k_{(sent)}$ corresponding to the checkpoint to which it has rolled back. If the cluster does not roll back (i.e. $D^k \leq 0$), then it will send only a flag value of 0. The algorithm will terminate when for

each cluster C^i, its corresponding flag value is equal to zero. That is, none of the clusters rolls back. Otherwise, the algorithm starts its next iteration. In this case, for any cluster that sent a flag of 0, its sent vector used in the previous iteration is used again in the current iteration.

An Illustration: Figure 3 gives an illustration of how cluster level checkpoints are taken in our approach as well as how a federation level recovery line is determined. Each horizontal line represents a parallel execution on a cluster. Each cluster C^i (i.e. each process in this cluster) maintains three vectors $V^i_{(sent)}$, $V^i_{(recv)}$, and CIC^i.

Initially all these vectors are initialized with zeros at the initial checkpoints. Cluster C^1 takes a forced cluster level checkpoint CLC^1_1 as soon as it receives the application message m_2 and updates CIC^1_1 from [0] to [01] (we take the last value in the vector at prior checkpoint, increment it by 1 and append it to the vector so that the last element of the new vector gives us the total number of forced checkpoints taken so far) and $V^1_{1(recv)}$ from [000] to [001] because it has received an inter cluster application message from cluster C^2. It also updates $V^1_{1(sent)}$ from [000] to [100] because it has sent an inter cluster application message m_3 to cluster C^0 after the checkpoint CLC^1_0 was taken.

Consider the cluster level checkpoint CLC^1_2 in cluster C^1. As this checkpoint is a regular CLC taken within the cluster, the CIC^1_2 is updated from [01] to [011], $V^1_{2(sent)}$ remains same as [100]. $V^1_{2(recv)}$ also remains same because it has not received any inter cluster application message after the checkpoint CLC^1_1. Similarly all checkpoints for all other clusters are taken and their vectors updated.

Suppose at time t, a failure 'f' occurs in cluster C^1. After recovering from the failure, cluster C^1 first rolls back to the checkpoint CLC^1_2. The algorithm is now initiated by cluster C^1. To start with, initiator cluster C^1 broadcasts a request asking the clusters C^0 and C^2 to send

