On Composition of Checkpoint and Recovery Protocols for Distributed Systems

Soumi Chattopadyay[1], Ansuman Banerjee[1], and Himadri Sekhar Paul[2(✉)]

[1] ACMU, Indian Statistical Institute, Kolkata, India
{soumi_r,ansuman}@isical.ac.in
[2] Innovation Labs, TCS Ltd., Kolkata, India
HimadriSekhar.Paul@tcs.com

Abstract. Rollback recovery has been studied as a low-cost fault tolerance mechanism for ensuring dependability of critical distributed applications. There is a rich variety of recovery protocols proposed in literature and they are broadly classified as *checkpoint-based recovery protocols* and *message-log based recovery protocols*. In this paper we attempt to model composition of protocol and check whether such composition is consistent with recovery. The composition of protocols in important in a system whether resources are hierarchically organized, for example grid or cloud systems.

1 Introduction

Distributed systems have always been looked at as a powerful computation platform for addressing and mitigating the resource limitations of a single-node system in terms of compute power or available memory. Over the past few years, distributed systems have evolved into the grid and then the cloud, imparting more control and ease of access to such distributed infrastructures. Such infrastructures have often been defined as a federation of resources from multiple administrative domains.

A distributed system like a grid or cloud, which is termed here as a *global infrastructure* or simply a *global system*, is composed of multiple smaller distributed installations, termed here as *native systems* or *native clusters*. Each native system is a conglomeration of basic resources, like computing nodes. Typically a native cluster is located in a constrained geography, for example, in the department of some university. But the clusters themselves can be geographically dispersed. Although the resources are managed in a hierarchical manner, the whole system is presented as a single system to the user. At the same time, it is important to maintain administrative segregation among the participating native systems to honor their own administrative policies. To maintain a single system view of the global infrastructure, it becomes important that some of the system level services of the constituent native systems may need to be combined to provide a system-wide, single, consistent service.

Fault tolerance remains a critical aspect of such systems and is being addressed in various ways depending on the services they provide. In this paper,

A. Norta et al. (Eds.): ICSOC 2015 Workshops, LNCS 9586, pp. 231–242, 2016.
DOI: 10.1007/978-3-662-50539-7_19

we assume that the native clusters employ some rollback recovery mechanisms as fault tolerance. These native recovery protocols are combined by some global recovery protocol to ensure system-wide consistent recovery. In this paper, we investigate the problem of composing different recovery protocols in a cluster-based distributed system. Section 2 presents discussions on definitions and models related to distributed systems. In Sect. 3, we present a model of a recovery protocol. Analysis of composition consistency of recovery protocols based on this model is presented in Sect. 4. Finally, Sect. 5 concludes this paper with directions for future work.

2 Background

A distributed system is a collection of independent resource nodes connected over a network. Since node failures and communication link malfunctions are assumed to be independent events, larger the system - higher is the probability of failure. Fault tolerance is, therefore, an important issue in distributed systems and is critically linked to reliability, availability, and throughput. Fault tolerance, however, can only be achieved through some form of redundancy. There are different levels of guarantees for different forms of redundancy and with different associated cost. Rollback recovery is a form of temporal redundancy which has been proposed as a cost-effective mechanism of fault tolerance for non-critical distributed applications. The following section presents a model of distributed systems which we use for the illustration of our key concepts.

2.1 Model of Distributed System

A distributed system consists of a set of n processes $\mathcal{P} = \{P_1, P_2, \ldots P_n\}$. The processes are connected by underlying communication channels, $\mathcal{C} = \{C_{ij} \mid 1 \leq i, j \leq n\}$, where channel $C_{i,j}$ denotes a uni-directional channel from P_i to P_j. Execution of a single process P_i is modelled as a state transition system $\mathcal{S}_i$. State transitions are triggered by events. Events in distributed systems are classified as two types: (1) *local events* like computation events, message send events; and (2) *external events*, like message receive events. A *local checkpoint* of a process P_i is a concrete realization of its execution state.

The execution model of the distributed system is the composition of the state transition systems of the constituent processes, *i.e.* $\mathbf{S} = \mathcal{S}_1 \times \mathcal{S}_2 \times \cdots \times \mathcal{S}_n$. The global state of a distributed system is represented as the tuple $DS = (SP, SC)$, where $SP = \{s_i | s_i$ is the state information for $p_i \in P\}$ and $SC = \{c_{ij} | 1 \leq i, j \leq n\}$ where, c_{ij} is a set of in-transit messages from p_i to p_j. A *global checkpoint* is a composition of local checkpoints, one taken from each of the processes and also the state of the channels, which essentially represent the in-transit messages in the channels.