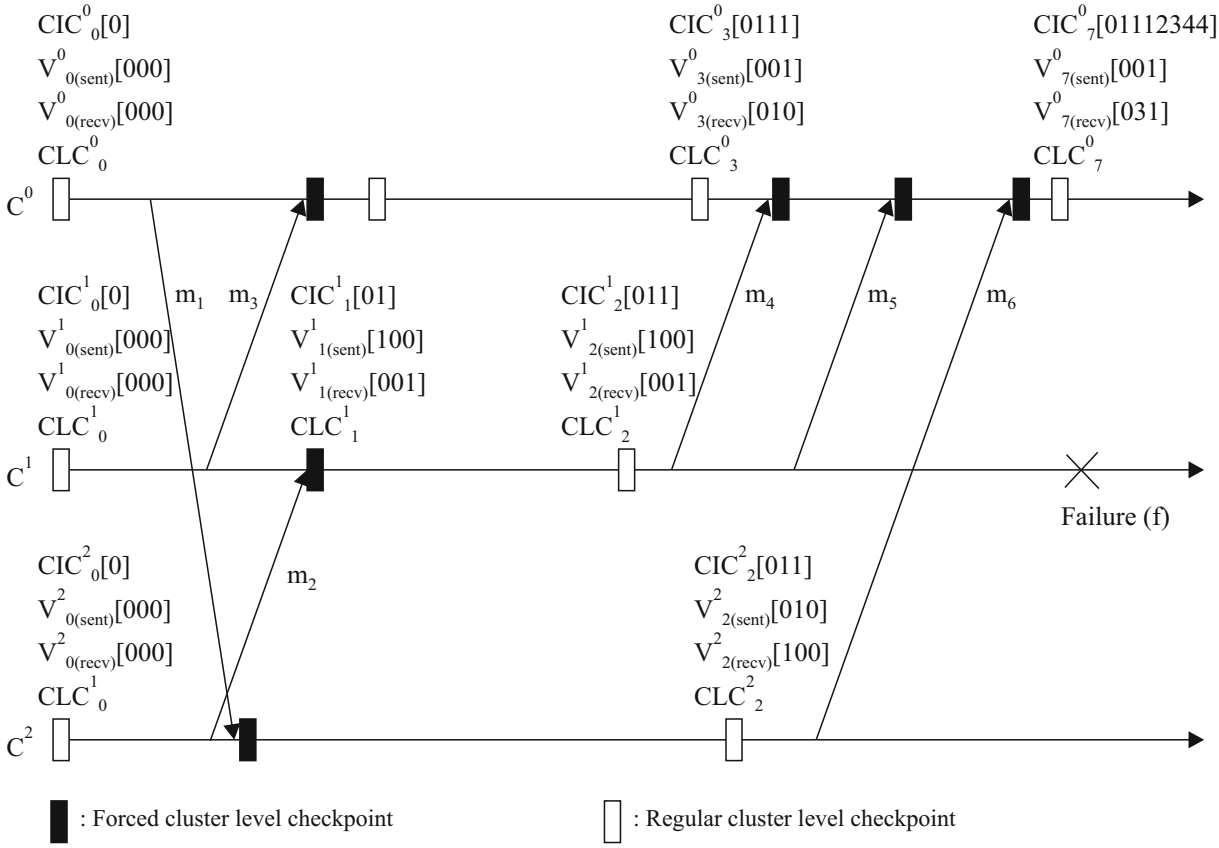

Figure 3. Federation level consistent checkpoints $\{CLC^0_3, CLC^1_2, CLC^2_2\}$.

their sent vectors corresponding to their latest checkpoints. In this example cluster C^0 sends the vector [001] and cluster C^2 sends [010]. After receiving the vectors the initiator creates a two dimensional array and performs the column sum and calculates V^3 in the following way:

$$\begin{vmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{vmatrix}$$

$V^3 = \text{Column Sum} = \overline{1 \quad 1 \quad 1}$

Now, cluster C^1 unicasts v_c^k to each cluster C^k, for $k = 0, 2$. After receiving v_c^k, each cluster C^k adds the elements in its $V^k_{(recv)}$ at its latest checkpoints to compute v_r^k and then it computes D^k $(= v_r^k - v_c^k)$. In this example, at the respective latest checkpoints of the three clusters we get the following: D^0 equal to 3 for cluster C^0, D^1 equal to 0 for cluster C^1, and D^2 equal to 0 for cluster C^2. This implies that cluster C^0 has received three orphan messages with respect to its latest checkpoint CLC_7^0; in fact the orphan messages are m_4, m_5, and m_6. Observe that cluster C^1 and cluster C^2 have received no orphan messages.

Now cluster C^0 checks the last element $(=X)$ of CIC_7^0. In this example it is 4. Then it calculates the difference d $(=X-D^0)$; in this example d is1 $(= 4 - 3)$. C^0 will now skip to a latest checkpoint where the last element of CIC^0 vector is equal to 1. This checkpoint is CLC_3^0.

Now cluster C^0 rolls back by 4 checkpoints i.e. to CLC_3^0 and sends a flag of 1 along with its $V^0_{3(sent)}$ to the initiator cluster. Cluster C^2 sends only a flag of 0 because it has not rolled back.

Cluster C^1 checks whether the flag values from all the clusters are equal to zero. If they are, then it will ask all clusters to restart their applications from their respective present CLCs; else it executes the next iteration. In this example, flag of cluster C^0 is equal to 1 and so the algorithm executes its next iteration. Cluster C^0 sends its sent vector at its checkpoint CLC_3^0. Since both the initiator cluster C^1 and cluster C^2 have not rolled back, their flags are 0 each. As a consequence, in this iteration the initiator cluster will use the same sent vectors both for itself as well as for C^2 which it used in the previous iteration. So we again calculate V^3. It is given below.

$$\begin{vmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{vmatrix}$$

$V^3 = \text{Column Sum} = \overline{1 \quad 1 \quad 1}$

Cluster C^1 unicasts v_c^k to cluster C^k, for $k = 0, 2$. As before, after receiving v_c^k each cluster C^k, adds the elements in its $V^k_{(recv)}$ to compute v_r^k and then it computes D^k $(= v_r^k - v_c^k)$. In this second iteration, we get D^0 equal to 0 for the checkpoint CLC_3^0 of cluster C^0; D^1 equal to 0 for cluster C^1 for its checkpoint CLC_2^1 and D^2 equal to 0 for cluster C^2 for its checkpoint CLC_2^2. This implies that there is no orphan message in the cluster federation with respect to these three checkpoints.

Now both clusters C^0 and C^2 send a flag of 0. Cluster C^1 has its own flag also set at 0. This is the termination condition of our approach. Hence the federation level recovery line can be represented as the set $\{CLC_3^0, CLC_2^1, CLC_2^2\}$.

Observe that in this example in the second iteration cluster C^0 starts from CLC_3^0 and in the first iteration it started from CLC_7^0. So we have been able to skip the comparison with checkpoints in between these two checkpoints, because they cannot belong to any federation level recovery line. Also note that the number of trips to the stable storage has been reduced by the number of checkpoints that have been skipped. Observe that the work in [2] fails to avoid these unnecessary trips to stable storage, resulting in a slower execution for recovery line determination compared to our work. Also, observe that the clusters simultaneously take their decisions whether to rollback or not. Thus our approach offers parallel execution as in [2].

The above discussion leads to the following Lemmas and Theorems.

Lemma 1: If $D^i > 0$, then cluster C^i has received D^i number of orphan messages from other clusters.

Proof: v_r^i represents the total number of messages cluster C^i has received so far and these are recorded in C^i's latest CLC, and v_c^i represents the total number of messages sent by all

other clusters to C^i as recorded in their latest CLC's. Therefore $D^i(= v^i_r - v^i_c) > 0$ means that at least some cluster C^k ($k \neq i$) has sent some message(s) to cluster C^i after taking its latest checkpoint. Since all such D^i messages have been received and recorded in C^i's latest CLC, but remain unrecorded by the sending clusters, C^i has received D^i number of orphan messages from the rest of the clusters.

Lemma 2: If $D^i \leq 0$, then cluster C^i has not received any orphan message.

Proof: $D^i = 0$ means that the number of messages received by cluster C^i is equal to the number of messages sent to cluster C^i and so the sending events of these messages are already recorded by the sending clusters in their latest checkpoints. Hence, the received messages cannot be orphan.

Also, $D^i < 0$ means that the number of the received messages by cluster C^i is less than the number of messages sent to it. It means that all the messages received by cluster C^i have already been recorded by the senders. Hence none of such received messages can be an orphan.

Theorem 1: Let $D^i > 0$ at the r^{th} checkpoint CLC^i_r of cluster C^i and the last element of the CIC^i_r vector at this checkpoint be X. Let CLC^i_m be the latest checkpoint prior to CLC^i_r such that the last element of CIC^i_m is equal to $X-D^i$. Then none of the checkpoints CLC^i_r, CLC^i_{r-1}, $CLC^i_{r-2}, \ldots, CLC^i_{m+1}$ can belong to any federation level recovery line.

Proof: According to Lemma 1, C^i has received exactly D^i number of orphan messages from all other clusters till its latest checkpoint CLC^i_r. Given that the last element of the CIC^i_r vector at the checkpoint CLC^i_r is X, this implies that the cluster C^i has taken X forced checkpoints so far according to Observation 1. But a cluster takes a forced CLC whenever it receives an inter cluster application message. Thus, in this case cluster C^i has recorded the events of receiving X inter cluster application messages at the checkpoint CLC^i_r. With respect to the checkpoint CLC^i_r it is clear that D^i is the number of orphan messages received by cluster C^i from all other clusters. So out of these X messages, only $X-D^i$ messages are such that their sent events are recorded by some other clusters. Thus cluster C^i has to rollback to a latest checkpoint which has recorded the receiving event of the $(X-D^i)^{th}$ inter cluster application message, skipping all the checkpoints which have recorded the events of receiving the orphan inter cluster application messages, received after the $(X-D^i)^{th}$ inter cluster application messages.

We also have assumed that the CLC^i_m is the latest checkpoint prior to CLC^i_r such that the last element of CIC^i_m is equal to $X-D^i$, thus CLC^i_m is the latest checkpoint that has recorded the receiving event of the $(X-D^i)^{th}$ inter cluster application message. Thus, the application messages which have caused the creation of the checkpoints CLC^i_r, CLC^i_{r-1}, $CLC^i_{r-2}, \ldots$, CLC^i_{m+1} are orphan and hence these checkpoints cannot belong to any federation level recovery line.