A recovery process uses information recorded during the failure-free execution of the system to construct a consistent global state. The necessary condition that a recovery is consistent is that, for every message received by some process in the

system, the corresponding send events are recorded in the state of the sender. In the next section, we present a model of consistent recovery, which we use for protocol composition.

3 Recovery Consistency Model

Protocols that employ temporal redundancy for fault tolerance, can be broadly classified into two categories, namely checkpoint-based rollback recovery and message-log-based roll-forward recovery. Checkpoint-based recovery ensures *roll-back* of system state to some past consistent state. Message-log based recovery uses *roll-forward* in conjunction with rollback recovery, where the processes are assumed to follow the Piece-Wise Deterministic (PWD) model [12]. According to the model, execution of a process is divided into series of deterministic executions, each terminated by a non-deterministic event or external event, like a message receive event. For example, Fig. 1 depicts one execution instance where a process has passed through the states s_1, s_2, s_3, s_4, and s_5 in sequence. The state transitions were triggered by events e_1, e_2, e_3, and e_4 in that order. Events $e_1 = receive(m_1)$ and $e_2 = receive(m_2)$ are non-deterministic events. The deterministic executions are highlighted with rectangular boxes. The non-deterministic events are logged and are replayed during recovery. In message passing distributed systems, the log consists of messages. We now define an operation PWD which, given the local state of a process and a set of non-deterministic event logs, determines the state at which the process can finally be rolled forward following the PWD model.

Definition 1. Reachable by PWD*: The reachability of a state s_s with respect to an event log M in the PWD model is defined as $PWD(s_s, M) = s_f$ where, s_s is the start state, s_f is the final state reached by PWD model from s_s by replaying events from M.*

We also use an alternative notation $s_s \rightsquigarrow^M s_f$ for $PWD(s_s, M) = s_f$. If a process is not assumed to follow PWD model, we say $PWD(s, \bullet) = s$.

With reference to Fig. 1, consider the message log $M = \{m_1\}$. The state of the system can be rolled-forward from s_1 to s_2 by replaying the receive event e_1 from M. In fact, by PWD, the system can be rolled forward up to s_4, but not beyond s_4 since $m_2 \notin M$. Hence $s_1 \rightsquigarrow^M s_4$, but $s_1 \not\rightsquigarrow^M s_5$.

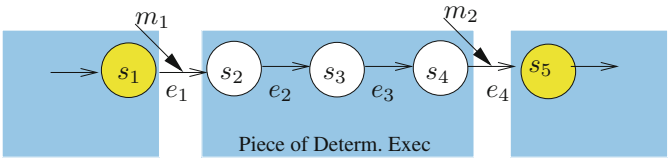

Fig. 1. Roll-forward by PWD model

3.1 Consistency of Global Recovery

In the event of a failure, processes need to be restarted. However, we can start processes from some intermediate state instead of re-starting them from their initial states. A global checkpoint represents such a state of the system and the recovery protocol computes such a state from available local checkpoints of the processes. However, any composition of local states does not guarantee consistency of the global checkpoint. In this section, we derive consistency conditions which are applicable for both checkpoint-based recovery protocols, as well as log-based recovery protocols. The essential idea of consistency is to capture the condition that no message becomes *orphan* after completion of the recovery process. We formalize the concept of the orphan message in the discussion below. This condition for consistency of a global checkpoint is a necessary condition. However, in an environment where the underlying communication channel is reliable, the recovery process must also ensure that there is no *lost message*, which essentially means that for every message whose send event is captured in the log, the corresponding receive event is present as well. Based on this discussion, the condition of consistency can be derived as follows.