Theorem 2: If $D^i \leq 0$ at the latest checkpoint of each cluster C^i, for $0 \leq i \leq N-1$ (i.e. flag of each C^i is 0), then the recovery algorithm terminates and all such latest checkpoints form a consistent federation level checkpoint of the cluster federation.

Proof: According to Lemma 2, $D^i \leq 0$ at the latest checkpoint of each cluster C^i means that none of the clusters in the cluster federation has received any orphan message till its latest checkpoint. Thus the set of all such checkpoints, one from each cluster, are mutually consistent and hence they form a consistent federation level checkpoint of the cluster federation.

4. Algorithm Recovery

Input: Given the latest N cluster level checkpoints, one for each cluster C^j, $0 \leq j \leq N-1$, for an N cluster system and the corresponding vectors $V^j_{(sent)}$, $V^j_{(recv)}$, CIC^j at these checkpoints.

Output: A federation level recovery line which is also a maximum consistent state of the cluster federation.

The responsibilities of each cluster C^i and the initiator cluster C^k are stated below.

Initiator cluster C^k:

Step 1: it asks each cluster C^i for $0 \leq i \leq N-1$, $i \neq k$, to send its $V^i_{(sent)}$ at its latest checkpoint CLC^i_r;

/ at CLC_r^i the vectors are same for all processes in cluster C^i */*

Step 2: it receives all $V^i_{(sent)}$ for $0 \leq i \leq N-1$;

Step 3: it computes $V^c = v_c^0, v_c^1, v_c^2, \ldots, v_c^k, \ldots, v_c^{N-1}$;

Step 4: it unicasts $v_c^i (= V^c(i))$ to each cluster C^i, for $0 \leq i \leq N-1$;

Step 5: it receives either a flag or (flag and $V^i_{(sent)}$) from each cluster if flag $= 0$ for each cluster C^i, for $0 \leq i \leq N-1$

/ no cluster rolls back, i.e. $D^i \leq 0$ for each cluster C^i */*

cluster C^k asks every cluster C^i for $0 \leq i \leq N-1$, $i \neq k$ to restart the application program from its last checkpoint corresponding to which $D^i \leq 0$ and cluster C^k does the same for itself; the algorithm terminates;

/ a federation level recovery line is determined */*

else

control flows to step 3;

/ for any cluster which has sent a flag of 0, its sent vector received in the previous iteration is used again */*

Cluster C^i:

Step 1: cluster C^i receives request from cluster C^k;

if C^k has requested to restart

processes in C^i restart from their respective local checkpoints corresponding to the CLC^i

where $D^i \leq 0$;

else

it sends $V^i_{(sent)}$ at its latest cluster level checkpoint to the initiator cluster C^k;

Step 2: it receives v_c^i from initiator cluster C^k;

Step 3: it computes D^i;

Step 4: if $D^i > 0$

/ C^i needs to rollback */*

it calculates $(X - D^i)$;

/ X is the last element in CIC_r^i */*

it sends a flag of 1 and $V^i_{(sent)}$ corresponding to its checkpoint CLC^i_m; (i.e. CLC^i_r is replaced by CLC^i_m)

/ C^i rolls back to CLC_m^i and CLC_m^i is the latest checkpoint prior to CLC_r^i such that the last element of CIC_m^i is equal to $X-D^i$; that is, the checkpoints CLC_r^i, CLC_{r-1}^i, CLC_{r-2}^i,..., CLC_{m+1}^i cannot belong to any federation level recovery line*/*

else

it sends a flag of 0 to cluster C^k;

Correctness Proof: Each Cluster C^i repeats its steps 1, 2, 3 and 4 to arrive at a checkpoint that has not recorded the receipt of any orphan message from the other clusters (using the observations of Lemmas 1 and 2). In other words, it identifies the checkpoints that cannot belong to the federation level recovery line and skips them (using the logic of Theorem 1). This decision is made based on the value of D^i. However, the initiator cluster C^k decides when to terminate the algorithm, i.e., when the cluster level checkpoints can become mutually consistent. Cluster C^k checks to see if $D^i \leq 0$ for each cluster C^i. If so, the algorithm terminates according to Theorem 2. Note that the condition $D^i \leq 0$ must always occur during the execution of the algorithm. It may be observed that in the worst case, because of some typical communication pattern, the domino effect may force processes in all clusters to restart from their initial states where for each cluster C^i we always have $D^i = 0$. Besides, since the algorithm starts with the latest checkpoints, the number of events (states) rolled back at each cluster is a minimum. This is true because, in its Step 4 each cluster C^i skips only the checkpoints that cannot belong to a federation level recovery line. Thus, the algorithm determines a federation level recovery line which is the maximum consistent state of the federation as well.

Message Complexity: Suppose the termination of the algorithm requires the construction of the vector V^N by the initiator cluster C^k to occur k times (i.e. k number of iterations). During each such time every cluster in the N-cluster system exchanges a couple of messages with initiator cluster C^k. Thus, $O(N)$ messages are sufficient for each time. Thus, considering k times, the message complexity of the algorithm is *O(kN)*.

5. Inter Cluster Lost Messages

The sender-based message logging scheme proposed for distributed computing [13] to identify and resend lost messages is used in this work. This scheme has been the choice since it does not require message ordering and message logging is done asynchronously. We apply it to cluster federation in the following way. When a sending process, say p_i in a cluster sends an inter cluster message m to a process p_k in another cluster, the message m is piggybacked with a send sequence number (SSN) which represents the number of messages sent by this process. The sender also logs the message m and its SSN in its local log. The receiving process p_k will assign a receive sequence number (RSN) to the message m, which represents the number of messages received by p_k. The RSN is incremented each time p_k receives a message. It then sends the RSN back to the sender p_i. After receiving the RSN corresponding to m, the sender records the RSN with the log of the message m. Thus message m is called a fully logged message. This local log is saved in stable storage when p_i takes its next checkpoint. Process p_i then sends an acknowledgement, ack to the receiver. In the meantime after sending the RSN to p_i, process p_k continues its execution, but cannot send any message (intra or inter cluster) until it has received the ack. Note that if the receiver fails before sending the RSN of the message m, the log of m does not have the RSN. In such a situation message m is called partially logged.

Recovery is performed when p_k fails. It restarts from its checkpoint that belongs to the federation level recovery line as determined by Algorithm Recovery. Now p_k looks for those (if any) inter cluster messages such that their sending events have already been recorded in the respective senders' checkpoints and the receiving events have not been recorded in p_k's checkpoint belonging to the recovery line. These are the inter cluster lost messages. To get these messages back, the receiver broadcasts a request to all clusters for resending the lost messages. At this time the receiver also sends the value of the SSNs for different sender processes. Every sender then resends only those messages with a higher SSN that were sent to p_k before the failure.

The messages received by p_k from the senders are consumed by p_k in the order of their RSNs. Since the messages that were assigned an RSN by the receiver form a total order, therefore process p_k gets the same sequence of messages as it did before the failure and therefore executes the same sequence of instructions as it did before the failure.

Next, p_k receives the partially logged messages following the fully logged ones. These partially logged messages do not have any RSN values attached to them. So there is no total ordering imposed on them by p_k. However, according to the work in [13] the receiver was constrained from communicating with any process if the ack for any message it had received was pending. Therefore any order in which these partially logged messages are resent to p_k is acceptable [13].

Observe that in our approach a federation level recovery line means that there does not exist any inter cluster orphan message between any two cluster level checkpoints belonging to the recovery line. Hence the mechanism to handle orphan messages in [13] is not needed in our approach. In this context, note that if the CLCs are taken following the single phase non-blocking check pointing algorithm of [3], there will be no intra cluster orphan messages either. Also note that the above schemes about handling inter cluster lost messages can be similarly applied to handle any intra cluster lost messages. Thus absence of any orphan and lost messages ensures correctness of computation.

6. Comparison

In the present work, we have considered both the determination of a federation level recovery line and identification of any inter cluster lost messages. The works in [1] and [2] have not considered the identification of inter cluster lost messages. The authors considered only the determination of a federation level recovery line. So we compare below only the effectiveness of our approach to find a recovery line with those of the works in [1] and [2].

Comparison with the work in [1]: In [1] a checkpointing/recovery algorithm has been proposed for typical cluster federation architecture which integrates independent and coordinated checkpointing schemes for applications running in

a hybrid distributed environment. In this architecture multiple coordinated checkpointing subsystems are connected with a single independent checkpointing subsystem. Each such 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. The architecture is a very restricted one in the sense that the above mentioned multiple coordinated subsystems cannot communicate directly with each other; rather, they do it via the independent subsystem.

The assumed restricted architecture is the main shortcoming of this work. Our proposed approach is independent of any particular architecture.