No Orphan Message: Given local states or checkpoints of a pair of processes p_s and p_r, denoted as c_s and c_r respectively, and a message m whose sender is the process p_s and the receiver is p_r; the message is said to be *orphan w.r.t.* the states $\{c_s, c_r\}$, if the receive event is recorded in c_r but the corresponding send event is not recorded in c_s. In a consistent recovery, there cannot be any orphan message present in the recovered global state. The condition is defined here as *No Orphan Message* (NOM). Given a global state $DS = (SP, SC)$ as defined earlier, for every pair of local checkpoints, $SP_i, SP_j \in DS$, the NOM criterion denotes *either* there is no orphan message for (SP_i, SP_j) *or* P_i can be rolled forward from SP_i to a state s such that there is no orphan message for the pair $< s, SP_j >$. Formally, this is defined as,

Definition 2. *Given a global checkpoint $\mathfrak{C} = (SP, SC)$, No Orphan Message (NOM) constraint for a pair of local checkpoints, $SP_i, SP_j \in SP$ is defined as,*

$$NOM(SP_i, SP_j) = \begin{cases} true & if\ \forall receive(m) \in SP_j : \\ & \quad (send(m) \in PWD\,(SP_i, M_i)) \\ false & otherwise \end{cases}$$

where, M_j is the event log of receive messages at process P_j, $send(m)$ denotes the send event of a message m, and $receive(m)$ denotes the receive event of a message m,

Definition 3. *No Orphan Message constraint for a global checkpoint, $\mathfrak{C}$, is defined as:*

$$NOM(\mathfrak{C}) = \bigwedge_{\forall_{i,j}} NOM(SP_i, SP_j) \quad | \quad SP_i, SP_j \in \mathfrak{C}$$

No Lost Message. Given local states or checkpoints of a pair of processes p_s and p_r, denoted as c_s and c_r respectively, and a message m whose sender is the process p_s and the receiver is p_r; the message is said to be *lost w.r.t.* the states $\{c_s, c_r\}$, if the send event is recorded in c_r but the corresponding receive event is not recorded in c_s. In a system, where the underlying communication channels are reliable, a recovery protocol must ensure that there are no lost messages in the recovered state to ensure consistency. Given a global state $DS = (SP, SC)$ as defined earlier, the condition of *No Lost Message* (NLM) is described for a pair of local checkpoints $SP_i, SP_j \in DS$ as follows:

- *Either,* there is no lost message for (SP_i, SP_j)
- *Or,* for every in-transit message m from P_i to P_j, $m \in SC_{ij}$, where $SC_{ij} \in DS$
- *Or,* P_j can be rolled forward to a state s such that for the state pair $< SP_i, s >$, there is no lost message or every in-transit message is included in the channel state SC_{ij}.

Definition 4. *Given a global checkpoint $\mathfrak{C} = (SP, SC)$, the No Message Loss constraint for a pair of checkpoints $SP_i, SP_j \in SP$, can be defined as follows:*

$$NLM(SP_i, SP_j) = \begin{cases} true & if \quad \forall send(m) \in SP_i : \\ & \quad \Big(receive(m) \in PWD(SP_j, M_j) \\ & \quad\quad \oplus \quad (m \in SC_{ij} \mid SC_{ij} \in SC)\Big) \\ false & otherwise \end{cases}$$

Definition 5. *No Lost Message (NLM) constraint for the global checkpoint, $\mathfrak{C}$, is defined as:*

$$NLM(\mathfrak{C}) = \bigwedge_{\forall_{i,j}} NLM(SP_i, SP_j) | SP_i, SP_j \in \mathfrak{C}$$

3.2 Consistency of Global Checkpoint

A failure free run of a distributed system is one of its many valid execution paths. Consistency of a global checkpoint ensures reachability of a global state which could have been reached by the system in some failure free run. However in none of these possible execution paths, an orphan message generation is possible. Consistency criterion of a global state must ensure that such conditions are not violated. Any arbitrary composition of local checkpoints into a global checkpoint does necessarily make it consistent. We now define the consistency criterion for a global checkpoint.

Definition 6. *Given a global checkpoint $\mathfrak{C} = (SP, SC)$*

$$consistent(\mathfrak{C}) \Leftrightarrow \begin{cases} NOM(SP_p, SP_q) & \forall SP_p, SP_q \in SP \\ & \ldots when\ the\ channels\ are\ unreliable \\ NOM(SP_p, SP_q) \wedge NLM(SP_p, SP_q) & \forall SP_p, SP_q \in SP \\ & \ldots when\ the\ channels\ are\ reliable \end{cases}$$

Intuitively, the definition above signifies that a collection of local checkpoints, as a global checkpoint, is consistent only when there is no orphan message, given any pair of local checkpoints from this collection. However, if the underlying communication channels are reliable, the consistency criterion is more stringent and must additionally satisfy 'no lost message' constraint.