Comparison with the work in [2]: The algorithm in [2] and ours have the following similarities. Both are architecture independent unlike in [1] and both determine the same federation level recovery line. Also these algorithms are simultaneously executed by all participating clusters, which obviously contributes to the speed of their execution.

However, the main drawback of the algorithm in [2] is that if we consider a particular application message pattern because of which all the clusters have to roll back except the failed cluster, then all these clusters broadcast a control message (named as alert message) before the start of the next iteration. This results in message storm. In our algorithm no such situation arises since its execution is centrally controlled by the initiator cluster.

Also, in [2] a cluster may have to make much larger number of trips to the stable storage compared to our approach, in order to determine which cluster level checkpoint(s) need to be skipped (i.e. which checkpoints cannot belong to the recovery line of the cluster federation). Note that larger the number of such trips, larger will be the execution time of the algorithm. To compare this number of trips for the two approaches, let us assume the following approximate analysis that offers a clear understanding of the advantage our approach offers over the one in [2]. Let us assume that in both algorithms each cluster will skip on an average r number of checkpoints per iteration and the algorithms will determine the federation level recovery line in k number of iterations. Then we find that in [2] the number of trips to the stable storage is (k+kr) compared to just k in our approach. In Table II we have summarized the comparisons.

Criteria	Our Algorithm	Algorithm [2]
Message Storm	No	Yes
Simultaneous execution by clusters	Yes	Yes
Architecture dependent	No	No
Number of trips to stable storage	k	k+kr
Message complexity	$O(kN)$	$O(kN^2)$

Table 2. Comparison with the work In [2].

Efficiency of the algorithm in terms of communication cost: To get a clear idea about how the communication cost of our recovery algorithm varies with the increase in the number of clusters when compared to that of the algorithm in [2], we do the following.

Let C_{air} be the communication cost of sending a message from one cluster to another cluster;

and C_{broad} be the communication cost of broadcasting a message to all clusters.

The algorithm in [2] and the one we have presented here will require the same number of iterations in order to determine a federation level recovery line. In its worst case [2], all clusters, except the failed one, rollback in each iteration until the recovery line is determined. Therefore, all these clusters will broadcast a control message (alert message) in each iteration.

So, in the worst case in each iteration, the cost is: $(N-1)\cdot C_{broad} = (N-1)^2 \cdot C_{air}$.

Total cost in k iterations is: $k\cdot(N-1)^2\cdot C_{air}$.

On the other hand, if on an average $N/2$ clusters roll back, the total cost is: $k\cdot(N/2)\cdot(N-1)\cdot C_{air}$.

For best case in [2], only one cluster rolls back in each iteration; so the total cost is: $k\cdot C_{broad} = k\cdot(N-1)\cdot C_{air}$.

In our approach, in each of the k iterations every cluster exchanges a couple of messages with the initiator cluster, irrespective of how many clusters roll back.

Thus, in our approach the total cost is always: $k \cdot 2(N-1) \cdot C_{air}$.

Without any loss of generality we have assumed $C_{air} = 1$. Figure 4 illustrates how the costs are affected by the increase in the number of clusters in [2] and in our approach. It is observed that the best case in [2] performs better than ours. However, our approach is shown to offer much better result when considering the worst and average cases of [2].

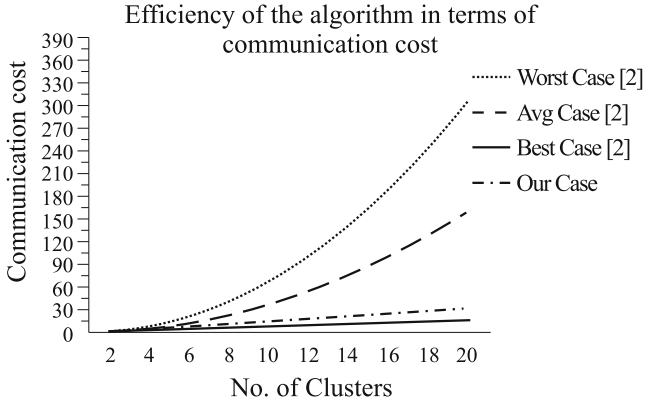

Figure 4. Efficiency of the algorithm in terms of communication cost.