3.3 Fault and Recovery Model

We now present a discussion on the fault and recovery model of the system following Hoare's logical notations denoting logical derivation from a set of given expressions [4]. For example, $\frac{P \quad Q}{R}$ expression denotes logical derivation of the expression R from given expressions P and Q. Fault and recovery can be modelled as processes. A fault takes the system to an unknown, possibly inconsistent state. A fault can occur at any time, therefore,

$$\frac{\{*\}}{I\ \{F\}\ \blacksquare}$$

where I is a consistency invariant condition, F denotes the fault. We denote unknown or unspecified condition as $\blacksquare$. The numerator denoted by $*$ stands for the fact that the fault can manifest at any state, denoted by the symbol $*$.

The recovery process takes the system from an unspecified state to a state where the consistency criterion holds. We assume that no fault occurs during the recovery procedure. Therefore,

$$\blacksquare\ \{R\}\ I$$

where I is a consistency invariant condition as above, R denotes the recovery process.

In these cases, I can be the global state condition tuple $\{NOM, NLM\}$, denoted in our usual representation of (process state, channel state) when the underlying channels are lossless or $\{NOM\}$ when the underlying channels are lossy.

When a fault process is triggered, a recovery process must be subsequently triggered, and the objective of the recovery process is to restore the system to a consistent state. Therefore, under the composition operator ($\circ$), the final system remains in a consistent state.

$$\frac{I\ \{F\}\ \blacksquare, \quad \blacksquare\ \{R\}\ I}{I\ \{F \circ R\}\ I}$$

Assumption 1. *Given a consistent global state* $\mathfrak{C}$*, recovery with* $\mathfrak{C}$ *leads to a consistent recovery.*

$$\frac{consistent(\mathfrak{C})}{\blacksquare\ R(\mathfrak{C})\ I}$$

4 Hierarchical Protocol Analysis

A cluster-based distributed system is composed of several clusters of machines where the clusters may be geographically dispersed and administered by independent organizations. As a consequence, different clusters can employ different checkpoint and recovery protocols. Figure 2 shows a snapshot of execution of processes in two different clusters. The figure shows the communication pattern and local checkpoints of the processes. Each of the clusters employ checkpoint and recovery protocols independent of each other. For a simple case, we assume both the clusters employ the same protocol. We analyze the consistent recovery phenomenon under this most simple setup. We also present an analysis of scenarios involving two and three protocols being composed. Composition involving more than one protocol is more complex than composing two instances of the same protocol since the requirements of the individual checkpointing protocols and their corresponding recovery protocols may be very different.

In a general framework of hierarchical protocols, there are two key elements, a *global protocol* which interacts with nodes outside its own cluster and a *local protocol* which implements the native checkpoint and recovery protocol inside the cluster. Native protocols are well known protocols found in literature. The global protocol is used to combine natives protocols running in different clusters so that a consistent recovery can be ensured across the whole system. Each cluster may have a node specially designated as the *leader* node, which usually acts as the local coordinator whenever coordination is demanded by the global protocol.

4.1 Composition with Un-Coordinated Protocol

We begin our analysis with the simplest case of hierarchical composition of checkpoint and recovery protocols, where all the clusters in the system use a

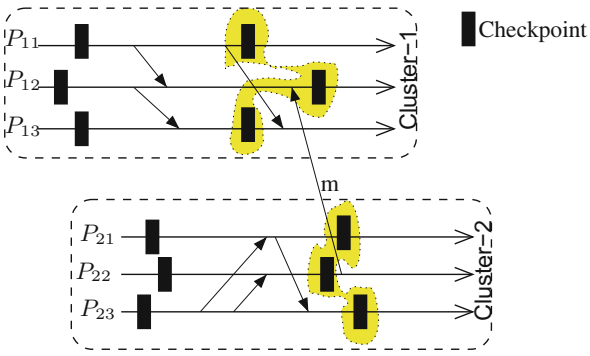

Fig. 2. Communication pattern involving two clusters

completely un-coordinated checkpointing protocol cite. The checkpointing protocol in this case is trivially composed where the global protocol has no role to play. However, the corresponding recovery protocol requires coordination among all nodes to construct a consistent global recovery state and this objective needs to be achieved by the global protocol. The global recovery line can be computed from partial Rollback Dependency Graphs (RDGs) maintained by all nodes in the system [13]. Algorithm 1 presents an outline of the global recovery protocol. The proof of the recovery is trivial and is guaranteed by the native recovery protocol and is not elaborated here.