7. Conclusion

In this paper, we have proposed a recovery approach for handling both inter cluster orphan and lost messages. The main feature of the proposed recovery algorithm to handle inter cluster orphan messages is that it is executed simultaneously by all participating clusters while determining a federation level recovery line. Besides, the algorithm in its each iteration does not need to compare all vectors at all checkpoints of the clusters. It reduces computational overhead to a good extent and as a result its execution becomes even faster. Also, the algorithm offers much smaller number of trips to the stable storage compared to the same in [2]. Note that if the CLCs are taken following the single phase non-blocking check pointing algorithm of [3], there will be no intra cluster orphan messages as well. We have used the existing idea of sender-based message logging [13] to handle effectively inter cluster lost messages. This idea can be similarly extended to handle all intra cluster lost messages as well. Thus absence of any orphan and lost messages ensures correctness of computation. Also, the proposed algorithm can be easily extended to handle multiple cluster failures. For this purpose we just need to have a watchdog process that will select the initiator cluster from the faulty clusters after their recovery. Then the algorithm can be executed by this initiator cluster.

Finally, note that we have not done any experiment, because existing tools are not sufficient to implement the algorithm. As a result, a large amount of additional work will be required for its implementation.

References

[1] J. CAO ET AL., Y. HE, Checkpointing in Hybrid Distributed Systems. *Proceedings of the 7th Intl. Symposium on Parallel Architectures, Algorithms and Networks (ISPAN'04)*, Hong Kong, China, (2004), pp. 136–141.

[2] S. MONNET ET AL., Hybrid Checkpointing for Parallel Applications in Cluster Federations. *Proceedings of the 4th IEEE/ACM Intl. Symposium on Cluster Computing and the Grid*, Chicago, USA, (2004), pp. 773–782.

[3] B. GUPTA ET AL., A Low-Overhead Non-Blocking Checkpointing Algorithm for Mobile Computing Environment. *Springer Verlag Lecture Notes in Computer Science*, 3947 (2006), pp. 597–608.

[4] R. KOO, S. TOUEG, Checkpointing and Rollback-Recovery for Distributed Systems. *IEEE Transactions on Software Engineering*, SE-13 (1) (1987), pp. 23–31.

[5] Y.-M. WANG, Consistent Global Checkpoints That Contain a Given Set of Local Checkpoints. *IEEE Transactions on Computers*, **46**(4) (1997), pp. 456–468.

[6] J. TSAI, S.-Y. KUO, Y.-M. WANG, Theoretical Analysis for Communication-Induced Checkpointing Protocols with Rollback Dependency Trackability. *IEEE Transactions on Parallel and Distributed Systems*, **9**(10) (1998), pp. 963–971.

[7] B. GUPTA ET AL., Design of New Roll-Forward Recovery Approach for Distributed Systems. *IEE Proceedings – Computers and Digital Techniques*, **149**(3) (2002), pp. 105–112.

[8] D. MINIVANAN, M. SINGHAL, Asynchronous Recovery Without Using Vector Timestamps. *Journal of Parallel and Distributed Computing*, **62** Issue 62 (2002), pp. 1695–1728.

[9] X. QI ET AL., An Efficient End-Host Architecture for Cluster Communication. *Proceedings of 2004 IEEE Intl. Conference on Cluster Computing*, San Diego, USA, (2004), pp. 83–92.

[10] M. SINGHAL, N. G. SHIVARATRI, *Advanced Concepts in Operating Systems*, McGraw-Hill, Inc., (1994).

[11] E. N. ELNOZAHY ET AL., The Performance of Consistent Checkpointing. *Proceedings of the 11th Symposium on Reliable Distributed Systems*, (1992), pp. 86–95.

[12] G. CAO, M. SINGHAL, On Coordinated Checkpointing in Distributed Systems. *IEEE Transactions on Parallel and Distributed Systems*, **9**(12) (1998), pp. 1213–1225.

[13] D. B. JOHNSON, W. ZWAENEPOEL, Sender-Based Message Logging. *Proceedings of the 17th Intl. Symposium on Fault Tolerant Computing Systems*, Pittsburgh, (1987), pp. 14–19.

[14] M. L. POWELL, D. L. PRESOTTO, Publishing: A Reliable Broadcast Communication Mechanism. *Proceedings of the 9th ACM Symposium on Operating Systems*, (1983), pp. 100–109.