Algorithm 1. Global Recovery Protocol

1: **procedure** GLOBALRECOVERYCOORD
2: The failed process initiates recovery by sending *Recovery Request Message* to all the *leader* processes in the system.
3: All *Leaders* on receipt of the *Recovery Request Message* broadcast the *Recovery Request Messages* to all nodes in its own cluster.
4: All nodes on receipt of the *Recovery Request Message* construct RDGs based on their partial view of the dependency of the checkpoints and send to the their leader process (native recovery procedure)
5: The *leader* combines the RDGs and sends to the failed processes
6: The failed process on receive of responses from all *leader* processes combines the RDGs and identifies the recovery line (native recovery process)
7: The failed process sends the recovery line to all *leader*s
8: The *leader* broadcasts the recovery line in its cluster
9: All nodes, on receipt of the recovery line roll back to the checkpoint identified in the recovery line and delete all local checkpoints beyond the recovery line (native recovery process)
10: **end procedure**

4.2 Composition with Coordinated Checkpointing Protocol

We address another scenario where all the clusters use a coordinated checkpoint and recovery protocol based on the two-phase commit protocol [3] as the native protocol. This allows them to work independently without any global protocol. It can be easily shown that this simple composition cannot guarantee a consistent recovery and a counter-example is depicted in Fig. 2 in support of this claim. The recovery lines in the individual clusters are highlighted. However, these two recovery lines are not consistent with each other due to the presence of the inter-cluster message m. If the global state recovers to the yellow recovery line highlighted in the figure, then m becomes an orphan message, *i.e.* $NOM(SP_{12}, SP_{22})$ does not hold, where SP_{12} and SP_{22} are the checkpoints of P_{12} and P_{22} respectively on the recovery line.

In general, due to consistency of native recovery protocols, $NOM(SP_{ki}, SP_{kj}) : 1 \leq i, j \leq n_k$ holds, where SP_{ki} and SP_{kj} are checkpoints of processes $P_{ki} \in K_k$ and $P_{kj} \in K_k$ respectively, K_k represents the k^{th} cluster, and n_k is the number

of nodes in K_k. However, trivially $NOM\,(SP_{ki}, SP_{lj})$ does not hold, where SP_{ki} and SP_{lj} are checkpoints on the recovery line of clusters K_k and K_l respectively. This violation is usually due to the inter-cluster message as depicted in the example above.

One possible method of composition to eliminate orphan inter-cluster messages is to impose coordination of checkpointing activities among clusters through a global protocol. However, this translates to the fact that all checkpointing and recovery activities become a global event and by the guarantees of the coordinated checkpointing protocol, the recovery is always consistent. Another way of composition is by selectively blocking clusters, during global level coordination. Paul *et al.* proposed this solution, where a process is blocked whenever it attempts to send an inter-cluster message during a global level coordination for checkpointing [10]. They implement the rule as *Policy B*, which, when in force, blocks the process on receipt of any inter-cluster message. The message can only be processed once the policy is revoked at the termination of global checkpoint coordination.

In the later sections, we explore more complex scenarios where multiple native checkpoint and recovery protocols are used among clusters.

4.3 Composition of Two Heterogeneous Protocols

In this section, we consider the composition problem in the context of two clusters, each employing its own checkpoint & recovery protocol. The case can be generalized to any number of clusters, where there are two different protocols being employed, with some processes running the first protocol and some running the second. We analyze the case of composition of coordinated checkpointing [5] and a pessimistic message-log-based recovery protocol [1]. Initially, we assume that they are trivially composed with no global protocol to coordinate activities between two different clusters. We have already shown that without global coordination, two clusters with coordinated checkpointing protocol cannot ensure consistent recovery. Also, without global coordination, a cluster with coordinated checkpointing cannot be consistent with the cluster employing message-log based recovery. Consider Fig. 2 and let us assume *cluster-1* employs a log-based recovery protocol and *cluster-2* employs coordinated checkpointing. In the event of a failure, by the native recovery protocol, the coordinated cluster rolls back to its previous checkpoint, while the log-based recovery protocol only restarts the failed process and rolls forward its state by replaying messages from its message log. Consider a failure in *cluster-2* which, by its native recovery protocol, rolls back to the state highlighted in the figure. However, the processes in *cluster-1* continue execution. Now, after recovery of the processes in *cluster-2*, it becomes inconsistent since $NOM\,(SP_{12}, SP_{22})$ does not hold due to the presence of m, which now is an orphan message.