[15] L. ALVISI, K. MARZULLO, Message Logging: Pessimistic, Optimistic, and Causal. *Proceedings of the 15th IEEE Intl. Conference on Distributed Computing Systems*, (1995), pp. 229–236.

[16] K. M. CHANDY, L. LAMPORT, Distributed Snapshots: Determining Global States of Distributed Systems. *ACM Transactions on Computing Systems*, **3**(1) (1985), pp. 63–75.

[17] B. GUPTA ET AL., A Novel Low-Overhead Roll-Forward Recovery Scheme for Distributed Systems. *IET Computers and Digital Techniques*, **1**(4) (2007), pp. 97–404.

[18] G. CAO, M. SINGHAL, Mutable Checkpoints: A New Checkpointing Approach for Mobile Computing Systems. *IEEE Transactions on Parallel and Distributed Systems*, **12**(2) (2001), pp. 157–172.

[19] S. VENKATESAN, T. T.-Y. JUANG, S. ALAGAR, Optimistic Crash Recovery Without Changing Application Messages. *IEEE Transactions on Parallel and Distributed Systems*, **8**(3) (1997), pp. 263–271.

[20] T.-Y. JUANG, S. VENKATESAN, Efficient Algorithm for Crash Recovery in Distributed Systems. *Proceedings of the 10th Conference on Foundations on Software Technology and Theoretical Computer Science*, (1990), pp. 49–361.

[21] P. LEU, B. BHARGAVA, Concurrent Robust Checkpointing and Recovery in Distributed Systems. *Proceedings of the 4th International Conference on Data Engineering*, (1988), pp. 154–163.

[22] J. MCDERMID, Checkpointing and Error Recovery in Distributed Systems. *Proceedings of the 2nd International Conference on Distributed Computing Systems*, (1982), pp. 271–282.

[23] R. E. STROM, S. YEMINI, Optimistic Recovery in Distributed Systems, *ACM Transactions on Computing Systems*, **3**(3) (1985), pp. 204–226.

[24] K. VENKATESH, T. RADHAKRISHNAN, H. F. LI, Optimal Checkpointing and Local Recording for Domino-Free Rollback Recovery. *Information Processing Letters*, **5**(25) (1987), pp. 295–304.

[25] D. MANIVANNAN, M. SINGHAL, Quasi-synchronous Checkpointing: Models, Characterization, and Classification. *IEEE Transactions on Parallel and Distributed Systems*, **10**(7) (1999), pp. 703-713.

[26] R. BALDONI, J. M. HELWAY, A. MOSTERFAOUI, M. RAYNAL, A Communication-induced Checkpointing Protocol that Ensures Rollback Dependency Trackability. *Proceedings of IEEE International Symposium on Fault-tolerant Computing*, (1997), pp. 68–77.

[27] R. BALDONI, F. QUAGLIA, P. FORNARA, An Indexed-based Checkpointing Algorithm for Autonomous Distributed Systems, *IEEE Transactions on Parallel and Distributed Systems*, **10**(2) (1999), pp. 181–192.

[28] O. P. DAMINI, V. K. GARG, How to Recover Efficiently Asynchronously When Optimism Fails. *Proceedings of the 16th International Conference on Distributed Computing Systems*, (1996), pp. 108–115.

Received: June, 2009
Revised: June, 2011
Accepted: June, 2011

Contact addresses:

Bidyut Gupta
Department of Computer Science
Southern Illinois University
Carbondale, IL 62901
USA
e-mail: bidyut@cs.siu.edu

Ruslan Nikolaev
Department of Computer Science
Southern Illinois University
Carbondale, IL 62901
USA
e-mail: rnikola@siu.edu

Raja Chirra
Department of Computer Science
Southern Illinois University
Carbondale, IL 62901
USA
e-mail: rchirra@siu.edu

BIDYUT GUPTA received his M.Tech. and Ph.D. degrees from the University of Calcutta, India. At present he is a Professor at the Department of Computer Science, Southern Illinois University in Carbondale. His research interests include secured data communication, mobile and distributed computing. He is a senior member of IEEE.

RUSLAN NIKOLAEV received his MS degree in computer science from the Department of Computer Science, Southern Illinois University in Carbondale. At present he is a doctoral student at Virginia Tech., USA.

RAJA CHIRRA received his MS degree in computer science from the Department of Computer Science, Southern Illinois University in Carbondale. At present he is working as a software engineer.