In a cluster with log-based recovery protocol, only the failed process recovers by restart and roll-forward mechanism. No other process takes part in the recovery activity. So after recovery, by the guarantee of the native recovery process, the recovered process becomes consistent with the states of rest of the processes

in the system. In order to eliminate any possibility of generation of orphan message in the event of a rollback in a cluster with coordinated checkpointing, the global protocol must ensure that the cluster does not rollback beyond any send event of an inter-cluster message. A forced checkpoint in the clusters with coordinated checkpointing after every send event of an inter-cluster message, was proposed as a solution in [11]. Due to this enforcement by the global protocol, in the event of a recovery, a cluster with coordinated checkpoint never rolls back beyond any send event of an inter-cluster message. Therefore, after recovery $NOM\,(SP_{ki}, SP_{lj})$ holds, where SP_{ki} is the recovered state of a process in the k^{th} cluster and SP_{lj} is the same for the l^{th} cluster, and $k \neq l$. Again by the guarantees of native recovery protocol of a cluster, $NOM\,(SP_{ki}, SP_{kj})$ holds for intra-cluster messages. Therefore, NOM holds for any two processes in the system and hence the recovery is consistent.

4.4 Composition of Three Protocols

In this section, we attempt to compose three different protocols, namely coordinated checkpointing and recovery [5], receiver-based pessimistic message log-based recovery [1], and quasi-synchronous checkpointing protocol [6]. We assume each cluster employs one of these three types of protocol. The recovery consistency guarantee provided by the protocols are as follows:

$NOM\,(SC_{ki}, SC_{kj})$	k^{th} cluster employs coordinated checkpointing
$NOM\,(s_{li}, PWD(\ominus, M_{kj}))$	k^{th} cluster employs log-based recovery and $k \neq l$ $\ominus$ represents the restarted state
$NOM\,(SC_{ki}, SC_{kj})$	k^{th} cluster employs quasi-synchronous checkpointing

The composed global checkpoint and recovery protocol must ensure the NOM condition for local states of all processes after recovery. A failure in the cluster employing log-based recovery is trivially consistent with states of all other processes by the guarantee of the protocol. Since violations of NOM condition arises due to inter-cluster messages, processes in clusters employing coordinated checkpointing or quasi-synchronous checkpointing must force extra checkpoints so that these processes never roll back beyond the send event of any inter-cluster message. The message send procedure is shown in Algorithm 2.

Algorithm 2. Message Send Event Handler for Clusters with Coordinated and Quasi-Synchronous Checkpointing

```
procedure MsgSendHandler(m : Message to be send)
    Buffer m
    Initiate coordinated checkpointing protocol
    On completion of the native checkpointing successfully, release m to be
    sent through network.
end procedure
```

We now prove that the global protocol with the message handler described in Algorithm 2 ensures consistent recovery in all clusters.

Proof. We need to prove that after rollback, the states of the processes are consistent with respect to each other. There can be two cases for a pair of processes, (1) both belong to the same cluster, and (2) both belong to different clusters. For the first case, the consistency of the states of the processes are guaranteed by the consistency of the native recovery protocol. We prove the second case by contradiction. Let the processes be P_{ki} and P_{lj} and they belong to clusters C_k and C_l respectively and $k \neq l$. Let the states of the processes after recovery be s_{ki} and s_{lj} respectively. We assume $NOM(s_{ki}, s_{lj})$ does not hold. Without loss of generality, there must be an inter-cluster message m sent before s_{ki} and received before s_{lj}. There can be the following cases.

Case 1: C_l employs log-based protocol and C_k employs Coordinated checkpointing. During recovery, by native recovery protocols, P_{li} does not rollback and continues with its normal operation. P_{kj} recovers to its latest checkpoint. But, due to the Algorithm 2, there cannot be any send event of an inter-cluster message recorded after s_{kj}. Hence a contradiction.

Case 2: C_l employs log-based protocol and C_k employs Quasi-Synchronous checkpointing. Same argument as in case 1 holds.

Case 3: C_l and C_k both employ either Coordinated or Quasi-Synchronous checkpointing. By the application of Algorithm 2, there cannot be any inter-cluster message send event beyond the recovered state of either P_{ki} or P_{lj}. Hence a contradiction. □

4.5 Related Work

The first work towards composition of protocols for hierarchical distributed systems was by Paul *et al.* [9] where they demonstrated that cluster-based systems running a coordinated checkpointing protocol and a communication induced checkpointing (CIC) protocol can be combined by a higher level checkpoint and recovery protocol to provide a consistent recovery. Monet *et al.* worked in the same direction and presented a more detailed protocol where they combined clusters running coordinated checkpointing with the method of the CIC protocol as global protocol [7]. Paul *et al.* also proposed a method to combine a coordinated checkpointing protocol with a message logging protocol [11]. Bhatia *et al.* proposed a hierarchical causal logging protocol that addresses the scalability problems of causal logging [2]. Ndiaye *et al.* present a comparison of these protocols obtained through composition by simulation using OmNet+ [8]. To the best of our knowledge, a formal treatment of protocol composition as discussed in this paper, is missing in literature.

5 Conclusion

In this paper, we discuss a model of checkpoint and recovery protocols for message passing distributed systems and extracted consistency criterion of a recovery process. The model we present is extended for inclusion of message-log based

recovery protocols. The motivation behind the model is to apply the same for protocol compositions and deduce the consistency of the recovery. Such composition is meaningful in a cloud or grid computing scenario, where multiple clusters from multiple administrative domains participate in the system. We apply the model on simple compositions involving various checkpoint and recovery protocols. We believe that this study will open up future research avenues on more protocol variants.

References

1. Alvisi, L., Murzullo, K.: Message logging: pessimistic, optimistic, causal and optimal. IEEE Trans. Softw. Eng. **24**, 149–159 (1998)
2. Bhatia, K., Marzullo, K., Alvisi, L.: Scalable causal message logging for wide-area environments. Concurency Comput. Pract. Exp. **15**(3), 873–889 (2003)
3. Gray, J.N.: Notes on database operating systems. In: Bayer, R., Graham, R.M., Seegmüller, G. (eds.) Operating Systems. LNCS, vol. 60, pp. 393–481. Springer, Berlin (1978)
4. Hoare, C.A.R.: Communicating sequential processes. Commun. ACM **21**(8), 666–677 (1978)
5. Koo, R., Toueg, S.: Checkpointing and rollback-recovery for distributed systems. IEEE Trans. Softw. Eng. **SE–13**(1), 23–31 (1987)
6. Manivannan, D., Singhal, M.: A low-overhead recovery technique using quasi-synchronous checkpointing. In: Proceedings of the 16th International Conference on Distributed Computing Systems, pp. 100–107, May 1996
7. Monnet, S.: Hybrid checkpointing for parallel applications in cluster federations. In: Proceedings of 4th IEEE/ACM International Symposium on Cluster Computing and the Grid, pp. 773–782 (2004)
8. Ndiaye, N.M., Sens, P., Thiare, O.: Performance comparison of hierarchical checkpoint protocols grid computing. Intl. J. Interact. Multimedia Artif. Intell. **1**, 46–53 (2012)
9. Paul, H.S., Gupta, A., Badrinath, R.: Combining checkpoint and recovery algorithm for distributed systems. In: 3rd Workshop on Distributed Computing, pp. 68–72 (2001)
10. Paul, H.S., Gupta, A., Badrinath, R.: Hierarchical coordinated checkpointing protocol. In: IASTED PDCS, pp. 235–240 (2002)
11. Paul, H.S., Gupta, A., Badrinath, R.: A heterogeneous checkpoint and recovery protocol in cluster-based distributed systems. In: PDPTA, pp. 1224–1230 (2003)
12. Strom, R., Yemini, S.: Optimistic recovery in distributed systems. ACM Trans. Comput. Syst. **3**(3), 204–226 (1985)
13. Wang, Y.M.: Reducing message logging overhead for log-based recovery. In: Proceedings of IEEE International Symposium on Circuits and Systems, pp. 1925–1928 (1993)